

Specifying QITs using Containers

Stefania Damato Thorsten Altenkirch

University of Nottingham, UK

Conference on
Homotopy Type Theory

24th May 2023

Motivation

Gain a better understanding of QIITs.

Motivation

Gain a better understanding of QIITs.

QIITs = { **Quotient** inductive types allow set-truncated equality
constructors

Motivation

Gain a better understanding of QIITs.

QIITs = $\left\{ \begin{array}{l} \text{Quotient inductive types} \\ \text{Inductive-inductive types} \end{array} \right.$ allow set-truncated equality constructors
 $A : \text{Type}$
 $B : A \rightarrow \text{Type}$

Example 1.0: Intrinsic syntax of type theory

```
data Con : Set
data Ty  : Con → Set
data Sub : Con → Con → Set
data Tm  : (Γ : Con) → Ty Γ → Set

data Con where
  ◇ : Con
  --, _ : (Γ : Con) → Ty Γ → Con

data Ty where
  [-] : Ty Δ → Sub Γ Δ → Ty Γ
  U   : {Γ : Con} → Ty Γ
  E1  : {Γ : Con} → Tm Γ U → Ty Γ
  Π   : {Γ : Con} {A : Ty Γ} →
        Ty (Γ , A) → Ty Γ
  [id] : A [id] ≡ A
  [ ] [] : A [σ] [ν] ≡ A [σ ∘ ν]
  ...

data Sub where
  id : {Γ : Con} → Sub Γ Γ
  _◦_ : {Γ Δ Θ : Con} → Sub Θ Δ → Sub Γ Θ →
        Sub Γ Δ
  ε : {Γ : Con} → Sub Γ ◇
  --, _ : {Γ Δ : Con} {A : Ty Δ} (σ : Sub Γ Δ)
        → Tm Γ A [σ] → Sub Γ (Δ , A)
  π1 : {Γ Δ : Con} {A : Ty Δ} →
        Sub Γ (Δ , A) → Sub Γ Δ
  id◦ : id ∘ σ ≡ σ
  oid : σ ∘ id ≡ σ
  ...

data Tm where
  [-] : {Γ Δ : Con} {A : Ty Δ} {σ : Sub Γ Δ}
        → Tm Δ A → (σ : Sub Γ Δ) → Tm Γ A [σ]
  π2 : {Γ Δ : Con} {A : Ty Δ} →
        (σ : Sub Γ (Δ , A)) → Tm Γ A [π1 σ]
  lam : {Γ : Con} {A : Ty Γ} {B : Ty (Γ , A)} →
        Tm (Γ , A) B → Tm Γ (Π A B)
  app : {Γ : Con} {A : Ty Γ} {B : Ty (Γ , A)} →
        Tm Γ (Π A B) → Tm (Γ , A) B
  πβ : app (lam t) ≡ t
  ...
```

Motivation

Example 1.1: (Simplified) intrinsic syntax of type theory

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) (A : Ty Γ) → Con
  eq : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) →
        ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ι : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

Specifications of inductive types

Class of types	Functor type	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N} : \mathbf{Set}$	$\mathbf{Set} \rightarrow \mathbf{Set}$	initial algebras of endofunctors on \mathbf{Set}	containers	W-type
inductive families e.g. $\mathbf{Fin} : \mathbb{N} \rightarrow \mathbf{Set}$	$(\mathbf{I} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{I} \rightarrow \mathbf{Set})$	initial algebras of endofunctors on $\mathbf{Set}^{\mathbf{I}}$	indexed containers	WI-type

Specifications of inductive types

Class of types	Functor type	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N} : \mathbf{Set}$	$\mathbf{Set} \rightarrow \mathbf{Set}$	initial algebras of endofunctors on \mathbf{Set}	containers	W-type
inductive families e.g. $\mathbf{Fin} : \mathbb{N} \rightarrow \mathbf{Set}$	$(\mathbf{I} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{I} \rightarrow \mathbf{Set})$	initial algebras of endofunctors on $\mathbf{Set}^{\mathbf{I}}$	indexed containers	WI-type
QIITs e.g. $\mathbf{Con} : \mathbf{Set}$, $\mathbf{Ty} : \mathbf{Con} \rightarrow \mathbf{Set}$?	?	?	?

Specifications of inductive types

Class of types	Functor type	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N} : \mathbf{Set}$	$\mathbf{Set} \rightarrow \mathbf{Set}$	initial algebras of endofunctors on \mathbf{Set}	containers	W-type
inductive families e.g. $\mathbf{Fin} : \mathbb{N} \rightarrow \mathbf{Set}$	$(\mathbf{I} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{I} \rightarrow \mathbf{Set})$	initial algebras of endofunctors on $\mathbf{Set}^{\mathbf{I}}$	indexed containers	WI-type
QIITs e.g. $\mathbf{Con} : \mathbf{Set}$, $\mathbf{Ty} : \mathbf{Con} \rightarrow \mathbf{Set}$	we have an alternative representation	?	?	?

QIIT semantics [Altenkirch et al., 2018] by example

Idea: Start with the sorts, and add constructors one by one.

Example 1.1

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  --, - : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ℓ : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

QIT semantics [Altenkirch et al., 2018] by example

Idea: Start with the sorts, and add constructors one by one.

Example 1.1

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  --, - : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ℓ : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

① Category \mathbf{A}_0 of sorts. $\mathbf{A}_0 = \mathbf{Fam}(\mathbf{Set})$ has

- objects of type $\sum(C : \mathbf{Set})(T : C \rightarrow \mathbf{Set})$,
- morphisms $(C, T) \rightarrow (C', T')$ are functions $f : C \rightarrow C'$ and $g : (c : C) \rightarrow T c \rightarrow T' (f c)$.

QIT semantics [Altenkirch et al., 2018] by example

Example 1.1

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ι : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

② Adding constructor $\diamond : \mathbf{Con}$.

$$L_0 : \mathbf{A}_0 \rightarrow \mathbf{Set}$$

$$L_0(C, T) := \mathbf{1}$$

$$R_0 : \int L_0 \rightarrow \mathbf{Set}$$

$$= \sum \sum (C : \mathbf{Set}) (T : C \rightarrow \mathbf{Set}) (L_0(C, T)) \rightarrow \mathbf{Set}$$

$$R_0(C, T, \star) := C$$

Example 1.1

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ι : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

3 Category of algebras \mathbf{A}_1 has

- objects of type
$$\sum \sum (C : \text{Set}) (T : C \rightarrow \text{Set}) (e : (x : L_0(C, T)) \rightarrow R_0(C, T, x))$$
$$\cong \sum \sum (C : \text{Set}) (T : C \rightarrow \text{Set}) (e : C)$$
- morphisms $(C, T, e) \rightarrow (C', T', e')$ are functions $f : C \rightarrow C'$ and $g : (c : C) \rightarrow T c \rightarrow T' (f c)$ such that $e' \equiv f(e)$.

Example 1.1

```
data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ι : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
```

- ④ Adding constructor $_,_ : (\Gamma : \text{Con}) (A : \text{Ty } \Gamma) \rightarrow \text{Con}$.

$$L_1 : \mathbf{A}_1 \rightarrow \mathbf{Set}$$

$$L_1(C, T, e) := \sum(\Gamma : C)(T \Gamma)$$

$$R_1 : \int L_1 \rightarrow \mathbf{Set}$$

$$= \sum \sum \sum (C : \text{Set})(T : C \rightarrow \text{Set})(e : C)(L_1(C, T, e)) \rightarrow \mathbf{Set}$$

$$R_1(C, T, e, \Gamma, A) := C$$

Example 1.1

```

data Con : Set
data Ty  : Con → Set

data Con where
  ◇ : Con
  _,_ : (Γ : Con) (A : Ty Γ) → Con
  eq  : (Γ : Con) (A : Ty Γ) (B : Ty (Γ , A)) → ((Γ , A) , B) ≡ (Γ , Σ Γ A B)

data Ty where
  ι : (Γ : Con) → Ty Γ
  Σ : (Γ : Con) (A : Ty Γ) → Ty (Γ , A) → Ty Γ
  
```

5 Category of algebras \mathbf{A}_2 has

- objects of type

$$\begin{aligned}
 & \sum \sum \sum (C : \text{Set}) (T : C \rightarrow \text{Set}) (e : C) (ex : (x : L_1(C, T, e)) \rightarrow \\
 & R_1(C, T, e, x)) \\
 & \cong \sum \sum \sum (C : \text{Set}) (T : C \rightarrow \text{Set}) (e : C) (ex : \sum (\Gamma : C) (T \Gamma \rightarrow \\
 & C)
 \end{aligned}$$

- morphisms $(C, T, e, ex) \rightarrow (C', T', e', ex')$ are functions $f : C \rightarrow C'$ and $g : (c : C) \rightarrow T c \rightarrow T' (f c)$ such that $e' \equiv f(e)$ and $ex' \circ (f \Gamma, g \Gamma t) \equiv f \circ ex(\Gamma, t)$,

and so on.

Specification of inductive types, revisited

Class of types	Representation	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N} : \mathbf{Set}$	functor $\mathbf{Set} \rightarrow \mathbf{Set}$	initial algebras of endofunctors on \mathbf{Set}	containers	W-type
inductive families e.g. $\mathbf{Fin} : \mathbb{N} \rightarrow \mathbf{Set}$	functor $(\mathbf{I} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{I} \rightarrow \mathbf{Set})$	initial algebras of endofunctors on $\mathbf{Set}^{\mathbf{I}}$	indexed containers	WI-type
QITs e.g. $\mathbf{Con} : \mathbf{Set}$, $\mathbf{Ty} : \mathbf{Con} \rightarrow \mathbf{Set}$	sequence of functors L_n and R_n and sequence of categories of dialgebras	initial object in last constructed category of dialgebras \mathbf{A}_n	?	?

Specification of inductive types, revisited

Class of types	Representation	Category theory semantics	Type theoretic normal form	Universal type
ordinary inductive types e.g. $\mathbb{N} : \mathbf{Set}$	functor $\mathbf{Set} \rightarrow \mathbf{Set}$	initial algebras of endofunctors on \mathbf{Set}	containers	W-type
inductive families e.g. $\mathbf{Fin} : \mathbb{N} \rightarrow \mathbf{Set}$	functor $(\mathbf{I} \rightarrow \mathbf{Set}) \rightarrow (\mathbf{I} \rightarrow \mathbf{Set})$	initial algebras of endofunctors on $\mathbf{Set}^{\mathbf{I}}$	indexed containers	WI-type
QITs e.g. $\mathbf{Con} : \mathbf{Set}$, $\mathbf{Ty} : \mathbf{Con} \rightarrow \mathbf{Set}$	sequence of functors L_n and R_n and sequence of categories of dialgebras	initial object in last constructed category of dialgebras \mathbf{A}_n	representations constructed via generalised containers	? (QW-type)

Our contribution: restricting L_n and R_n

We require $L_n: \mathbf{A}_n \rightarrow \mathbf{Set}$ and $R_n: \int L_n \rightarrow \mathbf{Set}$ to be **generalised container functors** (+ other restrictions on R_n).

Our contribution: restricting L_n and R_n

We require $L_n: \mathbf{A}_n \rightarrow \mathbf{Set}$ and $R_n: \int L_n \rightarrow \mathbf{Set}$ to be **generalised container functors** (+ other restrictions on R_n).

Definition

A *generalised container* $S \triangleleft P$ over a category \mathbf{C} is a pair $S : \mathbf{Set}$ and $P : S \rightarrow |\mathbf{C}|$.

Definition

The *generalised container extension functor* associated to $S \triangleleft P$ and having type $\mathbf{C} \rightarrow \mathbf{Set}$, is defined by

$$\llbracket S \triangleleft P \rrbracket X := \sum (s : S)(\mathbf{C}(P s, X))$$

on objects $X : |\mathbf{C}|$.

Our contribution: restricting L_n and R_n

$S_n : \mathbf{Set}$

$P_n : S_n \rightarrow |\mathbf{A}_n|$

$Q_n^X : S_n \rightarrow |\mathbf{A}_n|$

$Q_n^f : (s : S_n) \rightarrow \mathbf{A}_n(P_n s, Q_n^X s)$

$L_n : \mathbf{A}_n \rightarrow \mathbf{Set}$

$L_n X \cong \llbracket S_n \triangleleft P_n \rrbracket X = \sum (s : S_n) (f : \mathbf{A}_n(P_n s, X))$

$R_n : \int L_n \rightarrow \mathbf{Set}$

$R_n (X, (s, f)) \cong \llbracket S_n \triangleleft Q_n \rrbracket (X, (s, f))$

$\cong \sum (h : \mathbf{A}_n(Q_n^X s, X)) (h \circ Q_n^f s = f)$

Our contribution: restricting L_n and R_n

$S_n : \mathbf{Set}$

$P_n : S_n \rightarrow |\mathbf{A}_n|$

$Q_n^X : S_n \rightarrow |\mathbf{A}_n|$

$Q_n^f : (s : S_n) \rightarrow \mathbf{A}_n(P_n s, Q_n^X s)$

$L_n : \mathbf{A}_n \rightarrow \mathbf{Set}$

$L_n X \cong \llbracket S_n \triangleleft P_n \rrbracket X = \sum (s : S_n)(f : \mathbf{A}_n(P_n s, X))$

$R_n : \int L_n \rightarrow \mathbf{Set}$

$R_n(X, (s, f)) \cong \llbracket S_n \triangleleft Q_n \rrbracket(X, (s, f))$

$\cong \sum (h : \mathbf{A}_n(Q_n^X s, X))(h \circ Q_n^f s = f)$

Discussion and future work





- QIITs combine set-truncated equalities with induction-induction.
- We can represent QIITs semantically as initial dialgebras.
- We propose a **specification of strictly positive QIITs**, which is more semantic than the one given by [Kaposi et al., 2019].
- What does a **universal QW-type** look like? One proposal given by [Fiore et al., 2021].

Discussion and future work

- QIITs combine set-truncated equalities with induction-induction.
- We can represent QIITs semantically as initial dialgebras.
- We propose a **specification of strictly positive QIITs**, which is more semantic than the one given by [Kaposi et al., 2019].
- What does a **universal QW-type** look like? One proposal given by [Fiore et al., 2021].

Thank you!

References

-  Altenkirch, T., Capriotti, P., Dijkstra, G., Kraus, N., and Nordvall Forsberg, F. (2018).
Quotient inductive-inductive types.
In Baier, C. and Dal Lago, U., editors, *FoSSACS*, pages 293–310. Springer.
-  Altenkirch, T. and Kaposi, A. (2021).
A container model of type theory.
In *TYPES 2021*.
-  Fiore, M. P., Pitts, A. M., and Steenkamp, S. (2021).
Quotients, inductive types, and quotient inductive types.
Log. Methods Comput. Sci., 18.
-  Kaposi, A., Kovács, A., and Altenkirch, T. (2019).
Constructing quotient inductive-inductive types.
Proc. ACM Program. Lang., 3(POPL).