# Induction-Induction Part 2
## Specifying quotient inductive-inductive types

Fredrik Nordvall Forsberg
University of Strathclyde, Glasgow

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus

TYPES 2018, Braga, 21 June 2018

# What is equality?

# What is equality?

In Intensional Martin-Löf Type Theory [Martin-Löf 1972]:

- Equality type is smallest reflexive relation.
- In other words, equality type characterises judgemental equality.
- But judgemental equality is machine-checkable, so bound to be disappointing for humans.

# What is equality?

In Intensional Martin-Löf Type Theory [Martin-Löf 1972]:

- Equality type is smallest reflexive relation.
- In other words, equality type characterises judgemental equality.
- But judgemental equality is machine-checkable, so bound to be disappointing for humans.

In Observational Type Theory [Altenkirch, McBride 2006]:

- Equality characterises *observable* behaviour rather than intensionally same construction.
- Well-behaved computational properties are retained by making equality proof-irrelevant.

# What is equality?

In Intensional Martin-Löf Type Theory [Martin-Löf 1972]:

- Equality type is smallest reflexive relation.
- In other words, equality type characterises judgemental equality.
- But judgemental equality is machine-checkable, so bound to be disappointing for humans.

In Observational Type Theory [Altenkirch, McBride 2006]:

- Equality characterises *observable* behaviour rather than intensionally same construction.
- Well-behaved computational properties are retained by making equality proof-irrelevant.

In Homotopy Type Theory [Awodey, Warren 2009; Voevodsky 2010]:

- Homotopical models suggest that equality can be given much more intricate *proof-relevant* structure.
- Equality type $\equiv_A$ provides access to this structure, and is morally part of $A$ (cf. `cubicaltt` [Cohen, Coquand, Huber, Mörtberg 2015]).

# Higher Inductive Types

Inductive Types freely given by:

- value constructors (constructs elements)

# Higher Inductive Types

Higher Inductive Types freely given by:

- value constructors (constructs elements)
- equality constructors (constructs equalities)

# Higher Inductive Types

Higher Inductive Types freely given by:

- value constructors (constructs elements)
- equality constructors (constructs equalities)

Applications:

- Synthetic homotopy theory:
  - Definition of the circle $\mathbb{S}^1$, with $\pi_1(\mathbb{S}^1) = \mathbb{Z}$,
  - Higher spheres $\mathbb{S}^n$,
  - The Hopf fibration, ...
- Quotidian applications:
  - Cauchy Reals $\mathbb{R}_\mathsf{c}$,
  - the Partiality monad $(-)_\perp$,
  - Type Theory in Type Theory.

# Quotient Inductive-Inductive Types

Quotient Inductive Types (QITs): HITs with trivial higher structure (set-level HITs). [Hofmann 1995]

# Quotient Inductive-Inductive Types

Quotient Inductive Types (QITs): HITs with trivial higher structure (set-level HITs). [Hofmann 1995]

Inductive-inductive types (IITs): inductive types depending on each other, e.g.

$$A : \mathsf{Set} \qquad B : A \to \mathsf{Set}$$

mutually defined, both defined by induction. [N.F. 2013]

# Quotient Inductive-Inductive Types

Quotient Inductive Types (QITs): HITs with trivial higher structure (set-level HITs). [Hofmann 1995]

Inductive-inductive types (IITs): inductive types depending on each other, e.g.

$$A : \mathsf{Set} \qquad B : A \to \mathsf{Set}$$

mutually defined, both defined by induction. [N.F. 2013]

Quotient Inductive-Inductive Types (QIITs): QITs + IITs = QIITs.

All quotidian applications of HITs are QIITs.

# Type theory in type theory as a QIIT

Simplified adaption after Altenkirch and Kaposi [2016]:

$$\textbf{data}\ \mathsf{Con} : \mathsf{Set}$$
$$\textbf{data}\ \mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$$

$$\varepsilon : \mathsf{Con}$$
$$\mathsf{ext} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$
$$\mathsf{U} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma$$
$$\sigma : (\Gamma : \mathsf{Con}) \to (\mathtt{A} : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A) \to \mathsf{Ty}\,\Gamma$$
$$\sigma_{eq} : (\Gamma : \mathsf{Con}) \to (\mathtt{A} : \mathsf{Ty}\,\Gamma) \to (\mathtt{B} : \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A)) \to (\mathsf{ext}\,(\mathsf{ext}\,\Gamma\,\mathtt{A})\,\mathtt{B} \equiv_{\mathsf{Con}} \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,\mathtt{A}\,\mathtt{B}))$$

# Challenging features

- Constructors for Con refer to Ty (and vice versa):

$$\mathsf{ext} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$

# Challenging features

- Constructors for Con refer to Ty (and vice versa):

$$\mathsf{ext} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$

- Constructors refer to previous constructors in their type:

$$\sigma : (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A) \to \mathsf{Ty}\,\Gamma$$

# Challenging features

- Constructors for Con refer to Ty (and vice versa):

$$\mathsf{ext} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$$

- Constructors refer to previous constructors in their type:

$$\sigma : (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A) \to \mathsf{Ty}\,\Gamma$$

- "Path constructors" construct equalities, not elements:

$$\sigma_{eq} : (\Gamma : \mathsf{Con}) \to (A : \mathsf{Ty}\,\Gamma) \to (B : \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A)) \\ \to (\mathsf{ext}\,(\mathsf{ext}\,\Gamma\,A)\,B \equiv_{\mathsf{Con}} \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B))$$

# This work: specifying QIITs

How do we represent such definitions, in general?

How do we know that we have derived the right elimination rules?

# This work: specifying QIITs

How do we represent such definitions, in general?

How do we know that we have derived the right elimination rules?

We will take an (internal) "semantics-inspired" perspective, and specify QIITs as initial objects in a category of algebras;

# This work: specifying QIITs

How do we represent such definitions, in general?

How do we know that we have derived the right elimination rules?

We will take an (internal) "semantics-inspired" perspective, and specify QIITs as initial objects in a category of algebras;

Then derive/show that initiality corresponds exactly to ordinary elimination rules. The key lemma used is that the category of algebras is complete.

# This work: specifying QIITs

How do we represent such definitions, in general?

How do we know that we have derived the right elimination rules?

We will take an (internal) "semantics-inspired" perspective, and specify QIITs as initial objects in a category of algebras;

Then derive/show that initiality corresponds exactly to ordinary elimination rules. The key lemma used is that the category of algebras is complete.

Related/alternative work: Kaposi-Kovács [2018].

# High level view

A QIIT is given by a sequence of constructors.

[cat. $\mathcal{A}$ of algebras of previous constructors] $\xRightarrow{\text{specify}}$ [new constructor c on $\mathcal{A}$] $\xRightarrow{\text{extend } \mathcal{A} \text{ with c}}$ [new cat. of algebras $\mathcal{A}'$]

# High level view

A QIIT is given by a sequence of constructors.

complete cat. $\mathcal{C}$ $\overset{\text{specify}}{\Longrightarrow}$ new constructor c on $\mathcal{C}$ $\overset{\text{extend } \mathcal{C} \text{ with c}}{\Longrightarrow}$ complete cat. $\mathcal{C}'$

# High level view

A QIIT is given by a sequence of constructors.

$$\boxed{\text{complete cat. } \mathcal{C}} \overset{\text{specify}}{\Longrightarrow} \boxed{\begin{array}{c}\text{new constructor} \\ \mathsf{c} \text{ on } \mathcal{C}\end{array}} \overset{\text{extend } \mathcal{C} \text{ with } \mathsf{c}}{\Longrightarrow} \boxed{\text{complete cat. } \mathcal{C}'}$$

At a high level, a constructor

$$\mathsf{c} : (x : \mathsf{F}(\mathsf{X})) \to \mathsf{G}(\mathsf{X}, \mathsf{x})$$

is given by two Set-valued functors $F$, $G$.

# High level view

A QIIT is given by a sequence of constructors.

$$\boxed{\text{complete cat. } \mathcal{C}} \overset{\text{specify}}{\Longrightarrow} \boxed{\begin{array}{c}\text{new constructor} \\ \mathsf{c} \text{ on } \mathcal{C}\end{array}} \overset{\text{extend } \mathcal{C} \text{ with } \mathsf{c}}{\Longrightarrow} \boxed{\text{complete cat. } \mathcal{C}'}$$

At a high level, a constructor

$$\mathsf{c} : (x : \mathsf{F}(\mathsf{X})) \to \mathsf{G}(\mathsf{X}, \mathsf{x})$$

is given by two Set-valued functors $F$, $G$.

Of course, we need restrictions on these functors.

# Well-behaved functors

$$c : (x : F(X)) \to G(X, x)$$

**Argument functor** $F : \mathcal{C} \Rightarrow \mathrm{Set}$ needs to be constrained (strictly positive etc) to prove existence, but can otherwise be arbitrary.

**Target functor** $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ definitely cannot be arbitrary.

# Well-behaved functors

$$\mathsf{c} : (x : \mathsf{F(X)}) \to \mathsf{G(X, x)}$$

**Argument functor** $F : \mathcal{C} \Rightarrow \text{Set}$ needs to be constrained (strictly positive etc) to prove existence, but can otherwise be arbitrary.

**Target functor** $G : \int^{\mathcal{C}} F \Rightarrow \text{Set}$ definitely cannot be arbitrary.

category of elements of $F$:
**objects** $(X, x)$, where $X$ in $\mathcal{C}$ and $x : \mathsf{F(X)}$,
**morphisms** $(X, x) \to (X', x')$ consists of
$f : X \to X'$ with $F(f)x \equiv x'$.

# Well-behaved functors

$$\mathsf{c} : (x : \mathsf{F(X)}) \to \mathsf{G(X, x)}$$

**Argument functor** $F : \mathcal{C} \Rightarrow \mathsf{Set}$ needs to be constrained (strictly positive etc) to prove existence, but can otherwise be arbitrary.

**Target functor** $G : \int^{\mathcal{C}} F \Rightarrow \mathsf{Set}$ definitely cannot be arbitrary.

Intuitively, a constructor should only "construct" elements of one of the sorts.

# Well-behaved functors

$$\mathsf{c} : (x : \mathsf{F}(\mathsf{X})) \to \mathsf{G}(\mathsf{X}, \mathsf{x})$$

**Argument functor** $F : \mathcal{C} \Rightarrow \text{Set}$ needs to be constrained (strictly positive etc) to prove existence, but can otherwise be arbitrary.

**Target functor** $G : \int^{\mathcal{C}} F \Rightarrow \text{Set}$ definitely cannot be arbitrary.

Intuitively, a constructor should only "construct" elements of one of the sorts.

Mathematically, $G$ needs to be continuous, i.e. preserve limits.

# Well-behaved functors

$$\mathsf{c} : (x : \mathsf{F}(\mathsf{X})) \to \mathsf{G}(\mathsf{X}, \mathsf{x})$$

**Argument functor** $F : \mathcal{C} \Rightarrow \mathrm{Set}$ needs to be constrained (strictly positive etc) to prove existence, but can otherwise be arbitrary.

**Target functor** $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ definitely cannot be arbitrary.

Intuitively, a constructor should only "construct" elements of one of the sorts.

Mathematically, $G$ needs to be continuous, i.e. preserve limits.

Complication: $\int^{\mathcal{C}} F$ is often not complete, even if $\mathcal{C}$ is, so we need a less vacuous notion of continuity.

# Relative continuity

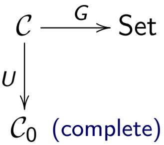
**Definition** Let $\mathcal{C}$ be a category, $\mathcal{C}_0$ a complete category, and $U : \mathcal{C} \Rightarrow \mathcal{C}_0$.

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{G} & \mathrm{Set} \\ {\scriptstyle U}\big\downarrow & & \\ \mathcal{C}_0 & \text{(complete)} & \end{array}$$

- A cone in $\mathcal{C}$ is a *$U$-limit cone* if it is mapped to a limit cone by $U$.
- A functor $G : \mathcal{C} \Rightarrow \mathrm{Set}$ is *$U$-relatively continuous* if it maps $U$-limit cones to limit cones in Set.

# Relative continuity

**Definition** Let $\mathcal{C}$ be a category, $\mathcal{C}_0$ a complete category, and $U : \mathcal{C} \Rightarrow \mathcal{C}_0$.




- A cone in $\mathcal{C}$ is a *U-limit cone* if it is mapped to a limit cone by $U$.
- A functor $G : \mathcal{C} \Rightarrow \mathsf{Set}$ is *U-relatively continuous* if it maps $U$-limit cones to limit cones in Set.

Id-limit cones are limit cones, and Id-relative continuity is continuity.

# Relative continuity

**Definition** Let $\mathcal{C}$ be a category, $\mathcal{C}_0$ a complete category, and $U : \mathcal{C} \Rightarrow \mathcal{C}_0$.

$$\begin{array}{ccc} \mathcal{C} & \xrightarrow{G} & \mathrm{Set} \\ {\scriptstyle U}\big\downarrow & & \\ \mathcal{C}_0 \ \text{(complete)} & & \end{array}$$

- A cone in $\mathcal{C}$ is a *$U$-limit cone* if it is mapped to a limit cone by $U$.
- A functor $G : \mathcal{C} \Rightarrow \mathrm{Set}$ is *$U$-relatively continuous* if it maps $U$-limit cones to limit cones in Set.

Id-limit cones are limit cones, and Id-relative continuity is continuity.

**Example** Let $U : \int^{\mathcal{C}} F \Rightarrow \mathcal{C}$ be the forgetful functor $U(X, x) = X$. If a functor $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ is $U$-relatively continuous, then e.g.

$$G(X \times Y, z) = G(X, z_0) \times G(Y, z_1)$$

where $z_i = F(\pi_i)z$.

# Constructor specifications

**Definition** A constructor specification on a complete category $\mathcal{C}$ is given by

- A functor $F : \mathcal{C} \Rightarrow \mathrm{Set}$ (the *argument functor*).
- A $U$-relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ for the forgetful functor $U : \int^{\mathcal{C}} F \Rightarrow \mathcal{C}$ (the *target functor*).

# Constructor specifications

**Definition** A constructor specification on a complete category $\mathcal{C}$ is given by

- A functor $F : \mathcal{C} \Rightarrow \mathrm{Set}$ (the *argument functor*).
- A $U$-relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ for the forgetful functor $U : \int^{\mathcal{C}} F \Rightarrow \mathcal{C}$ (the *target functor*).

The corresponding *category of algebras* $\mathcal{C}.(F, G)$ has

**objects** pairs $(X : \mathcal{C}, f : (x : \mathsf{F(X)}) \to \mathsf{G(X, x)})$

**morphisms** $(X, f) \to (Y, g)$ consisting of $\alpha : X \to Y$ making the obvious "dependent diagram" commute.

# Constructor specifications

**Definition** A constructor specification on a complete category $\mathcal{C}$ is given by

- A functor $F : \mathcal{C} \Rightarrow \mathrm{Set}$ (the *argument functor*).
- A $U$-relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ for the forgetful functor $U : \int^{\mathcal{C}} F \Rightarrow \mathcal{C}$ (the *target functor*).

The corresponding *category of algebras* $\mathcal{C}.(F, G)$ has

**objects** pairs $(X : \mathcal{C}, f : (x : \mathrm{F(X)}) \to \mathrm{G(X, x)})$

**morphisms** $(X, f) \to (Y, g)$ consisting of $\alpha : X \to Y$ making the obvious "dependent diagram" commute.

**Example**

$\sigma_{eq} : (\Gamma : \mathsf{Con})(A : \mathsf{Ty}\,\Gamma)(B : \mathsf{Ty}\,(\mathsf{ext}\,\Gamma\,A)) \to (\mathsf{ext}\,(\mathsf{ext}\,\Gamma\,A)\,B \equiv_{\mathsf{Con}} \mathsf{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B))$

## Constructor specifications

**Definition** A constructor specification on a complete category $\mathcal{C}$ is given by

- A functor $F : \mathcal{C} \Rightarrow \mathrm{Set}$ (the *argument functor*).
- A $U$-relatively continuous functor $G : \int^{\mathcal{C}} F \Rightarrow \mathrm{Set}$ for the forgetful functor $U : \int^{\mathcal{C}} F \Rightarrow \mathcal{C}$ (the *target functor*).

The corresponding *category of algebras* $\mathcal{C}.(F, G)$ has

**objects** pairs $(X : \mathcal{C}, f : (x : \mathrm{F(X)}) \to \mathrm{G(X, x)})$

**morphisms** $(X, f) \to (Y, g)$ consisting of $\alpha : X \to Y$ making the obvious "dependent diagram" commute.

**Example**

$\sigma_{eq} : (\Gamma : \mathrm{Con})(A : \mathrm{Ty}\,\Gamma)(B : \mathrm{Ty}\,(\mathrm{ext}\,\Gamma\,A)) \to (\mathrm{ext}\,(\mathrm{ext}\,\Gamma\,A)\,B \equiv_{\mathrm{Con}} \mathrm{ext}\,\Gamma\,(\sigma\,\Gamma\,A\,B))$

$$F_{\sigma_{eq}}(C, T, ext, \sigma) = (\Sigma\Gamma : C)(\Sigma A : T(\Gamma))(T(ext\,\Gamma\,A))$$
$$G_{\sigma_{eq}}(C, T, ext, \sigma, \Gamma, A, B) = \big((ext\,(ext\,\Gamma\,A)\,B) \equiv_C (ext\,\Gamma\,(\sigma\,\Gamma\,A\,B))\big).$$

# Categories of algebras are complete

**Theorem** Let $(F, G)$ be a constructor specification on a complete category $\mathcal{C}$. Then the category of algebras $C.(F, G)$ is also complete. □

For this, relative continuity of target functors is essential.

# Categories of algebras are complete

**Theorem** Let $(F, G)$ be a constructor specification on a complete category $\mathcal{C}$. Then the category of algebras $C.(F, G)$ is also complete. □

For this, relative continuity of target functors is essential.

Consequences:

# Categories of algebras are complete

**Theorem** Let $(F, G)$ be a constructor specification on a complete category $\mathcal{C}$. Then the category of algebras $C.(F, G)$ is also complete. □

For this, relative continuity of target functors is essential.

Consequences:

1. Preconditions satisfied for adding another constructor to the category of algebras.

# Categories of algebras are complete

**Theorem** Let $(F, G)$ be a constructor specification on a complete category $\mathcal{C}$. Then the category of algebras $C.(F, G)$ is also complete. $\square$

For this, relative continuity of target functors is essential.

Consequences:

1. Preconditions satisfied for adding another constructor to the category of algebras.
2. Allows using limits when reasoning about algebras, as is needed for the elimination rules.

# Categories of algebras are complete

**Theorem** Let $(F, G)$ be a constructor specification on a complete category $\mathcal{C}$. Then the category of algebras $C.(F, G)$ is also complete. $\square$

For this, relative continuity of target functors is essential.

Consequences:

1. Preconditions satisfied for adding another constructor to the category of algebras.
2. Allows using limits when reasoning about algebras, as is needed for the elimination rules.
3. Partial progress towards existence of initial algebras (solution set condition missing).

# Point and path constructors

This works for any relatively continuous target functor.

# Point and path constructors

This works for any relatively continuous target functor.

In particular, for QIITs, we are interested in point and path constructors:

- Point constructors have target functors that project out a base sort.
- Path constructors have target functors that can be represented by two natural transformations between target functors (giving LHS, RHS).

# Point and path constructors

This works for any relatively continuous target functor.

In particular, for QIITs, we are interested in point and path constructors:

- Point constructors have target functors that project out a base sort.
- Path constructors have target functors that can be represented by two natural transformations between target functors (giving LHS, RHS).

**Theorem** Target functors for point and path constructors are relatively continuous.

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

For dependent elimination, we have:

**Theorem**

$$\text{initiality} \;\Leftrightarrow\; \text{induction}$$

□

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

For dependent elimination, we have:

**Theorem**

$$\text{initiality} \Leftrightarrow \text{section induction} \Leftrightarrow \text{induction}$$

□

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

For dependent elimination, we have:

**Theorem**

$$\text{initiality} \Leftrightarrow \text{section induction} \Leftrightarrow \text{induction}$$

Dijkstra thesis [2017]
(syntactic)

□

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

For dependent elimination, we have:

**Theorem**

initiality $\Leftrightarrow$ section induction $\Leftrightarrow$ induction

completeness of cat. of algebras (semantic)

Dijkstra thesis [2017] (syntactic)

□

# Initiality and induction

Concise QIITs formulation: every category of algebras has an initial object.

Since the initial object is an algebra, we get the introduction rules.

Since it is initial, it is the smallest algebra, and we get the (non-dependent) elimination rules.

For dependent elimination, we have:

**Theorem**

initiality $\simeq$ section induction $\simeq$ induction

completeness of cat. of algebras (semantic)

Dijkstra thesis [2017] (syntactic)

□

# Summary

QIITs represented by sequence of constructor specifications.

Constructor specification given by argument and target functors.

Each QIIT representation gives rise to a category of algebras; we are interested in its initial object.

An algebra is initial exactly when it satisfies the usual induction principle.

Same method should work also for higher inductive types, but we want to make sure that all categorical concepts still make sense.

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus and Fredrik Nordvall Forsberg
Quotient Inductive-Inductive Types.
FoSSaCS 2018.

# Summary

QIITs represented by sequence of constructor specifications.

Constructor specification given by argument and target functors.

Each QIIT ... are
interested ...

## Thank you!

An algebra is initial exactly when it satisfies the usual induction principle.

Same method should work also for higher inductive types, but we want to make sure that all categorical concepts still make sense.

Thorsten Altenkirch, Paolo Capriotti, Gabe Dijkstra, Nicolai Kraus and Fredrik Nordvall Forsberg
Quotient Inductive-Inductive Types.
FoSSaCS 2018.