Constructing Quotient Inductive-Inductive Types[1]

Ambrus Kaposi[1], **András Kovács**[1], Thorsten Altenkirch[2]

[1]Eötvös Loránd University, Budapest

[2]University of Nottingham

POPL, Cascais, 18 January 2019

[1]This work was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002) and COST Action EUTypes CA15123.

Motivation

Quotient inductive-inductive types (QIITs) are a more general notion of the inductive types familiar from proof assistants such as Coq and Agda.

Motivation

Quotient inductive-inductive types (QIITs) are a more general notion of the inductive types familiar from proof assistants such as Coq and Agda.

QIITs could be useful for formalizing lots of mathematics, including the **metatheory of various logics and type theories**.

Motivation

Quotient inductive-inductive types (QIITs) are a more general notion of the inductive types familiar from proof assistants such as Coq and Agda.

QIITs could be useful for formalizing lots of mathematics, including the **metatheory of various logics and type theories**.

We aim to build background theory for QIITs, then eventually do practical implementations. Current proof assistants do not support QIITs.

Motivation

Quotient inductive-inductive types (QIITs) are a more general notion of the inductive types familiar from proof assistants such as Coq and Agda.

QIITs could be useful for formalizing lots of mathematics, including the **metatheory of various logics and type theories**.

We aim to build background theory for QIITs, then eventually do practical implementations. Current proof assistants do not support QIITs.

We believe that the mechanized metatheory of type theories can be made much more convenient, and QIITs are one key feature (but not the only one).

Motivation

Quotient inductive-inductive types (QIITs) are a more general notion of the inductive types familiar from proof assistants such as Coq and Agda.

QIITs could be useful for formalizing lots of mathematics, including the **metatheory of various logics and type theories**.

We aim to build background theory for QIITs, then eventually do practical implementations. Current proof assistants do not support QIITs.

We believe that the mechanized metatheory of type theories can be made much more convenient, and QIITs are one key feature (but not the only one).

The study of QIITs also has **intrinsic mathematical value**.

Example: type theory as a QIIT (1)

A fragment of a well-typed syntax of a type theory, quotiented by $\beta\eta$-conversion:

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

•     : Con
_▶_   : (Γ : Con) → Ty Γ → Con

Bool : Ty Γ
Pi    : (A : Ty Γ) → Ty (Γ ▶ A) → Ty Γ
...
trueβ  : BoolElim t f true  = t
falseβ : BoolElim t f false = f
...
```

Example: type theory as a QIIT (2)

The **conventional way** is to define conversion as a family of inductive relations on preterms.

Example: type theory as a QIIT (2)

The **conventional way** is to define conversion as a family of inductive relations on preterms.

In that way, we need to explicitly specify that all type and term formers are congruences with respect to conversion. We also need lots of proofs that constructions in the ambient theory respect conversion in the syntax.

Example: type theory as a QIIT (2)

The **conventional way** is to define conversion as a family of inductive relations on preterms.

In that way, we need to explicitly specify that all type and term formers are congruences with respect to conversion. We also need lots of proofs that constructions in the ambient theory respect conversion in the syntax.

We find ourselves in setoid hell.

Example: type theory as a QIIT (3)

With the QIIT definition:

Example: type theory as a QIIT (3)

With the QIIT definition:

- Equations are automatically respected by everything in the ambient theory.

Example: type theory as a QIIT (3)

With the QIIT definition:

- Equations are automatically respected by everything in the ambient theory.
- We automatically get congruence closure.

Example: type theory as a QIIT (3)

With the QIIT definition:

- Equations are automatically respected by everything in the ambient theory.
- We automatically get congruence closure.
- Everything is well-typed, including equations. Type preservation becomes a trivial (*inexpressible!*) property.

Example: type theory as a QIIT (3)

With the QIIT definition:

- Equations are automatically respected by everything in the ambient theory.
- We automatically get congruence closure.
- Everything is well-typed, including equations. Type preservation becomes a trivial (*inexpressible!*) property.

(Is this heaven?

Example: type theory as a QIIT (3)

With the QIIT definition:

- Equations are automatically respected by everything in the ambient theory.
- We automatically get congruence closure.
- Everything is well-typed, including equations. Type preservation becomes a trivial (*inexpressible!*) property.

(Is this heaven? No: we still need to escape from transport hell and some other hells)

Key questions

- **Syntax:** what is a valid QIIT definition/signature?

Key questions

- **Syntax:** what is a valid QIIT definition/signature?
- **Semantics:** can we build categories of algebras for QIITs? What is the induction principle for a QIIT and what are its computation rules? How can we relate induction and recursion?

Key questions

- **Syntax:** what is a valid QIIT definition/signature?
- **Semantics:** can we build categories of algebras for QIITs? What is the induction principle for a QIIT and what are its computation rules? How can we relate induction and recursion?
- **Construction:** do QIITs (initial algebras) exists? How can we construct them, and from what building blocks?

Key questions

- **Syntax:** what is a valid QIIT definition/signature?
- **Semantics:** can we build categories of algebras for QIITs? What is the induction principle for a QIIT and what are its computation rules? How can we relate induction and recursion?
- **Construction:** do QIITs (initial algebras) exists? How can we construct them, and from what building blocks?

Our contribution: giving answers to all of the above.

QIIT syntax: a theory of signatures

Constructors can depend on any previous constructor.

QIIT syntax: a theory of signatures

Constructors can depend on any previous constructor.

Natural choice: use a domain-specific dependent type theory for describing signatures. See also: Cartmell's generalized algebraic theories.

QIIT syntax: a theory of signatures

Constructors can depend on any previous constructor.

Natural choice: use a domain-specific dependent type theory for describing signatures. See also: Cartmell's generalized algebraic theories.

We call it the **theory of signatures**. Every Γ typing context in this theory is a valid QIIT signature.

QIIT syntax: a theory of signatures

Constructors can depend on any previous constructor.

Natural choice: use a domain-specific dependent type theory for describing signatures. See also: Cartmell's generalized algebraic theories.

We call it the **theory of signatures**. Every Γ typing context in this theory is a valid QIIT signature.

Example:

$$(\mathit{Nat} : \mathsf{U},\ \ \mathit{zero} : \mathsf{El}\,\mathit{Nat},\ \ \mathit{suc} : \mathit{Nat} \to \mathsf{El}\,\mathit{Nat})$$

QIIT syntax: a theory of signatures

Constructors can depend on any previous constructor.

Natural choice: use a domain-specific dependent type theory for describing signatures. See also: Cartmell's generalized algebraic theories.

We call it the **theory of signatures**. Every Γ typing context in this theory is a valid QIIT signature.

Example:

$$(\mathit{Nat} : \mathsf{U},\ \ \mathit{zero} : \mathsf{El}\,\mathit{Nat},\ \ \mathit{suc} : \mathit{Nat} \to \mathsf{El}\,\mathit{Nat})$$

In this talk I focus on **closed** QIITs, which don't refer to external types.

Theory of signatures (1)

Formally, the theory of signatures is a category with families (CwF) extended with some structure. Rules for the universe and functions:

$$\frac{}{\Gamma \vdash \mathsf{U}} \qquad \frac{\Gamma \vdash a : \mathsf{U}}{\Gamma \vdash \mathsf{El}\, a}$$

$$\frac{\Gamma \vdash a : \mathsf{U} \qquad \Gamma, x : \mathsf{El}\, a \vdash B}{\Gamma \vdash (x : a) \to B} \qquad \frac{\Gamma \vdash t : (x : a) \to B \qquad \Gamma \vdash u : \mathsf{El}\, a}{\Gamma \vdash t\, u : B[x \mapsto u]}$$

Strict positivity is enforced by function domain types.

Signature for a fragment of a type theory:

$$(\mathit{Con} : \mathsf{U},\ \mathit{Ty} : \mathit{Con} \to \mathsf{U},\ \bullet : \mathsf{El}\, \mathit{Con},\ - \blacktriangleright - : (\Gamma : \mathit{Con}) \to \mathit{Ty}\, \Gamma \to \mathsf{El}\, \mathit{Con})$$

Theory of signatures (2)

Allowing non-iterated equality constructors in signatures:

$$\frac{\Gamma \vdash a : \mathsf{U} \qquad \Gamma \vdash t : \mathsf{El}\,a \qquad \Gamma \vdash u : \mathsf{El}\,a}{\Gamma \vdash t = u} \qquad \frac{\Gamma \vdash p : t = u}{\Gamma \vdash t \equiv u}$$

Signature for intervals:

$$(\mathit{Int} : \mathsf{U},\ \mathit{left} : \mathsf{El}\,\mathit{Int},\ \mathit{right} : \mathsf{El}\,\mathit{Int},\ \mathit{segment} : \mathit{left} = \mathit{right})$$

Semantics

For each signature, we need a category where objects are algebras and morphisms are homomorphisms.

Semantics

For each signature, we need a category where objects are algebras and morphisms are homomorphisms.

Example: for natural numbers, an algebra is a set together with an element (zero) and an endofunction (successor), and a morphism is a function which preserves zeros and successors.

Semantics

For each signature, we need a category where objects are algebras and morphisms are homomorphisms.

Example: for natural numbers, an algebra is a set together with an element (zero) and an endofunction (successor), and a morphism is a function which preserves zeros and successors.

Initiality in the category of algebras describes **recursion**.

Semantics

For each signature, we need a category where objects are algebras and morphisms are homomorphisms.

Example: for natural numbers, an algebra is a set together with an element (zero) and an endofunction (successor), and a morphism is a function which preserves zeros and successors.

Initiality in the category of algebras describes **recursion**.

However, we also want to talk about **induction**, which can be viewed as a dependent version of recursion.

Semantics

For each signature, we need a category where objects are algebras and morphisms are homomorphisms.

Example: for natural numbers, an algebra is a set together with an element (zero) and an endofunction (successor), and a morphism is a function which preserves zeros and successors.

Initiality in the category of algebras describes **recursion**.

However, we also want to talk about **induction**, which can be viewed as a dependent version of recursion.

So, we instead use categories with families (CwFs), where families provide the language to talk about induction.

The CwF model (1)

See details in the paper, appendix and the formalization.

signatures	**semantics**
contexts	CwFs of algebras
types	displayed (fibered) CwFs
substitutions	strict CwF-morphisms
terms	sections of displayed CwFs
empty context	terminal CwF
context extension	total CwF of a displayed CwF
universe	CwF of sets
El	discrete CwF formation
...	...

The CwF model (2)

This is a **large** construction, partially formalized in Agda.

The CwF model (2)

This is a **large** construction, partially formalized in Agda.

But once we're done, nice results follow:

The CwF model (2)

This is a **large** construction, partially formalized in Agda.

But once we're done, nice results follow:

- We can compute notions of algebras, homomorphisms, recursion, induction, and do this in an exact way (not just up to isomorphisms).

The CwF model (2)

This is a **large** construction, partially formalized in Agda.

But once we're done, nice results follow:

- We can compute notions of algebras, homomorphisms, recursion, induction, and do this in an exact way (not just up to isomorphisms).
- We can prove the equivalence of induction and unique recursion by easy internal reasoning in CwFs of algebras.

The CwF model (2)

This is a **large** construction, partially formalized in Agda.

But once we're done, nice results follow:

- We can compute notions of algebras, homomorphisms, recursion, induction, and do this in an exact way (not just up to isomorphisms).
- We can prove the equivalence of induction and unique recursion by easy internal reasoning in CwFs of algebras.
- We also get to know that CwFs are modelled by any QIIT-specifiable algebraic structure. This can be useful when building models of type theories.

Constructing QIITs (1)

A question remains: is there an initial algebra for each signature?

Constructing QIITs (1)

A question remains: is there an initial algebra for each signature?

We use a **term model** construction to show this.

Constructing QIITs (1)

A question remains: is there an initial algebra for each signature?

We use a **term model** construction to show this.

Example: consider the constructible t terms in the theory of signatures, such that:

$$Nat : \mathsf{U},\ zero : \mathsf{El}\, Nat,\ suc : Nat \to \mathsf{El}\, Nat \vdash t : \mathsf{El}\, Nat$$

Since we can only use *zero* and *suc* in t, exactly the natural numerals are constructible.

Constructing QIITs (1)

A question remains: is there an initial algebra for each signature?

We use a **term model** construction to show this.

Example: consider the constructible t terms in the theory of signatures, such that:

$$\mathit{Nat} : \mathrm{U},\ \mathit{zero} : \mathrm{El}\,\mathit{Nat},\ \mathit{suc} : \mathit{Nat} \to \mathrm{El}\,\mathit{Nat} \vdash t : \mathrm{El}\,\mathit{Nat}$$

Since we can only use *zero* and *suc* in t, exactly the natural numerals are constructible.

We can also have **equations** in the context, and since we have equality reflection, constructible terms are quotiented by these equations.

Constructing QIITs (1)

A question remains: is there an initial algebra for each signature?

We use a **term model** construction to show this.

Example: consider the constructible t terms in the theory of signatures, such that:

$$Nat : \mathsf{U},\ zero : \mathsf{El}\ Nat,\ suc : Nat \to \mathsf{El}\ Nat \vdash t : \mathsf{El}\ Nat$$

Since we can only use *zero* and *suc* in t, exactly the natural numerals are constructible.

We can also have **equations** in the context, and since we have equality reflection, constructible terms are quotiented by these equations.

These observations allow us to construct term models and prove their initiality.

Constructing QIITs (2)

Some remarks:

Constructing QIITs (2)

Some remarks:

- Term models for closed QIITs can be built from the theory of signatures, which is itself a closed QIITs.

Constructing QIITs (2)

Some remarks:

- Term models for closed QIITs can be built from the theory of signatures, which is itself a closed QIITs.
- Hence, it is a *universal* closed QIIT.

Constructing QIITs (2)

Some remarks:

- Term models for closed QIITs can be built from the theory of signatures, which is itself a closed QIITs.
- Hence, it is a *universal* closed QIIT.
- Moreover, the universal closed QIIT is a fragment of usual extensional type theory.

Constructing QIITs (2)

Some remarks:

- Term models for closed QIITs can be built from the theory of signatures, which is itself a closed QIITs.
- Hence, it is a *universal* closed QIIT.
- Moreover, the universal closed QIIT is a fragment of usual extensional type theory.
- (Open QIITs are reducible to a non-conventional variation of extensional type theory)

Future work

- Generalization to large and infinitary QIITs.

Future work

- Generalization to large and infinitary QIITs.
- (Generalization to higher induction).

Future work

- Generalization to large and infinitary QIITs.
- (Generalization to higher induction).
- Integrating QIITs into a type theory with computing transports (cubical, observational).

Thank you!