Pragmatic Quotient Types in Coq

Cyril Cohen

University of Gothenburg
cyril.cohen@gu.se

July 24, 2013
Examples of quotient

- \( \mathbb{Z} = \mathbb{N} \times \mathbb{N} / \text{equivZ} \)
  \[ \text{equivZ } x \ y := (x.1 + y.2 = y.1 + x.2) \]
- \( \mathbb{Q} = \mathbb{Z} \times \mathbb{Z}^* / \text{equivQ} \)
  \[ \text{equivQ } x \ y := (x.1 \times y.2 = y.1 \times x.2) \]
- \( F = D \times D^* / \text{equivF} \)
  \[ \text{equivF } m \ n := (x.1 \times y.2 = y.1 \times x.2) \]
- Setoid/ \( \equiv \)

Original motivation for this work:

algebraic numbers for Feit Thompson.
Why forming a quotient type?

Context: Intensional Type Theory (here Coq)

Main goal: get Leibniz equality in Type Theory.

Because it is substitutive in any context:

\[
\text{eq_rect} : \forall (A : \text{Type}) (x : A) (P : A \to \text{Type}),
\]

\[
P x \to \forall y : A, x = y \to P y.
\]
What is this work about?

It is **not** about

- category theory,
- adding quotient types to the meta theory of Coq,
- giving a general axiomatization of quotients.

It is about a framework **usable in practice** for our applications (mainly discrete algebra), **without** modifying Coq.
An abstraction layer

Operations and properties on the Quotient.

\[ \text{addZC : } \forall x \, y \, z : \mathbb{Z}, \]
\[ x +_\mathbb{Z} (y +_\mathbb{Z} z) = (x +_\mathbb{Z} y) +_\mathbb{Z} z \]

\[ \text{addNNC : } \forall m \, n \, p : \mathbb{N} \times \mathbb{N}, \]
\[ \text{equivZ} (m +_{\mathbb{N} \times \mathbb{N}} (n +_{\mathbb{N} \times \mathbb{N}} p)) ((m +_{\mathbb{N} \times \mathbb{N}} n) +_{\mathbb{N} \times \mathbb{N}} p) \]
\[ \text{(where } \text{equivZ} x \, y := (x.1 + y.2 = y.1 + x.2)) \]

Low level operations and properties.
An abstraction layer

Operations and properties on the Quotient.

\[ \text{addZC} : \forall x \ y \ z : \mathbb{Z}, \]
\[ x +_{\mathbb{Z}} (y +_{\mathbb{Z}} z) = (x +_{\mathbb{Z}} y) +_{\mathbb{Z}} z \]

\[ \text{addNNC} : \forall m \ n \ p : \mathbb{N} \ast \mathbb{N}, \]
\[ m +_{\mathbb{N} \ast \mathbb{N}} (n +_{\mathbb{N} \ast \mathbb{N}} p) = (m +_{\mathbb{N} \ast \mathbb{N}} n) +_{\mathbb{N} \ast \mathbb{N}} p \mod Z \]

Low level operations and properties.
Equality modulo

Notation "x = y % [mod Q]\)" := \pi_Q x = \pi_Q y. where \pi_Q is the canonical surjection in the quotient type Q.
Quotient primitives

Base Type $T$

Canonical surjection to $x$

Representative of $x$

Canonical surjection to $y$

Representative of $y$

Figure: Quotients without equivalence relation
Record quot_class_of (T Q : Type) := QuotClass{
  repr : Q -> T;
  pi : T -> Q;
  reprK : forall x : Q, pi (repr x) = x
}.

Record quotType (T : Type) := QuotType { 
  quot_sort :> Type;
  quot_class : quot_class_of T quot_sort 
}

An instance of the quotient interface is called a quotient structure. (Altenkirch et al.: definable quotient)
Outline:

1. Quotient theory
   (embedding, lifting and automated translation)
2. Quotient construction
   (on choice types, on discrete algebraic structures)
3. Applications
Outline:

1. Quotient theory
   (embedding, lifting and automated translation)
2. Quotient construction
   (on choice types, on discrete algebraic structures)
3. Applications
Morphism lifting

Possible definition:

\[ x +_Z y := \pi (\text{repr } x +_{N \times N} \text{repr } y) \]
\[ x \leq_Z y := (\text{repr } x \leq_{N \times N} \text{repr } y) \]

Possible specification:

\[ \forall m \ n, \ \pi (m +_{N \times N} n) = \pi m +_Z \pi n \]
\[ \forall m \ n, \ (\pi m \leq_Z \pi n) = (m \leq_{N \times N} n) \]

(morphism properties)
Morphism lifting

Possible definition:

\[ x +_Z y := \pi (\text{repr } x +_{\mathbb{N}\times\mathbb{N}} \text{repr } y) \]
\[ x \leq_Z y := (\text{repr } x \leq_{\mathbb{N}\times\mathbb{N}} \text{repr } y) \]

Possible specification:

\[
\forall m, n, \pi (m +_{\mathbb{N}\times\mathbb{N}} n) = \pi m +_Z \pi n \\
\forall m, n, (\pi m \leq_Z \pi n) = (m \leq_{\mathbb{N}\times\mathbb{N}} n)
\]

(morphism properties)

\[
\forall m, n, (m = n \mod Z) = (\text{equivZ } m n)
\]
An abstraction layer

Operations and properties on the Quotient.

\[ \text{addZC} : \forall x \ y \ z : \mathbb{Z}, \]
\[ x +_{\mathbb{Z}} (y +_{\mathbb{Z}} z) = (x +_{\mathbb{Z}} y) +_{\mathbb{Z}} z \]

\[ \text{addNNC} : \forall m \ n \ p : \mathbb{N} \times \mathbb{N}, \]
\[ m +_{\mathbb{N} \times \mathbb{N}} (n +_{\mathbb{N} \times \mathbb{N}} p) = (m +_{\mathbb{N} \times \mathbb{N}} n) +_{\mathbb{N} \times \mathbb{N}} p \mod \mathbb{Z} \]

Low level operations and properties.
Translation from $Q$ to $T$

1. Elimination principle:

$$\text{quot}W : (\forall a : T, P (\pi a)) \rightarrow \forall x : Q, P x.$$ 

2. Pushing $\pi$ outwards,

$$\pi m + (\pi n + \pi p) = (\pi m + \pi n) + \pi p \downarrow$$

$$\pi (m + (n + p)) = \pi ((m + n) + p).$$
Translation from Q to T

1. Elimination principle:

\[
\text{quot} : (\forall a : T, P (\pi a)) \rightarrow \forall x : Q, P x.
\]

2. Pushing \(\pi\) outwards,

\[
\pi m + (\pi n + \pi p) = (\pi m + \pi n) + \pi p
\]

\[
\downarrow
\]

\[
m + (n + p) = (m + n) + p \pmod Z.
\]
Record **equal_to** \( Q \) \( u \) := EqualTo

\[
\{\text{equal}_\text{val} : Q; \_ : u = \text{equal}_\text{val}\}.
\]

\((\text{equal}_\text{to} \ u)\) is the type of all elements of \( Q \) that are equal to \((u : Q)\).
Lemma **eq_to_addZ** (m n : N * N)
(x : equal_to (\pi m))
(y : equal_to (\pi n)) :
equal_to (\pi (m +_{N*N} n)) :=
  EqualTo (x +_{Z} y)
  (_ : \pi (m +_{N*N} n) = x +_{Z} y).
Term inference

Example of term translation using

**Lemma** `equalE (Q : Type) (u : Q)
  (m : equal_to u) : equal_val m = u.

on \(\pi m + (\pi n + \pi p)\)
Term inference

Example of term translation using

Lemma equalE (Q : Type) (u : Q) (m : equal_to u) : equal_val m = u.

\pi m + (\pi n + \pi p).

Unification problem

\pi m + (\pi n + \pi p) \equiv equal_val ?(equal_to ?)
\( \pi m + (\pi n + \pi p) \equiv \text{equal\_val} ?(\text{equal\_to} ?) \)
Unification flow

\[ \pi m + (\pi n + \pi p) \equiv \text{equal_val \ ?(equal_to \ ?)} \]

Canonical solution:

\[ \text{equal_to_addZ \ ?m \ ?n (\?x : equal_to (\pi \ ?m)) (\?y : equal_to (\pi \ ?n)) : equal_to (\pi (?m + \?n))} \]
Unification flow

\[ \pi m + (\pi n + \pi p) \equiv \]
\[ \text{equal_val} \ \text{eq_to_addZ} \ (\text{equal_to} \ (\pi \ (?m + \ ?n))) \]

Canonical solution:

\[ \text{equal_to_addZ} \ ?m \ ?n \ (?x : \text{equal_to} \ (\pi \ ?m)) \]
\[ \ (\?y : \text{equal_to} \ (\pi \ ?n)) : \text{equal_to} \ (\pi \ (?m + \ ?n)) \]
\[ \pi m + (\pi n + \pi p) \equiv \]
\[
\text{equal_val eq_to_addZ} (\text{equal_to} (\pi (?m + ?n)))
\]

\[ \pi m \equiv \text{equal_val} ?x(\text{equal_to} ?) \]
\[ \pi n + \pi p \equiv \text{equal_val} ?y(\text{equal_to} ?) \]
\[ \pi m + (\pi n + \pi p) \equiv \]
\[ \text{equal_val eq_to_addZ } (\text{equal_to } (\pi (m + ?n))) \]

\[ \pi m \equiv \text{equal_val eq_to_pi } (\text{equal_to } (\pi m)) \]
\[ \pi n + \pi p \equiv \text{equal_val } ?y (\text{equal_to } ?) \]
Unification flow

\[ \pi m + (\pi n + \pi p) \equiv \]
\[ \text{equal_val eq_to_addZ} (\text{equal_to (}\pi (m + (? + ?)))) \]

\[ \pi m \equiv \text{equal_val eq_to_pi} (\text{equal_to (}\pi m)) \]
\[ \pi n + \pi p \equiv \]
\[ \text{equal_val eq_to_addZ} (\text{equal_to (}\pi (? + ?))) \]
\[\pi m + (\pi n + \pi p) \equiv\]
\[\text{equal_val } \text{eq_to_addZ} \ (\text{equal_to } (\pi (m + (? + ?))))\]

\[\pi m \equiv \text{equal_val } \text{eq_to_pi}(\text{equal_to } (\pi m))\]
\[\pi n + \pi p \equiv\]
\[\text{equal_val } \text{eq_to_addZ} \ (\text{equal_to } (\pi (? + ?)))\]

\[\pi n \equiv \text{equal_val } ?(\text{equal_to } ?)\]
\[\pi p \equiv \text{equal_val } ?(\text{equal_to } ?)\]
Unification flow

\[ \pi m + (\pi n + \pi p) \equiv \]

\[ \text{equal_val eq_to_addZ} (\text{equal_to (}\pi (m + (n + p))) \text{)} \]

\[ \pi m \equiv \text{equal_val eq_to_pi} (\text{equal_to (}\pi m)) \]
\[ \pi n + \pi p \equiv \]

\[ \text{equal_val eq_to_addZ} (\text{equal_to (}\pi (n + p))) \]

\[ \pi n \equiv \text{equal_val eq_to_pi} (\text{equal_to (}\pi n)) \]
\[ \pi p \equiv \text{equal_val eq_to_pi} (\text{equal_to (}\pi p)) \]
Term inference

Example of term translation using

```
Lemma equalE (Q : Type) (u : Q)
  (m : equal_to u) : equal_val m = u.
```

```
on \pi m + (\pi n + \pi p).
```

Unification problem

```
\pi m + (\pi n + \pi p) ≡ equal_val ?(equal_to ?)
```

Solution

```
\pi m + (\pi n + \pi p) ≡
  equal_val eq_to_addZ (equal_to (\pi (m + (n + p))))
```
When not use quotients?

Quotient are not a replacement for setoids.

Example: associate polynomials.

Definition \( \text{eqp } p \ q := (p \ %|\ q) \land (q \ %|\ p) \).
Notation "\( p \ %=\ q " := (\text{eqp } p \ q) .

- Compatible with division \((\_ \ %|\ \_ )\) and \((\_ \ %=\ \_ )\),
- compatible with multiplication,
When not use quotients?

Quotient are not a replacement for setoids.

Example: associate polynomials.

**Definition** \( \text{eqp } p \ q := (p \ %| \ q) \ \&\& (q \ %| \ p). \)

**Notation** "\( p \ %= \ q \)" := (eqp \( p \ q \)).

- Compatible with division (_ %| _) and (_ %= _),
- compatible with multiplication,
- **not compatible** with addition.
Outline:

1. Quotient theory
   (embedding, lifting and automated translation)
2. Quotient construction
   (on choice types, on discrete algebraic structures)
3. Applications
In an ideal system

An impredicative definition.

\begin{definition}
\text{is\_class} (C : T \to \text{Prop}) : \text{Prop} := \exists x, \forall y, C \ y \leftrightarrow R \ x \ y.
\end{definition}

\begin{definition}
Q : \text{Type} := \{C : \text{pred} \ T \mid \text{is\_class} \ C\}
\end{definition}

One element by equivalence class?

- First projection: functional extensionality.
- Second projection: proof irrelevance (for Prop).
Back to Coq ≤ 8.4, without axioms

Particular case:

- a decidable equivalence `equiv : T → T → bool`
- and
- a countable type,

This suffices to select a unique element in each equivalence class.
Particular case:

- a decidable equivalence \(\text{equiv} : T \rightarrow T \rightarrow \text{bool}\)
- and \(\text{either}\)
  - a countable type,
  - a choice type

This suffices to select a unique element in each equivalence class.
Particular case:
- a decidable equivalence $\text{equiv} : T \to T \to \text{bool}$
  and either
  - a countable type,
  - a choice type
  - a type encodable to a choice type.

This suffices to select a unique element in each equivalence class.
Quotient theory vs. Quotient construction

Outline:

1. Quotient theory
   (embedding, lifting and automated translation)

2. Quotient construction
   (on choice types, on discrete algebraic structures)

3. Applications
Applications

- Fraction field: \((D \ast D^*)/(\lambda x, y. x_1y_2 \equiv y_1x_2)\),
- Real algebraic numbers:
  \[
  \left( \sum_{x: \mathbb{R}} \sum_{P: \mathbb{Q}[X]} (P(x) \equiv_{\mathbb{R}} 0) \right) / \equiv_{\mathbb{R}}.
  \]
- Multivariate polynomials
- Elliptic curves (Bartzia and Strub)
- Field extensions (O’Connor)
- Countable algebraic closure (Gonthier)
Related work

In intensional type theory:
- Hoffman (PhD Thesis),
- Chicli et al. (mathematical quotients in TT),
- Courtieu (normalized types),
- Voevodsky’s UF and HoTT (cf HoTT book),
- Altenkirch et al. (definable quotients).

Elsewhere (Isabelle/HOL, . . . )
A framework to both use and construct quotients.
Deals with quotients in discrete algebra.
Successfully used in the Mathematical Components project.

Future work:
Generalize to encompass quotients without \texttt{repr}.
Elimination of quotient and \textbackslash\texttt{pi} in one go?
Thank you for your attention.
HIT Quotients

From the HoTT book (homotopytypetheory.org/book/): the higher inductive type $A/R$ generated by

- A function $q : A \to A/R$;
- For each $a, b : A$ such that $R(a, b)$, an equality $q(a) = q(b)$; and
- The 0-truncation constructor: for all $x, y : A/R$ and $r, s : x = y$, we have $r = s$. 
Choice types

If a type $T$ has a choice structure, there exists an operator

$$\textbf{xchoose} : \forall P : T \rightarrow \text{bool},$$

$$(\exists y : T, P y) \rightarrow T.$$

which given a proof of $\exists y, P y$ returns an element $z$, such that $z$ is the same if $\textbf{xchoose}$ is given a proof of $\exists y, Q y$ when $P$ and $Q$ are logically equivalent.
Lemma \texttt{equiv\_exists} (x : T) :
exists y, (equiv x) y.
(equiv x) is the equivalence class of x.

Definition \texttt{canon} (x : T) :=
xchoose (equiv\_exists x).
Choice of a unique element in (equiv x).
Record `equiv_quot` := EquivQuot { erepr : T; erepr_canon : canon erepr == erepr }.
Choice types (sordid details)

Georges Gonthier’s definition:

```
Record Choice.mixin_of T := Choice.Mixin { 
   find : pred T -> nat -> option T; 
   _ : forall P n x, 
      find P n = Some x -> P x; 
   _ : forall P : pred T, 
      (exists x, P x) -> exists n, find P n; 
   _ : forall P Q : pred T, 
      P =1 Q -> find P =1 find Q 
}. 
```