# Definable quotients in intensional type theory

Nuo Li

School of Computer Science
University of Nottingham, UK

FP Away day, July 2014

# What is a quotient?

- In arithmatics, 4 is the quotient of $8 \div 2$ (or $8/2$)
- In set theory, given a set $A$ and an equivalence relation $\sim$ on it, a quotient set $A/\sim$ is the collection of equivalence classes.
- In type theory, a quotient type is a type $A$ whose equality is redefined by an equivalence relation $\sim$ on it.

# Quotients in type theory

- *Quotient type* is an extensional notion which is not available in intensional type theory.
- *Setoids* are sometimes used to represent quotients. A setoid consists of
  1. a set $A$,
  2. a relation $\sim: A \to A \to \mathbf{Prop}$, and
  3. it is an equivalence, i.e. it is reflexive, symmetric and transitive.
- Some quotients are definable inductively without using quotient type former, e.g. the set of rational numbers $\mathbb{Q}$
- Algebraic structures for these quotients in intensional type theory:
  - Prequotient
  - Quotient (dependent eliminator)
  - Quotent (Hofmann's version)
  - Exact quotient
  - Definable quotient

# Prequotient

### Prequotient

Given a setoid $(A, \sim)$, a *prequotient* $(Q, [\_], \text{sound})$ over that setoid consists of

1. a set $Q$,
2. a function $[\_] : A \to Q$,
3. a proof *sound* that the function $[\_]$ is compatible with the relation $\sim$, that is
$$\text{sound} : (a, b : A) \to a \sim b \to [a] = [b],$$

In category theory, prequotient corresponds to a commute diagram of morphisms – *(co)fork*

$$\begin{tikzcd} R \arrow[r, shift left, "\pi_0"] \arrow[r, shift right, "\pi_1"'] & A \arrow[r, "{[\_]}"] & Q \end{tikzcd}$$

# Quotient (Hofmann's version)

### Quotient (Hofmann's)

A prequotient $(Q, [\_], \mathrm{sound})$ is a quotient (Hofmann's) if we also have

$$\mathrm{lift} : (f : A \to B) \to (\forall a, b \to a \sim b \to f\ a \equiv f\ b) \to (Q \to B)$$

such that $\mathrm{lift} - \beta : \mathrm{lift}\ \ f\ \ p[a] = f\ \ a$,
together with an induction principle. Suppose $B$ is a predicate,

$$\mathrm{qind} : ((a : A) \to B\ [a] \to ((q : Q) \to B\ q)$$

In category theory, a quotient is just a coequalizer (a special fork)

$$\begin{tikzcd}
R \arrow[r, shift left, "\pi_1"] \arrow[r, shift right, "\pi_2"'] & A \arrow[r, "{[\_]}"] \arrow[dr, "f"'] & Q \arrow[d, dotted, "\hat{f}"] \\
 & & Q'
\end{tikzcd}$$

# Quotient (dependent eliminator)

## Quotient(dependent eliminator)

A prequotient $(Q, [\_], \mathrm{sound})$ is a quotient if we also have

4. for any $B : Q \to \textbf{Set}$, an eliminator

$$\begin{aligned} \mathrm{qelim}_B \ &: \ (f : (a : A) \to B\,[a]) \\ &\to ((p : a \sim b) \to f\ a \simeq_{\mathrm{sound}\ p} f\ b) \\ &\to ((q : Q) \to B\,q) \end{aligned}$$

such that $\mathrm{qelim} - \beta : \mathrm{qelim}_B\ f\ p\,[a] \equiv f\ a$.

They are equivalent.

# Exact quotient

- A quotient is *exact* (or effective) if

  $$exact : (\forall a, b : A) \to [a] \equiv [b] \to a \sim b$$

  i.e. a coequalizer is exact if following diagram commutes (i.e. a pullback)

  $$\begin{tikzcd}
R \arrow[r, "\pi_1"] \arrow[d, "\pi_2"'] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "{[\_]}"] \\
A \arrow[r, "{[\_]}"'] & Q
\end{tikzcd}$$

- Propositional extensionality (univalence for propositions) $\implies$ every quotient is exact.

# Definable quotient

### Definable quotient

Given a setoid $(A, \sim)$, a *definable quotient* is a prequotient $(Q, [\_], \text{sound})$ with

$$\begin{aligned} \text{emb} &: Q \to A \\ \text{complete} &: (a : A) \to \text{emb}\,[a] \sim a \\ \text{stable} &: (q : Q) \to [\text{emb } q] \equiv q. \end{aligned}$$

It corresponds to a split coequalizer:

$$\begin{tikzcd} R \arrow[r, shift left, "\pi_0"] \arrow[r, shift right, "\pi_1"'] & A \arrow[r, "{[\_]}"] & Q \end{tikzcd}$$

a fork with morphisms $\text{emb} : Q \to A$ and $t : A \to R$ such that $[\_] \circ emb = 1_C$ , $\text{emb} \circ [\_] = \pi_2 \circ t$ and $\pi_1 \circ t = 1_B$. $(t\ a = (\text{emb}[a], a))$

# Relations of these structures

- Definable quotients give rise to exact quotients.

  1. Quotient (dependent version) $\iff$ Quotient (Hofmann's version)
  2. Definable quotients $\to$ quotient
  3. Definable quotient is exact

     Proof:

     $[a] = [b]$

     $\implies$ emb $[a] = \mathit{emb}\ [b]$ (by congruence)

     $\implies$ $a \sim$ emb $[a] = \mathit{emb}\ [b] \sim b$ (by completeness)

     $\implies$ $a \sim b$

# Example : Integers $\mathbb{Z}$ I

- setoid integer :

$$\mathbb{Z}_0 := \mathbb{N} \times \mathbb{N}$$

$$(n_1, n_2) \sim (n_3, n_4) := n_1 + n_4 = n_3 + n_2$$

An example of the operation defined on setoid integers: addition

```
_+_ : ℤ0 → ℤ0 → ℤ0
(x+ , x-) + (y+ , y-) = (x+ ℕ+ y+) , (x- ℕ+ y-)
```

# Example : Integers $\mathbb{Z}$ II

- set integer:

```
data ℤ : Set where
   +_    : ℕ → ℤ
   -suc_ : ℕ → ℤ
```

- Normalisaton function:

$$\begin{aligned}
&[\_] &&: \mathbb{Z}_0 \to \mathbb{Z}\\
&[\ \mathsf{m}\ ,\ 0\ ] &&= +\ \mathsf{m}\\
&[\ 0\ ,\ \mathsf{suc}\ \mathsf{n}\ ] &&= \text{-}\mathsf{suc}\ \mathsf{n}\\
&[\ \mathsf{suc}\ \mathsf{m}\ ,\ \mathsf{suc}\ \mathsf{n}\ ] &&= [\ \mathsf{m}\ ,\ \mathsf{n}\ ]
\end{aligned}$$

# Example : Integers ℤ III

- To find a canonical choice for each equivalence class, embedding function:

```
⌜_⌝        : ℤ → ℤ₀
⌜ + n ⌝    = n , 0
⌜ -suc n ⌝ = 0 , suc n
```

*sound complete stable* and *exact* can be verified.

# Example : Rational numbers $\mathbb{Q}$ I

- Setoid encoding : unreduced fractions

$$\mathbb{Q}_0 = \mathbb{Z} \times \mathbb{N}$$

Note: 1/2 represents $\frac{1}{3}$
Equivalence relation :

$$\frac{a}{b} = \frac{c}{d} := a \times d = c \times b$$

- Set encoding: reduced fractions

$$\mathbb{Q} = \Sigma((n, d) : \mathbb{Z} \times \mathbb{N})).\text{coprime } n\ (d + 1)$$

# Example : Rational numbers ℚ II

- Normalisation is essentially just fraction reducing

```
[_] : ℚ₀ → ℚ
[ (+ 0) /suc d ] = ℤ.+_ 0 ÷ 1
[ (+ (suc n)) /suc d ] with gcd (suc n) (suc d)
[ (+ suc n) /suc d ] | di , g = GCD′→ℚ (suc n) (suc d)
   di (λ ()) (C.gcd-gcd′ g)
[ (-suc n) /suc d ] with gcd (suc n) (suc d)
... | di , g = - GCD′→ℚ (suc n) (suc d) di (λ ())
   (C.gcd-gcd′ g)
```

- Embedding is trivial – forgetting proof of coprimarity

```
⌜_⌝ : ℚ → ℚ₀
⌜ x ⌝ = (ℤcon (ℚ.numerator x)) /suc (ℚ.denominator-1 x)
```

# The application of definable quotients I

- As long as we have defined operations or have proved theorems for the setoid representations, we can lift them by just mixing normalisation and embedding functions. e.g.
  In the case of Integers:

$$\begin{aligned} &\mathsf{lift}_1 : (\mathsf{op} : \mathbb{Z}_0 \to \mathbb{Z}_0) \to \mathbb{Z} \to \mathbb{Z} \\ &\mathsf{lift}_1\ \mathsf{op} = [\_] \circ \mathsf{op} \circ \ulcorner \_ \urcorner \end{aligned}$$

Advantages of definable quotients:

- Setoid has simpler encoding, better manipulation
- More auxiliary functions
- Flexible in conversions of two representations
- Benefit from both

# Case : The distributivity of integers I

- For set encoding:

```
dist^l    :  _ℤ*_ DistributesOver^l _ℤ+_
dist^l x y z with sign y S≟ sign z
dist^l x y z    | yes p
    rewrite p
        | lem1 y z p
        | lem2 y z p =
    trans (cong (λ n → sign x S* sign z ◃ n)
    (ℕdist^l ∣ x ∣ ∣ y ∣ (∣ z ∣)))
    (lem3 (∣ x ∣ ℕ* ∣ y ∣) (∣ x ∣ ℕ* ∣ z ∣) _)
dist^l x y z | no ¬p = …
```

Three lemmas for first case, second case has to be proved from scratch.

# Case : The distributivity of integers II

- For setoid encoding:
  It can be done automatically

```
dist' :   _*_ DistributesOver' _+_
dist' (a , b) (c , d) (e , f) = solve 6
    (λ a b c d e f → a :* (c :+ e) :+ b :* (d :+ f) :+
        (a :* d :+ b :* c :+ (a :* f :+ b :* e))
        :=
        a :* c :+ b :* d :+ (a :* e :+ b :* f) :+
        (a :* (d :+ f) :+ b :* (c :+ e))) refl a b c d e f
```

- Combined with "Reflection” technique, it can be done completely automatic, like the **ring** tactic in Coq.
- It can be generalised to any properties which can be converted into equations of natural numbers.

# Case : The distributivity of integers III

- In practice, we manually construct the proof for efficiency, which is still simpler.

```
dist′ : _ℤ₀*_ DistributesOver′ _ℤ₀+_
dist′ (a , b) (c , d) (e , f) =
  cong₂ _ℕ+_ (dist-lem′ a b c d e f)
  (sym (dist-lem′ a b d c f e))

dist-lem′ : ∀ a b c d e f →
  a ℕ* (c ℕ+ e) ℕ+ b ℕ* (d ℕ+ f) ≡
  (a ℕ* c ℕ+ b ℕ* d) ℕ+ (a ℕ* e ℕ+ b ℕ* f)
dist-lem′ a b c d e f = trans
  (cong₂ _ℕ+_ (ℕdist′ a c e) (ℕdist′ b d f))
  (swap23 (a ℕ* c) (a ℕ* e) (b ℕ* d) (b ℕ* f))
```

# Summary

- Quotients types are not available in intensional type theory.
- Setoids are not good enough.
- Algebraic structures to encode quotient sets which are definable.
- The advantages of using definable quotient structures.
- Any Questions?