1 Appendix

```
module Quotient where

open import Data.Product
open import Function

open import Relation.Binary.Core
open import Relation.Binary.PropositionalEquality
  hiding (isEquivalence)

open import ThomasProperties
```

Definition of setoids

```
record Setoid : Set₁ where
  infix 4 _≈_
  field
    Carrier : Set
    _≈_     : Carrier → Carrier → Set
    isEquivalence : IsEquivalence _≈_
  open IsEquivalence isEquivalence public

open Setoid renaming
  (refl to reflexive; sym to symmetric; trans to transitive)
```

Prequotients

```
record PreQu (S : Setoid) : Set₁ where
  constructor
    Q:_[ ]:_sound:_
  private
    A  =  Carrier S
    _~_  =  _≈_ S
  field
    Q    : Set
    [_]  : A → Q
    sound  : ∀ {a b : A} → a ~ b → [ a ] ≡ [ b ]

open PreQu renaming
  (Q to Q'; [_] to nf; sound to sound')
```

Quotients as prequotients with a dependent eliminator.

```
record Qu {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
```

```
    qelim:_qelim-β:_
  private
    A      =  Carrier S
    _~_    =  _≈_ S
    Q      =  Q' PQ
    [_]    =  nf PQ
    sound  :  ∀ {a b : A} → (a ~ b) → [ a ] ≡ [ b ]
    sound  =  sound' PQ
  field
    qelim  :  {B : Q → Set}
           → (f : (a : A) → B [ a ])
           → ((a b : A) → (p : a ~ b)
             → subst B (sound p) (f a) ≡ f b)
           → (q : Q) → B q
    qelim-β  :  ∀ {B a f} q → qelim {B} f q [ a ] ≡ f a
open Qu
```

Proof irrelevance of qelim

```
qelimIrr : {S : Setoid} {PQ : PreQu S} (x : Qu PQ)
  → ∀ {B a f q q'}
  → qelim x {B} f q (nf PQ a)
    ≡ qelim x {B} f q' (nf PQ a)
qelimIrr x {B} {a} {f} {q} {q'}  =  (qelim-β x {B} {a} {f} q)
  ▶ ⟨ qelim-β x {B} {a} {f} q' ⟩
```

Exact quotients

```
record QuE {S : Setoid} {PQ : PreQu S} (QU : Qu PQ) : Set₁ where
  constructor
    exact:_
  private
    A  =  Carrier S
    _~_  =  _≈_ S
    [_]  =  nf PQ
  field
    exact  :  ∀ {a b : A} → [ a ] ≡ [ b ] → a ~ b
open QuE
```

Quotients as prequotients with a non-dependent eliminator (lift).
(As in Hofmann's PhD dissertation.)

```
record QuH {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
    lift:_lift-β:_qind:_
  private
    A       =  Carrier S
    _~_     =  _≈_ S
    Q       =  Q' PQ
    [_]     =  nf PQ
  field
    lift     : {B : Set}
             → (f : A → B)
             → ((a b : A) → (a ~ b) → f a ≡ f b)
             → Q → B
    lift-β : ∀ {B a f q} → lift {B} f q [a] ≡ f a
    qind : (P : Q → Set)
             → (∀ x → (p p' : P x) → p ≡ p')
             → (∀ a → P [a])
             → (∀ x → P x)
open QuH renaming (lift to lift'; lift-β to lift-β')
```

Definable quotients

```
record QuD {S : Setoid} (PQ : PreQu S) : Set₁ where
  constructor
    emb:_complete:_stable:_
  private
    A     =  Carrier S
    _~_   =  _≈_ S
    Q     =  Q' PQ
    [_]   =  nf PQ
  field
    emb    : Q → A
    complete : ∀ a → emb [a] ~ a
    stable : ∀ q → [emb q] ≡ q
open QuD
```

Relations between types of quotients:
Below, we show the following, where the arrow → means "gives rise to" :
QuH → Qu (Proposition 3 in the paper)
Qu → QuH (Reverse of Proposition 3)
QuD → QuE (A definable quotient is always exact)

QuD → Qu
QuD → QuH (Also a consequence of QuD → Qu and Qu → QuH)

```
QuH→Qu : {S : Setoid} → {PQ : PreQu S}
  → (QuH PQ) → (Qu PQ)
QuH→Qu {S} {Q: Q [ ] : [_] sound: sound}
  (lift: lift lift-β: β qind: qind) =
  record
    {qelim = λ {B} → qelim₁ {B}
    ;qelim-β = λ {B} {a} {f} → qelim-β₁ {B} a f
    }
  where
    A     = Carrier S
    _~_  = _≈_ S
      -- the dependent function f is made independent
    indep : {B : Q → Set} → ((a : A) → B [a]) → A → Σ Q B
    indep f a = [a], f a
    indep-β : {B : Q → Set}
           → (f : (a : A) → B [a])
           → (∀ a b → (p : a ~ b) → subst B (sound p) (f a) ≡ f b)
           → ∀ a a' → (a ~ a') → indep {B} f a ≡ indep f a'
    indep-β {B} f q a a' p = (cong_,_ [a] [a'] (sound p) (f a))
                              ▶ ((λ b → [a'], b) ⋆ (q a a' p))
    lift₀ : {B : Q → Set}
      → (f : (a : A) → (B [a]))
      → ((a a' : A) → (p : a ~ a')
      → subst B (sound p) (f a) ≡ f a')
      → Q → Σ Q B
    lift₀ f q = lift (indep f) (indep-β f q)
    qind₁ : {B : Q → Set}
      → (f : (a : A) → B [a])
      → (q : ∀ a b → (p : a ~ b) → subst B (sound p) (f a) ≡ f b)
      → ∀ (c : Q) → proj₁ (lift₀ f q c) ≡ c
    qind₁ {B} f q = qind P heredity base
      where
        f' : Q → Σ Q B
        f' = lift₀ f q
        P : Q → Set
        P c = proj₁ {_} {_} {Q} {B} (lift₀ f q c) ≡ c
        heredity : ∀ x → (p p' : P x) → p ≡ p'
```

```
      heredity x p p' = ≡-prfIrr ((lift₀ f q x) ₁) x p p'
      base : ∀ a → P [ a ]
      base a = proj₁ ⋆ β
  qelim₁ : {B : Q → Set}
         → (f : (a : A) → (B [ a ]))
         → (∀ a b → (p : a ∼ b) → subst B (sound p) (f a) ≡ f b)
         → (c : Q) → B c
  qelim₁ {B} f q c = subst B (qind₁ f q c)
    (proj₂ {_} {_} {Q} {B} (lift₀ f q c))
  qelim-β₁ : ∀ {B} a f q → qelim₁ {B} f q [ a ] ≡ f a
  qelim-β₁ {B} a f q =
    (substIrr B (qind₁ f q [ a ])
      (cong-proj₁ {Q} {B} (lift₀ f q [ a ]) (indep f a) β)
      (proj₂ {_} {_} {Q} {B} (lift₀ f q [ a ]))) ▶
    (cong-proj₂ {Q} {B} (lift₀ f q [ a ]) (indep f a) β)


Qu→QuH : {S : Setoid} → {PQ : PreQu S}
  → (Qu PQ) → (QuH PQ)
Qu→QuH {S} {Q: Q [ ] : [_] sound: sound} (qelim: qelim qelim-β: β) =
  record
  {lift = λ {B} f s → qelim {λ _ → B} f (λ a b p
       → (subFix (sound p) B (f a)) ▶ (s a b p))
  ; lift-β = λ {B} {a'} {f} {s} → β {λ _ → B} {a'} {f} (λ a b p
       → (subFix (sound p) B (f a)) ▶ (s a b p))
  ; qind = λ P irr f
    → qelim {P} f (λ a b p → irr [ b ] (subst P (sound p) (f a)) (f b))
  }
  where
    subFix : ∀ {A : Set} {c d : A} (x : c ≡ d) (B : Set) (p : B)
      → subst (λ _ → B) x p ≡ p
    subFix refl _ _ = refl


QuD→QuE : {S : Setoid} {PQ : PreQu S} {QU : Qu PQ}
  → (QuD PQ) → (QuE QU)
QuD→QuE {S} {Q: Q [ ] : [_] sound: _}
  (emb: emb complete: complete stable: _) =
  record {exact = λ {a} {b} [ a ] ≡ [ b ]
    → ⟨ complete a ⟩₀
      ▶₀ subst (λ x → x ∼ b) (emb ⋆ ⟨ [ a ] ≡ [ b ] ⟩) (complete b)
```

```
      }
      where
        A        =  Carrier S
        _~_      =  _≈_ S
        ⟨_⟩₀  :  Symmetric _~_
        ⟨_⟩₀  =  symmetric S
        _▶₀_  :  Transitive _~_
        _▶₀_  =  transitive S


  QuD→Qu : {S : Setoid} → {PQ : PreQu S}
    → (QuD PQ) → (Qu PQ)
  QuD→Qu {S} {Q: Q [ ] : [_] sound: sound}
    (emb: ⌜_⌝ complete: complete stable: stable) =
    record
    {qelim  =  λ {B} f _ a → subst B (stable a) (f ⌜ a ⌝)
    ;qelim-β  =  λ {B} {a} {f} s
       → substIrr B (stable [a]) (sound (complete a)) (f ⌜ [a] ⌝)
       ▶ s _ _ (complete a)
    }


  QuD→QuH : {S : Setoid} → {PQ : PreQu S}
    → (QuD PQ) → (QuH PQ)
  QuD→QuH {S} {Q: Q [ ] : [_] sound: sound}
    (emb: ⌜_⌝ complete: complete stable: stable) =
    record
    {lift   =  λ f _ q → f ⌜ q ⌝
    ;lift-β  =  λ {B} {a} {f} {s} → s ⌜ [a] ⌝ a (complete a)
    ;qind  =  λ P _ f → λ x → subst P (stable x) (f ⌜ x ⌝)
    }
```

Or

```
  QuD→QuH' : {S : Setoid} → {PQ : PreQu S}
    → (QuD PQ) → (QuH PQ)
  QuD→QuH' {S}  =  Qu→QuH ∘ QuD→Qu
```