GUARDED RECURSIVE TYPE THEORY VIA SIZED TYPES

Abstract. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Praesent convallis orci arcu, eu mollis dolor. Aliquam eleifend suscipit lacinia. Maecenas quam mi, porta ut lacinia sed, convallis ac dui. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Suspendisse potenti.

1. Introduction

[1]

2. Preliminaries

```
postulate funext : ∀{ℓ ℓ'} → Extensionality ℓ ℓ'

uip : ∀{ℓ} {A : Set ℓ} → {a a' : A}
  → {p p' : a ≡ a'} → p ≡ p'
uip {p = refl} {refl} = refl

data tag : Set where
  set : tag
  tot : tag

record PSh : Set₁ where
  field
    Obj : Size → Set
    Mor : (i : Size) (j : Size< (↑ i))
      → Obj i → Obj j
    MorId : {i : Size} {x : Obj i}
      → Mor i i x ≡ x
    MorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
      → {x : Obj i}
      → Mor i k x ≡ Mor j k (Mor i j x)

ConstObj : Size → Set
ConstObj  = A

ConstMor : (i : Size) (j : Size< (↑ i))
  → ConstObj i → ConstObj j
ConstMor   x = x

ConstMorId : {i : Size} {x : A}
  → ConstMor i i x ≡ x
ConstMorId = refl
```

```
ConstMorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
  → {x : ConstObj i}
  → ConstMor i k x ≡ ConstMor j k (ConstMor i j x)
ConstMorComp = refl

Terminal : PSh
Terminal = Const ⊤

ProdObj : Size → Set
ProdObj i = Obj P i × Obj Q i

ProdMor : (i : Size) (j : Size< (↑ i))
  → ProdObj i → ProdObj j
ProdMor i j = map (Mor P i j) (Mor Q i j)

ProdMorId : {i : Size} {x : ProdObj i}
  → ProdMor i i x ≡ x
ProdMorId {i} {x} =
  begin
    (Mor P i i (proj₁ x) , Mor Q i i (proj₂ x))
  ≡⟨ cong (λ z → (z , )) (MorId P) ⟩
    (proj₁ x , Mor Q i i (proj₂ x))
  ≡⟨ cong (λ z → ( , z)) (MorId Q) ⟩
    x
  ∎

ProdMorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
  → {x : ProdObj i}
  → ProdMor i k x ≡ ProdMor j k (ProdMor i j x)
ProdMorComp {i} {j} {k} {x} =
  begin
    (Mor P i k (proj₁ x) , Mor Q i k (proj₂ x))
  ≡⟨ cong (λ z → (z , )) (MorComp P) ⟩
    (Mor P j k (Mor P i j (proj₁ x)) , Mor Q i k (proj₂ x))
  ≡⟨ cong (λ z → ( , z)) (MorComp Q) ⟩
    (Mor P j k (Mor P i j (proj₁ x)) , Mor Q j k (Mor Q i j (proj₂ x)))
  ∎

SumObj : Size → Set
SumObj i = Obj P i ⊎ Obj Q i

SumMor : (i : Size) (j : Size< (↑ i))
  → SumObj i → SumObj j
SumMor i j = map (Mor P i j) (Mor Q i j)

SumMorId : {i : Size} {x : SumObj i}
  → SumMor i i x ≡ x
```

```
SumMorId {i} {inj₁ p} =
  begin
    inj₁ (Mor P i i p)
  ≡⟨ cong inj₁ (MorId P) ⟩
    inj₁ p
  ∎
SumMorId {i} {inj₂ q} =
  begin
    inj₂ (Mor Q i i q)
  ≡⟨ cong inj₂ (MorId Q) ⟩
    inj₂ q
  ∎

SumMorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
  → {x : SumObj i}
  → SumMor i k x ≡ SumMor j k (SumMor i j x)
SumMorComp {i} {j} {k} {inj₁ p} =
  begin
    inj₁ (Mor P i k p)
  ≡⟨ cong inj₁ (MorComp P) ⟩
    inj₁ (Mor P j k (Mor P i j p))
  ∎
SumMorComp {i} {j} {k} {inj₂ q} =
  begin
    inj₂ (Mor Q i k q)
  ≡⟨ cong inj₂ (MorComp Q) ⟩
    inj₂ (Mor Q j k (Mor Q i j q))
  ∎

ExpObj : Size → Set
ExpObj i =
  Σ ((j : Size< (↑ i)) → Obj P j → Obj Q j)
    (λ f → (j : Size< (↑ i)) (k : Size< (↑ j))
             (x : Obj P j)
               → Mor Q j k (f j x)
                 ≡
                 f k (Mor P j k x))

ExpMor : (i : Size) (j : Size< (↑ i))
  → ExpObj i → ExpObj j
ExpMor i j (f , p) = f , p

ExpMorId : {i : Size} {x : ExpObj i}
  → ExpMor i i x ≡ x
ExpMorId = refl
```

```
ExpMorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
  → {x : ExpObj i}
  → ExpMor i k x ≡ ExpMor j k (ExpMor i j x)
ExpMorComp = refl

Ctx : tag → Set₁
Ctx set = Set
Ctx tot = PSh

Ty : tag → Set₁
Ty set = Set
Ty tot = PSh

Tm : {b : tag} ( : Ctx b) (A : Ty b) → Set
Tm {set}  A =  → A
Tm {tot}  A =
  Σ ((i : Size) → PSh.Obj  i → PSh.Obj A i)
    (λ f → (i : Size) (j : Size< (↑ i)) (x : PSh.Obj  i)
        → PSh.Mor A i j (f i x) ≡ f j (PSh.Mor  i j x))


• : (b : tag) → Ctx b
• set = ⊤
• tot = Terminal

,, : {b : tag} → Ctx b → Ty b → Ctx b
,, {set}  A =  × A
,, {tot}  A = Prod  A

var : {b : tag} ( : Ctx b) (A : Ty b) → Tm ( ,, A) A
var {set}  A = proj₂
proj₁ (var {tot}  A) i (γ , x) = x
proj₂ (var {tot}  A) i j (γ , x) = refl

weaken : {b : tag} ( : Ctx b) (A B : Ty b)
  → Tm  B → Tm ( ,, A) B
weaken {set}  A B t (x , ) = t x
proj₁ (weaken {tot}  A B (t , p)) i (x₁ , x₂) = t i x₁
proj₂ (weaken {tot}  A B (t , p)) i j (x₁ , x₂) = p i j x₁

subst-Tm : {b : tag} ( : Ctx b) (A B : Ty b)
  → (t : Tm ( ,, A) B) ( : Tm  A)
  → Tm  B
subst-Tm {set}  A B t  x = t (x ,  x)
proj₁ (subst-Tm {tot}  A B (t , p) ( , q)) i x = t i (x ,  i x)
proj₂ (subst-Tm {tot}  A B (t , p) ( , q)) i j x =
  begin
    PSh.Mor B i j (t i (x ,  i x))
```

```
  ≡⟨ p i j (x ,  i x) ⟩
    t j (PSh.Mor ( ,, A) i j (x ,  i x))
  ≡⟨ cong (λ z → t j ( , z)) (q i j x) ⟩
    t j (PSh.Mor  i j x ,  j (PSh.Mor  i j x))
  ■

Unit : {b : tag} → Ty b
Unit {set} = ⊤
Unit {tot} = Terminal

⋆ : {b : tag} ( : Ctx b) → Tm  Unit
⋆ {set}  x = tt
proj₁ (⋆ {tot} ) i x = tt
proj₂ (⋆ {tot} ) i j x = refl

Unit-rec : {b : tag} ( : Ctx b) (A : Ty b)
  → Tm  A → Tm ( ,, Unit) A
Unit-rec {set}  A t x = t (proj₁ x)
proj₁ (Unit-rec {tot}  A t) i x = proj₁ t i (proj₁ x)
proj₂ (Unit-rec {tot}  A t) i j x =
  begin
    PSh.Mor A i j (proj₁ t i (proj₁ x))
  ≡⟨ proj₂ t i j (proj₁ x) ⟩
    proj₁ t j (proj₁ (ProdMor  Terminal i j x))
  ■

⊕ : {b : tag} (A B : Ty b) → Ty b
⊕ {set} A B = A ⊎ B
⊕ {tot} A B = Sum A B

inl : {b : tag} ( : Ctx b) (A B : Ty b) (x : Tm  A) → Tm  (A ⊕ B)
inl {set}  A B t x = inj₁ (t x)
proj₁ (inl {tot}  A B (x , p))  y = inj₁ (x  y)
proj₂ (inl {tot} A B (x , p))  ' y =
  begin
    inj₁ (Mor A  ' (x  y))
  ≡⟨ cong inj₁ (p  ' y) ⟩
    inj₁ (x ' (Mor  ' y))
  ■

inr : {b : tag} ( : Ctx b) (A B : Ty b) (x : Tm  B) → Tm  (A ⊕ B)
inr {set}  A B t x = inj₂ (t x)
proj₁ (inr {tot}  A B (x , p))  y = inj₂ (x  y)
proj₂ (inr {tot}  A B (x , p))  ' y =
  begin
    inj₂ (Mor B  ' (x  y))
```

```
  ≡⟨ cong inj₂ (p  ' y) ⟩
    inj₂ (x ' (Mor   ' y))
  ∎

sum-rec : {b : tag} ( : Ctx b) (A B C : Ty b)
            (left : Tm ( ,, A) C) (right : Tm ( ,, B) C)
            → Tm ( ,, (A ⊕ B)) C
sum-rec {set}  A B C left right (x , inj₁ l) = left (x , l)
sum-rec {set}  A B C left right (x , inj₂ r) = right (x , r)
proj₁ (sum-rec {tot}  A B C (left , p) (right , q)) i (x , inj₁ l) = left i (x , l)
proj₂ (sum-rec {tot}  A B C (left , p) (right , q)) i j (x , inj₁ l) =
  begin
    Mor C i j (left i (x , l))
  ≡⟨ p i j (x , l) ⟩
    left j (Mor  i j x , Mor A i j l)
  ∎
proj₁ (sum-rec {tot}  A B C (left , p) (right , q)) i (x , inj₂ r) = right i (x , r)
proj₂ (sum-rec {tot}  A B C (left , p) (right , q)) i j (x , inj₂ r) =
  begin
      Mor C i j (right i (x , r))
    ≡⟨ q i j (x , r) ⟩
      right j (Mor  i j x , Mor B i j r)
    ∎

sum-rec-inl : {b : tag} ( : Ctx b) (A B C : Ty b)
  → (left : Tm ( ,, A) C) (right : Tm ( ,, B) C)
  → (x : Tm  A)
  → def-eq  C
            (subst-Tm   (sum-rec  A B C left right) (inl  A B x))
            (subst-Tm   left x)
sum-rec-inl {set}  A B C left right x z = refl
sum-rec-inl {tot}  A B C (left , p) (right , q) (x , r) i z = refl

sum-rec-inr : {b : tag} ( : Ctx b) (A B C : Ty b)
  → (left : Tm ( ,, A) C) (right : Tm ( ,, B) C)
  → (x : Tm  B)
  → def-eq  C
            (subst-Tm   (sum-rec  A B C left right) (inr  A B x))
            (subst-Tm   right x)
sum-rec-inr {set}  A B C left right x z = refl
sum-rec-inr {tot}  A B C (left , p) (right , q) (x , r) i z = refl

⊗ : {b : tag} (A B : Ty b) → Ty b
⊗ {set} A B = A × B
⊗ {tot} A B = Prod A B
```

```
pair : {b : tag} (Γ : Ctx b) (A B : Ty b) (x : Tm Γ A) (y : Tm Γ B)
  → Tm Γ (A ⊗ B)
pair {set} Γ A B x y t = x t , y t
proj₁ (pair {tot} Γ A B (x , p) (y , q)) i t = (x i t) , (y i t)
proj₂ (pair {tot} Γ A B (x , p) (y , q)) i j t =
  begin
    (Mor A i j (x i t) , Mor B i j (y i t))
  ≡⟨ cong (λ z → (z , _)) (p i j t) ⟩
    (x j (Mor Γ i j t) , Mor B i j (y i t))
  ≡⟨ cong (λ z → (_ , z)) (q i j t) ⟩
    (x j (Mor Γ i j t) , y j (Mor Γ i j t))
  ∎

pr₁ : {b : tag} (Γ : Ctx b) (A B : Ty b) → Tm Γ (A ⊗ B) → Tm Γ A
pr₁ {set} Γ A B x t = proj₁ (x t)
proj₁ (pr₁ {tot} Γ A B (x , p)) i t = proj₁ (x i t)
proj₂ (pr₁ {tot} Γ A B (x , p)) i j t =
  begin
    Mor A i j (proj₁ (x i t))
  ≡⟨ cong proj₁ (p i j t) ⟩
    proj₁ (x j (Mor Γ i j t))
  ∎

pr₂ : {b : tag} (Γ : Ctx b) (A B : Ty b) → Tm Γ (A ⊗ B) → Tm Γ B
pr₂ {set} Γ A B x t = proj₂ (x t)
proj₁ (pr₂ {tot} Γ A B (x , p)) i t = proj₂ (x i t)
proj₂ (pr₂ {tot} Γ A B (x , p)) i j t =
  begin
    Mor B i j (proj₂ (x i t))
  ≡⟨ cong proj₂ (p i j t) ⟩
    proj₂ (x j (Mor Γ i j t))
  ∎

pr₁-pair : {b : tag} (Γ : Ctx b) (A B : Ty b) (x : Tm Γ A) (y : Tm Γ B)
  → def-eq Γ A
           (pr₁ Γ A B (pair Γ A B x y))
           x
pr₁-pair {set} Γ A B x y t = refl
pr₁-pair {tot} Γ A B x y i t = refl

pr₂-pair : {b : tag} (Γ : Ctx b) (A B : Ty b) (x : Tm Γ A) (y : Tm Γ B)
  → def-eq Γ B
           (pr₂ Γ A B (pair Γ A B x y))
           y
pr₂-pair {set} Γ A B x y t = refl
pr₂-pair {tot} Γ A B x y i t = refl
```

```
prod-eta : {b : tag} ( : Ctx b) (A B : Ty b) (x : Tm  (A ⊗ B))
  → def-eq  (A ⊗ B)
             (pair  A B (pr₁  A B x) (pr₂  A B x))
             x
prod-eta {set}  A B x t = refl
prod-eta {tot}  A B x i t = refl


⇒ : {b : tag} (A B : Ty b) → Ty b
⇒ {set} A B = A → B
⇒ {tot} A B = Exp A B


lambda : {b : tag} ( : Ctx b) (A B : Ty b) (t : Tm ( ,, A) B)
  → Tm  (A ⇒ B)
lambda {set}  A B t x y = t (x , y)
proj₁ (proj₁ (lambda {tot}  A B (t , p)) i x) j z = t j (Mor  i j x , z)
proj₂ (proj₁ (lambda {tot}  A B (t , p)) i x) j k y =
  begin
    Mor B j k (t j (Mor  i j x , y))
  ≡⟨ p j k (Mor  i j x , y) ⟩
    t k (Mor ( ,, A) j k (Mor  i j x , y))
  ≡⟨ cong (λ z → t k (z , )) (sym (MorComp )) ⟩
    t k (Mor  i k x , Mor A j k y)
  ■
proj₂ (lambda {tot}  A B (t , p)) i j x =
  Σ≡-uip
    (funext (λ  → funext (λ  → funext (λ  → uip))))
    (funext (λ k → (funext (λ z → cong (λ z → t k (z , )) (MorComp )))))


app : {b : tag} ( : Ctx b) (A B : Ty b)
      (f : Tm  (A ⇒ B)) (t : Tm  A)
  → Tm  B
app {set}  A B f t x = f x (t x)
proj₁ (app {tot}  A B (f , p) (t , q)) i x =
  let (f' , ) = f i x in
  f' (t i x)
proj₂ (app {tot}  A B (f , p) (t , q)) i j x =
  let (f' , p') = f i x in
  begin
    Mor B i j (proj₁ (f i x)  (t i x))
  ≡⟨ p' i j (t i x) ⟩
    proj₁ (f i x) j (Mor A i j (t i x))
  ≡⟨ cong₂ (λ z g → proj₁ g  z) (q i j x) (p i j x) ⟩
    proj₁ (f j (Mor  i j x))  (t j (Mor  i j x))
  ■
```

```
beta : {b : tag} {Γ : Ctx b} {A B : Ty b} (t : Tm (Γ ,, A) B) (x : Tm Γ A)
    → def-eq Γ B
             (app Γ A B (lambda Γ A B t) x)
             (subst-Tm Γ A B t x)
beta {set} t x = refl
beta {tot} {Γ} (t , p) (x , q) i γ =
  begin
    t i (Mor Γ i i γ , x i γ)
  ≡⟨ cong (λ z → t i (z , x i γ)) (MorId Γ) ⟩
    t i (γ , x i γ)
  ∎

eta : {b : tag} {Γ : Ctx b} {A B : Ty b} (t : Tm Γ (A ⇒ B))
  → def-eq Γ (A ⇒ B)
           (lambda Γ A B (app (Γ ,, A) A B (weaken Γ A (A ⇒ B) t) (var Γ A)))
           t
eta {set} t x = refl
eta {tot} (t , p) i x =
  Σ≡-uip
    (funext (λ j → funext (λ _ → funext (λ _ → uip))))
    (funext (λ j' → funext (λ z → sym (cong (λ h → proj₁ h j' z) (p i j' x)))))

□ : Ty tot → Ty set
□ A = Σ ((i : Size) → Obj A i)
         (λ x → (i : Size) (j : Size< (↑ i))
             → Mor A i j (x i) ≡ x j)

box : (Γ : Ctx set) (A : Ty tot) (t : Tm (WC Γ) A) → Tm Γ (□ A)
proj₁ (box Γ A (t , p) x) i = t i x
proj₂ (box Γ A (t , p) x) i j = p i j x

unbox : (Γ : Ctx set) (A : Ty tot) (t : Tm Γ (□ A)) → Tm (WC Γ) A
proj₁ (unbox Γ A t) i x = proj₁ (t x) i
proj₂ (unbox Γ A t) i j x = proj₂ (t x) i j

box-beta : {Γ : Ctx set} {A : Ty tot} (t : Tm (WC Γ) A)
  → def-eq (WC Γ) A (unbox Γ A (box Γ A t)) t
box-beta t i x = refl

box-eta : {Γ : Ctx set} {A : Ty tot} (t : Tm Γ (□ A))
  → def-eq Γ (□ A) (box Γ A (unbox Γ A t)) t
box-eta t i = refl

data SizeLt (i : Size) : Set where
  [_] : (j : Size< i) → SizeLt i
```

```
size : ∀ {i} → SizeLt i → Size
size [ j ] = j

elimLt : ∀{ℓ} {A : Size → Set ℓ} {i : Size} (j : SizeLt i)
  → ((j : Size< i) → A j) → A (size j)
elimLt [ j ] f = f j

Later : (Size → Set) → Size → Set
Later A i = (j : SizeLt i) → A (size j)

module (A : Size → Set) (m : (i : Size) (j : Size< (↑ i)) → A i → A j) where

  LaterLim : (i : Size) (x : Later A i) → Set
  LaterLim i x = (j : SizeLt i)
    → elimLt j (λ { j' → (k : SizeLt (↑ j'))
      → elimLt k (λ k' → m j' k' (x [ j' ]) ≡ x [ k' ]) })

  LaterLimMor : (i : Size) (j : Size< (↑ i)) (x : Later A i)
    → LaterLim i x → LaterLim j x
  LaterLimMor i j x p [ k ] [ l ] = p [ k ] [ l ]

module (A : Ty tot) where

  -- 3.  Object part
  ▷Obj : (i : Size) → Set
  ▷Obj i = Σ (Later (PSh.Obj A) i) (LaterLim (PSh.Obj A) (PSh.Mor A) i)

  -- 4.  Morphism part
  ▷Mor : (i : Size) (j : Size< (↑ i))
    → ▷Obj i → ▷Obj j
  ▷Mor i j (x , p) = x , LaterLimMor (PSh.Obj A) (PSh.Mor A) i j x p
    where
      p' : LaterLim (PSh.Obj A) (PSh.Mor A) j x
      p' [ j ] [ k ] = p [ j ] [ k ]

  -- 5.  Preservation of identity
  ▷MorId : {i : Size} {x : ▷Obj i}
           → ▷Mor i i x ≡ x
  ▷MorId = Σ≡-uip (funext (λ { [ j ] → funext (λ { [ k ] → uip }) })) refl

  -- 6.  Preservation of composition
  ▷MorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)} {x : ▷Obj i}
               → ▷Mor i k x ≡ ▷Mor j k (▷Mor i j x)
  ▷MorComp = Σ≡-uip (funext (λ { [ j ] → funext (λ { [ k ] → uip }) })) refl

  ▷ : Ty tot
```

```
  ▷ = record
    { Obj = ▷Obj
    ; Mor = ▷Mor
    ; MorId = ▷MorId
    ; MorComp = ▷MorComp
    }

pure : (Γ : Ctx tot) (A : Ty Γ tot) (t : Tm Γ A) → Tm Γ (▷ A)
proj₁ (proj₁ (pure Γ A (t , _)) i x) [ j ] = t j (Mor Γ i j x)
proj₂ (proj₁ (pure Γ A (t , p)) i x) [ j ] [ k ] =
  begin
    Mor A j k (t j (Mor Γ i j x))
  ≡⟨ p j k (Mor Γ i j x) ⟩
    t k (Mor Γ j k (Mor Γ i j x))
  ≡⟨ cong (t k) (sym (MorComp Γ)) ⟩
    t k (Mor Γ i k x)
  ■
proj₂ (pure Γ A (t , p)) i j x =
  Σ≡-uip
    (funext (λ { [ _ ] → funext (λ { [ _ ] → uip }) }))
    (funext (λ { [ k ] → cong (t k) (MorComp Γ) }))

fmap : (Γ : Ctx tot) (A B : Ty Γ tot)
         → (f : Tm Γ (▷ (A ⇒ B))) (t : Tm Γ (▷ A))
         → Tm Γ (▷ B)
proj₁ (proj₁ (fmap Γ A B (f , _) (t , _)) i x) [ j ] = proj₁ (proj₁ (f i x) [ j ]) j (proj₁ (t i x) [ j ])
proj₂ (proj₁ (fmap Γ A B (f , p) (t , q)) i x) [ j ] [ k ] =
  begin
    Mor B j k (proj₁ (proj₁ (f i x) [ j ]) j (proj₁ (t i x) [ j ]))
  ≡⟨ proj₂ (proj₁ (f i x) [ j ]) j k (proj₁ (t i x) [ j ]) ⟩
    proj₁ (proj₁ (f i x) [ j ]) k (Mor A j k (proj₁ (t i x) [ j ]))
  ≡⟨ cong (proj₁ (proj₁ (f i x) [ j ]) k) (proj₂ (t i x) [ j ] [ k ]) ⟩
    proj₁ (proj₁ (f i x) [ j ]) k (proj₁ (t i x) [ k ])
  ≡⟨ cong (λ z → proj₁ z k (proj₁ (t i x) [ k ])) (sym (proj₂ (f i x) [ j ] [ j ])) ⟩
    proj₁ (Mor (A ⇒ B) j j (proj₁ (f i x) [ j ])) k (proj₁ (t i x) [ k ])
  ≡⟨ cong (λ z → proj₁ z k (proj₁ (t i x) [ k ])) (proj₂ (f i x) [ j ] [ k ]) ⟩
    proj₁ (proj₁ (f i x) [ k ]) k (proj₁ (t i x) [ k ])
  ■
proj₂ (fmap Γ A B (f , p) (e , q)) i j x =
  Σ≡-uip
    (funext (λ { [ _ ] → funext (λ { [ _ ] → uip })}))
    (funext (λ { [ k ] → cong₂ (λ a b → proj₁ (proj₁ a [ k ]) k (proj₁ b [ k ])) (p i j x) (q i j x) }))

pure-fmap-pure : (Γ : Ctx tot) (A B : Ty Γ tot)
  → (f : Tm Γ (A ⇒ B)) (t : Tm Γ A)
  → def-eq Γ (▷ B)
```

```
                (fmap  A B (pure  (A ⇒ B) f) (pure  A t))
                (pure  B (app  A B f t))
pure-fmap-pure  A B (f , p) (t , q) i x =
  Σ≡-uip
    (funext (λ { [ ] → funext (λ { [ ] → uip })}))
    (funext (λ { [ j ] → refl }))

pure-id-fmap : ( : Ctx tot) (A B : Ty tot) (t : Tm  (▷ A))
  → def-eq  (▷ A)
                (fmap  A A (pure  (A ⇒ A) (id-tm  A)) t)
                t
pure-id-fmap  A B (t , p) i ɣ =
  Σ≡-uip
    (funext (λ { [ ] → funext (λ { [ ] → uip })}))
    (funext (λ { [ j ] → refl }))

pure-comp-fmap : ( : Ctx tot) (A B C : Ty tot)
             → (g : Tm  (▷ (B ⇒ C))) (f : Tm  (▷ (A ⇒ B))) (t : Tm  (▷ A))
             → def-eq
                        (▷ C)
                        (fmap  A C
                              (fmap  (A ⇒ B) (A ⇒ C)
                                    (fmap  (B ⇒ C) ((A ⇒ B) ⇒ (A ⇒ C))
                                          (pure  ((B ⇒ C) ⇒ ((A ⇒ B) ⇒ (A ⇒ C))) (comp-tm  A B C))
                                          g)
                                    f)
                              t)
                        (fmap  B C g (fmap  A B f t))
pure-comp-fmap  A B C g f t i ɣ =
  Σ≡-uip (funext (λ { [ ] → funext (λ { [ ] → uip })}))
           (funext (λ { [ j ] → refl}))

fmap-pure-fun : ( : Ctx tot) (A B : Ty tot)
  → (f : Tm  (▷ (A ⇒ B))) (t : Tm  A)
  → def-eq
                (▷ B)
                (fmap  A B f (pure  A t))
                (fmap  (A ⇒ B) B
                       (pure  ((A ⇒ B) ⇒ B)
                              (lambda  (A ⇒ B) B
                                       (app ( ,, (A ⇒ B)) A B
                                            (var  (A ⇒ B))
                                            (weaken  (A ⇒ B) A t))))
                        f)
fmap-pure-fun  A B (f , p) (t , q) i ɣ =
  Σ≡-uip
```

```
    (funext (λ { [ ] → funext (λ { [ ] → uip })}))
    (funext (λ { [ j ] → cong (λ a → proj₁ (proj₁ (f i γ) [ j ]) j (t j a)) (sym (MorId ))}))

WC : Ty set → Ty tot
WC A = Const A

WC-fun : ( : Ctx set) (A : Ty set) → Tm  A → Tm (WC ) (WC A)
proj₁ (WC-fun  A t)  = t
proj₂ (WC-fun  A t)    = refl

WC-unfun : ( : Ctx set) (A : Ty set) → Tm (WC ) (WC A) → Tm  A
WC-unfun  A (t , p) = t ∞

dfix₁ : (A : Ty tot) (i : Size) → ExpObj (▷ A) A i → ▷Obj A i
proj₁ (dfix₁ A i (f , p)) [ j ] = f j (dfix₁ A j (f , p))
proj₂ (dfix₁ A i (f , p)) [ j ] [ k ] =
  begin
    Mor A j k (f j (dfix₁ A j (f , p)))
  ≡⟨ p j k (dfix₁ A j (f , p)) ⟩
    f k (▷Mor A j k (dfix₁ A j (f , p)))
  ≡⟨ cong (f k) (Σ≡-uip (funext (λ { [ j ] → funext (λ { [ k ] → uip }) })) (funext (λ { [ ] → refl}))) ⟩
    f k (dfix₁ A k (f , p))
  ∎

dfix : ( : Ctx tot) (A : Ty tot) (f : Tm  (▷ A ⇒ A)) → Tm  (▷ A)
proj₁ (dfix  A (f , )) i γ = dfix₁ A i (f i γ)
proj₂ (dfix  A (f , p)) i j γ =
  Σ≡-uip (funext (λ { [ j ] → funext (λ { [ k ] → uip }) }))
         (funext (λ { [ k ] → cong (λ a → proj₁ a k (dfix₁ A k (proj₁ a , proj₂ a))) (p i j γ) }))

fix : ( : Ctx tot) (A : Ty tot) (f : Tm  (▷ A ⇒ A)) → Tm  A
fix  A f = app  (▷ A) A f (dfix  A f)

dfix-eq : ( : Ctx tot) (A : Ty tot) (f : Tm  (▷ A ⇒ A))
  → def-eq {tot}  (▷ A) (dfix  A f) (pure  A (fix  A f))
dfix-eq  A (f , p) i γ =
  Σ≡-uip
    (funext (λ { [ j ] → funext (λ { [ k ] → uip }) }))
    (funext (λ { [ j ] → cong (λ a → proj₁ a j (dfix₁ A j (proj₁ a , proj₂ a))) (p i j γ)}))

fix-eq : ( : Ctx tot) (A : Ty tot) (f : Tm  (▷ A ⇒ A))
  → def-eq  A
           (fix  A f)
           (app  (▷ A) A f (pure  A (fix  A f)))
fix-eq  A f i x = cong (proj₁ (proj₁ f i x) i) (dfix-eq  A f i x)

force-tm : ( : Ctx set) (A : Ty tot) (t : Tm  (□ (▷ A))) → Tm  (□ A)
proj₁ (force-tm  A t x) j = proj₁ (proj₁ (t x) ∞) [ j ]
```

```
proj₂ (force-tm A t x) i j =
  begin
    PSh.Mor A i j (proj₁ (proj₁ (t x) ∞) [ i ])
  ≡⟨ proj₂ (proj₁ (t x) ∞) [ i ] [ j ] ⟩
    proj₁ (proj₁ (t x) ∞) [ j ]
  ∎
```

2.1. Sized Types.

2.2. Guarded Recursive Type Theory.

3. The Model

3.1. Clock Contexts.

3.2. Presheaves.

4. The Interpretation

4.1. Types, Contexts, Terms.

4.2. Operations on Contexts.

4.3. Substitution.

4.4. Simple Types.

4.5. Later.

4.6. Clock Quantification.

4.7. Fix.

4.8. Inductive Types.

5. Conclusion

Appendix A. Omitted Proofs

References

[1] Robert Atkey and Conor McBride. Productive coprogramming with guarded recursion. In *ACM SIGPLAN Notices*, volume 48, pages 197–208. ACM, 2013.