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

[?]

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

Σ≡ : {A : Set}{P : A → Set}
  → {a a' : A} {p : P a} {p' : P a'}
  → (e : a ≡ a') → subst P e p ≡ p' → (a , p) ≡ (a' , p')
Σ≡ refl refl = refl

isProp : Set → Set
isProp P = {x y : P} → x ≡ y

Σ≡-uip : {A : Set}{P : A → Set}
  → ({a : A} → isProp (P a))
  → {a a' : A} {p : P a} {p' : P a'}
  → (e : a ≡ a') → (a , p) ≡ (a' , p')
Σ≡-uip q refl = cong (, ) q
```

Dependent functions preserve equality

```
cong-dep : {A : Set}{P : A → Set}
  → (f : (a : A) → P a)
  → {x y : A}
  → (e : x ≡ y) → subst P e (f x) ≡ f y
cong-dep f refl = refl
```

Functions with two arguments preserve equality

```
cong₂-dep : {A B : Set}{P : A → Set}
  → (f : (a : A) (p : P a) → B)
  → {a a' : A} {p : P a} {p' : P a'}
  → (e : a ≡ a') → subst P e p ≡ p' → f a p ≡ f a' p'
cong₂-dep f refl refl = refl
```

Transport of a composition

```
subst-trans : {A : Set}{P : A → Set}
  → {x y z : A}(e : x ≡ y)(e' : y ≡ z)
  → {p : P x}
  → subst P e' (subst P e p) ≡ subst P (trans e e') p
subst-trans refl refl = refl

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

Const : PSh
Const = record
  { Obj = ConstObj
  ; Mor = ConstMor
  ; MorId = ConstMorId
```

```
  ; MorComp = ConstMorComp
  }

Terminal : PSh
Terminal = Const ⊤

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

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

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

Prod : PSh
Prod = record
  { Obj = ProdObj
  ; Mor = ProdMor
  ; MorId = ProdMorId
  ; MorComp = ProdMorComp
  }

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

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

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

Sum : PSh
Sum = record
  { Obj = SumObj
  ; Mor = SumMor
  ; MorId = SumMorId
  ; MorComp = λ {}{}{}{x} → SumMorComp {x = x}
  }

ExpObj : Size → Set
ExpObj i =
  Σ ((j : Size< (↑ i)) → P.Obj j → Q.Obj j)
```

```
    (λ f → (j : Size< (↑ i)) (k : Size< (↑ j))
            (x : P.Obj j)
              → Q.Mor j k (f j x)
                  ≡
                f k (P.Mor j k x))

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

Exp : PSh
Exp = record
  { Obj = ExpObj
  ; Mor = ExpMor
  ; MorId = ExpMorId
  ; MorComp = ExpMorComp
  }

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 b ( ,, 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  B → Tm b ( ,, 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 b ( ,, A) B) (x : Tm b  A)
  → Tm b  B
subst-Tm {set} t x y = t (y , (x y))
proj₁ (subst-Tm {tot} (t , p) (x , q)) i y = t i (y , x i y)
proj₂ (subst-Tm {tot} {} {A} {B} (t , p) (x , q)) i j y =
  begin
    PSh.Mor B i j (t i (y , x i y))
  ≡⟨ p i j (y , x i y) ⟩
    t j (PSh.Mor ( ,, A) i j (y , x i y))
  ≡⟨ cong (λ z → t j ( , z)) (q i j y) ⟩
    t j (PSh.Mor  i j y , x j (PSh.Mor  i j y))
  ■

Unit : (b : tag) → Ty b
Unit set = ⊤
Unit tot = Terminal

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

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

```
    (x j (PSh.Mor  i j t) , y j (PSh.Mor  i j t))
  ■

pr₁ : {b : tag} ( : Ctx b) (A B : Ty b) → Tm b  (A ⊗ B) → Tm b  A
pr₁ {set}  A B x t = proj₁ (x t)
proj₁ (pr₁ {tot}  A B (x , p)) i t = proj₁ (x i t)
proj₂ (pr₁ {tot}  A B (x , p)) i j t = cong proj₁ (p i j t)

pr₂ : {b : tag} ( : Ctx b) (A B : Ty b) → Tm b  (A ⊗ B) → Tm b  B
pr₂ {set}  A B x t = proj₂ (x t)
proj₁ (pr₂ {tot}  A B (x , p)) i t = proj₂ (x i t)
proj₂ (pr₂ {tot}  A B (x , p)) i j t = cong proj₂ (p i j t)

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

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

prod-eta : {b : tag} ( : Ctx b) (A B : Ty b) (x : Tm b  (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 b ( ,, A) B) → Tm b  (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)
  where open PSh  renaming (Mor to Mor;MorComp to MorComp)
proj₂ (proj₁ (lambda {tot}  A B (t , p)) i x) j k y =
  begin
    PSh.Mor B j k (t j (Mor i j x , y))
  ≡⟨ p j k (Mor i j x , y) ⟩
    t k (PSh.Mor ( ,, A) j k (Mor i j x , y))
```

```
  ≡⟨ cong (λ z → t k (z , )) (sym MorComp) ⟩
    t k (Mor i k x , PSh.Mor A j k y)
  ∎
  where open PSh  renaming (Mor to Mor;MorComp to MorComp)
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))))
  where open PSh  renaming (Mor to Mor;MorComp to MorComp)

app : {b : tag} { : Ctx b} {A B : Ty b} (f : Tm b  (A ⇒ B)) (t : Tm b  A) → Tm b  B
app {set} f t x = f x (t x)
proj₁ (app {tot} {} (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
    PSh.Mor B i j (proj₁ (f i x)  (t i x))
  ≡⟨ p' i j (t i x) ⟩
    proj₁ (f i x) j (PSh.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 (PSh.Mor  i j x))  (t j (PSh.Mor  i j x))
  ∎

beta : {b : tag} { : Ctx b} {A B : Ty b} (t : Tm b ( ,, A) B) (x : Tm b  A)
     → def-eq  B (app {b} {} {A} {B} (lambda  A B t) x) (subst-Tm {} {} {A} {B} t x)
beta {set} t x  = refl
beta {tot} {} (t , p) (x , q)  z = cong (λ z → t  (z , )) MorId
  where open PSh  renaming (MorId to MorId)

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

id-tm : {b : tag} ( : Ctx b) (A : Ty b) → Tm b  (A ⇒ A)
id-tm  A = lambda  A A (var  A)

comp-tm : {b : tag} ( : Ctx b) (A B C : Ty b)
  → Tm b  ((B ⇒ C) ⇒ ((A ⇒ B) ⇒ (A ⇒ C)))
comp-tm  A B C = lambda  (B ⇒ C) ((A ⇒ B) ⇒ (A ⇒ C))
                         (lambda ( ,, (B ⇒ C)) (A ⇒ B) (A ⇒ C)
                                 (lambda (( ,, (B ⇒ C)) ,, (A ⇒ B)) A C (app
                                         (weaken (( ,, (B ⇒ C)) ,, (A ⇒ B)) A (B ⇒ C) (weaken ( ,, (B =
```

```
                                                (app (weaken ((( ,, (B ⇒ C)) ,, (A ⇒ B)) A (A ⇒ B) (var ( ,, (B
                                                    (var ((( ,, (B ⇒ C)) ,, (A ⇒ B))) A)))))

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

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

fmap : ( : Ctx tot) (A B : Ty tot)
         → (f : Tm tot  (▷ (A ⇒ B))) (t : Tm tot  (▷ A))
         → Tm tot  (▷ B)
proj₁ (proj₁ (fmap  A B (f , ) (t , )) i x) [ j ] = proj₁ (proj₁ (f i x) [ j ]) j (proj₁ (t i x) [ j ])
proj₂ (proj₁ (fmap  A B (f , p) (t , q)) i x) [ j ] [ k ] =
  begin
    PSh.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 (PSh.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 (PSh.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 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 : ( : Ctx tot) (A B : Ty tot)
  → (f : Tm tot (A ⇒ B)) (t : Tm tot A)
  → def-eq (▷ B) (fmap A B (pure (A ⇒ B) f) (pure A t)) (pure B (app {tot}{}{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 tot (▷ A))
         → def-eq (▷ A) (fmap A A (pure (A ⇒ A) (id-tm A)) t) t
pure-id-fmap A B (t , p) i γ =
  Σ≡-uip (funext (λ { [ ] → funext (λ { [ ] → uip })}))
         (funext (λ { [ j ] → refl }))

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

fmap-pure-fun : ( : Ctx tot) (A B : Ty tot)
  → (f : Tm tot (▷ (A ⇒ B))) (t : Tm tot 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
fmap-pure-fun A B (f , p) (t , q) i γ =
  Σ≡-uip (funext (λ { [ ] → funext (λ { [ ] → uip })}))
         (funext (λ { [ j ] → cong (λ a → proj₁ (proj₁ (f i γ) [ j ]) j (t j a)) (sym (PSh.MorId ))}))

WCObj : Size → Set
WCObj = A

WCMor : (i : Size) (j : Size< (↑ i))
  → WCObj i → WCObj j
WCMor x = x
```

```
WCMorId : {i : Size} {x : WCObj i}
  → WCMor i i x ≡ x
WCMorId = refl

WCMorComp : {i : Size} {j : Size< (↑ i)} {k : Size< (↑ j)}
  {x : WCObj i}
  → WCMor i k x ≡ WCMor j k (WCMor i j x)
WCMorComp = refl

WC : Ty tot
WC = record
  { Obj = WCObj
  ; Mor = WCMor
  ; MorId = WCMorId
  ; MorComp = WCMorComp
  }

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

WC-unfun : ( : Ctx set) (A : Ty set) → Tm tot (WC ) (WC A) → Tm set  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
    PSh.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 tot  (▷ A ⇒ A)) → Tm tot  (▷ 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 tot  (▷ A ⇒ A)) → Tm tot  A
fix  A f = app {tot} {} {▷ A} {A} f (dfix  A f)

dfix-eq : ( : Ctx tot) (A : Ty tot) (f : Tm tot  (▷ 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 y)}))

fix-eq : ( : Ctx tot) (A : Ty tot) (f : Tm tot (▷ A ⇒ A))
  → def-eq {tot} A (fix A f) (app {tot} {} {▷ 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 set (□ (▷ A))) → Tm set (□ 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 ]
  ■
```


References

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