Dependent paths in HoTT

(Topology Seminar, University of Lousiana at Lafayette)

Jonathan Prieto-Cubides

Universitetet i Bergen
Bergen, Norway

14th December, 2018

Updated: December 14, 2018

In this talk, we show evidence of the geometrical intuition in HoTT behind the dependent paths, also called pathovers, which has been only mentioned but not proved as far as we know. We provide two proofs type-checked in Agda. For the latter, we prove three lemmas that makes a shorter proof.

Type theory (HoTT book [3])

Type universes

We postulate a hierarchy of **universes** denoted by primitive constants

$$\mathcal{U}_0, \quad \mathcal{U}_1, \quad \mathcal{U}_2, \quad \ldots$$

The first two rules for universes say that they form a cumulative hierarchy of types:

- $\mathcal{U}_m : \mathcal{U}_n$ for $m < n$,
- if $A : \mathcal{U}_m$ and $m \leq n$, then $A : \mathcal{U}_n$,

Judgments: there are three kinds of judgments in TT.

$$\Gamma\ \mathrm{ctx} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a \equiv a' : A$$

Inference rule:

$$\frac{\mathcal{J}_1 \qquad \cdots \qquad \mathcal{J}_k}{\mathcal{J}}\ \text{Name}$$

- **hypotheses** $\mathcal{J}_1, \ldots, \mathcal{J}_k$
- **conclusion** $\mathcal{J}$

Contexts:

$$x_1 : A_1, x_2 : A_2, \ldots, x_n : A_n$$

The judgment Γ ctx formally expresses the fact that Γ is a well-formed context, and is governed by the rules of inference

$$\frac{}{\cdot \ \mathrm{ctx}}\ \text{ctx-emp} \qquad \frac{x_1 : A_1, \ldots, x_{n-1} : A_{n-1} \vdash A_n : \mathcal{U}_i}{(x_1 : A_1, \ldots, x_n : A_n)\ \mathrm{ctx}}\ \text{ctx-ext}$$

Structural rules:

$$\frac{(x_1 : A_1, \ldots, x_n : A_n)\ \mathrm{ctx}}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}\ \text{Vble}$$

The following important principles, called **substitution** and **weakening**, need not be explicitly assumed. For the typing judgments these principles are manifested as

$$\frac{\Gamma \vdash a : A \qquad \Gamma, x : A, \Delta \vdash b : B}{\Gamma, \Delta[a/x] \vdash b[a/x] : B[a/x]}\ \text{Subst}_1$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, \Delta \vdash b : B}{\Gamma, x : A, \Delta \vdash b : B}\ \text{Wkg}_1$$

Judgmental equality is an equivalence relation

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash a \equiv a : A} \qquad \frac{\Gamma \vdash a \equiv b : A}{\Gamma \vdash b \equiv a : A}$$

$$\frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash b \equiv c : A}{\Gamma \vdash a \equiv c : A}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a : B}$$

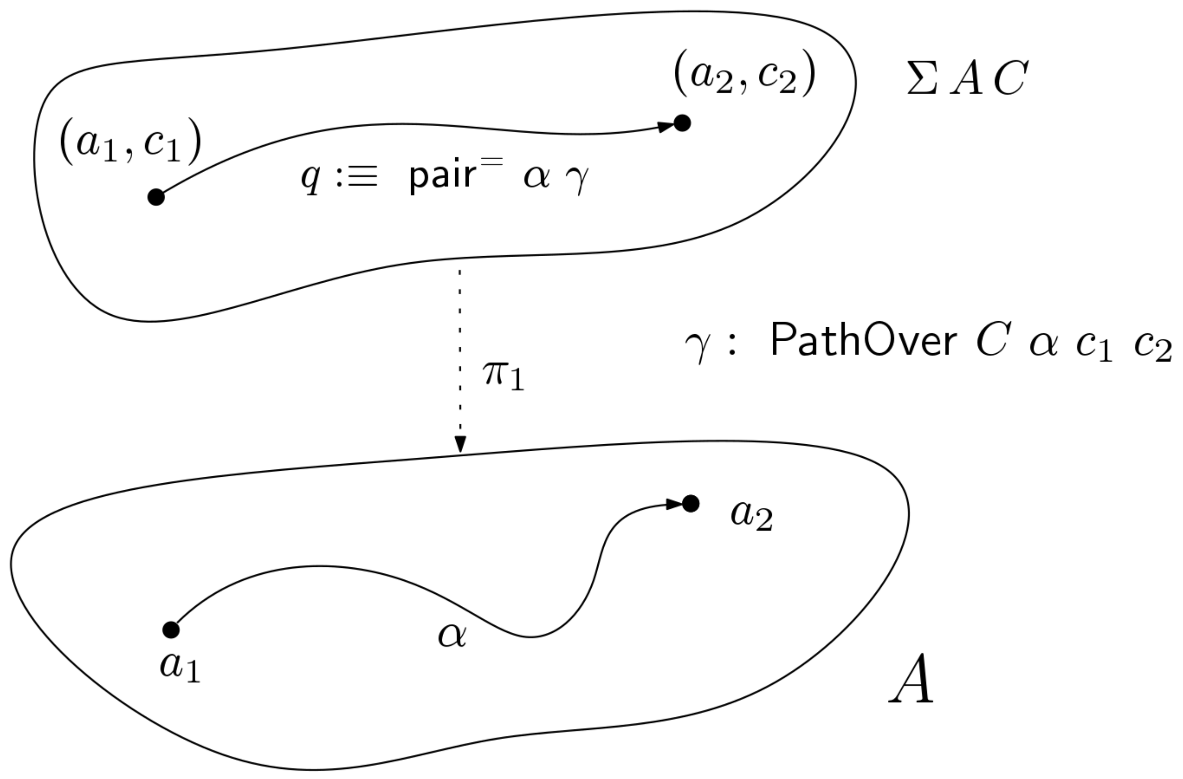
$$\frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a \equiv b : B}$$

A **derivation** of $\cdot \vdash \lambda x.\, x : \mathbf{1} \to \mathbf{1}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\cdot\ \mathrm{ctx}}\ \scriptstyle\text{ctx-emp}}{\vdash \mathbf{1} : \mathcal{U}_0}\ \scriptstyle\mathbf{1}\text{-form}}{x{:}\mathbf{1}\ \mathrm{ctx}}\ \scriptstyle\text{ctx-ext}}{x{:}\mathbf{1} \vdash x : \mathbf{1}}\ \scriptstyle\text{Vble}}{\cdot \vdash \lambda x.\, x : \mathbf{1} \to \mathbf{1}}\ \scriptstyle\Pi\text{-intro}$$

Dependent paths = pathovers.

Let $A : \mathcal{U}$, $a_1, a_2 : A$, $C : A \to \mathcal{U}$, $c_1 : C a_1$ and $c_2 : C a_2$. A *pathover* is a *term* that inhabits the type $({}_*(C)\, \alpha\, c_1) = c_2$ also denoted by $PathOver\ C\, \alpha\, c_1\, c_2$.


(a_1, c_1)
(a_2, c_2)
$q :\equiv \mathsf{pair}^{=}\ \alpha\ \gamma$
$\Sigma\, A\, C$
π_1
$\gamma :\ \mathsf{PathOver}\ C\ \alpha\ c_1\ c_2$
a_1
a_2
α
A

Agda

```
{-# OPTIONS --without-K #-}
open import Agda.Primitive  using ( Level ; lsuc; _⊔_ )
Type  : (ℓ : Level ) → Set (lsuc ℓ)
Type  ℓ = Set ℓ
```

Homogeneous equality:

The *homogeneous equality* is the Identity type denoted by Path that relates two elements a_0 and a_1 whose types are *definitionally/judgmentally* equal. We also refer to this type as $a_1 == a_2$ or Path $a_1 a_2$.

```
infix 30 _==_
data _==_ {ℓ} {A : Type ℓ}
  (a : A) : A → Type ℓ where

  idp : a == a

Path = _==_
```

Heterogeneous equality:

The heterogeneous equality as it is defined in [2] is a type for equality between two elements $a : A$, $b : B$, along an equality $\alpha : A = B$. Its terms are constructed by the reflexivity constructor which applies only when their types and terms are judgementally equal.

```
data HEq₁ {ℓ} (A : Type ℓ)
              : (B : Type ℓ)
              → (α : A == B) (a : A) (b : B)
              → Type ℓ where
  idp  : ∀ {a : A} → HEq₁ A A idp a a
```

This type can define it in other equivalent ways as the following.

To define those equivalent types to HEq_1, we use transport by path-induction or by using coercion (coe).

Transport:

```
transport
  : ∀ {ℓi ℓj} {A : Type ℓi} (C : A → Type ℓj)
  → {a b : A} → a == b
  → C a
  → C b
transport C idp = (  x → x)
```

Coercion:

```
coe
  : ∀ {ℓ}{A B : Type ℓ}
  → A == B
  → (A → B)
coe p A = transport (  X → X) p A
```

Heterogeneous Equality Types:

Let be $\alpha : A == B$, $a : A$, and $b : B$ then the following types are equivalent to the previous type HEq_1.

```
HEq₂
  : ∀ {ℓ} (A : Type ℓ)(B : Type ℓ)
  → (α : A == B)
  → (a : A)(b : B)
  ------------------
  → Type ℓ
HEq₂ A B α a b = Path (coe α a) b
```

$$
\begin{array}{l}
\mathrm{HEq}_3 \\
\quad : \forall \{\ell\}\,(\boldsymbol{A} : \mathrm{Type}\ \ell)(\boldsymbol{B} : \mathrm{Type}\ \ell) \\
\quad \to (\alpha : \boldsymbol{A} == \boldsymbol{B}) \\
\quad \to (a : A)(b : \boldsymbol{B}) \\
\quad \text{-----------------} \\
\quad \to \mathrm{Type}\ \ell \\
\mathrm{HEq}_3\ \boldsymbol{A}\ \boldsymbol{B}\ \alpha\ a\ b = \mathrm{Path}\ a\,(\mathrm{coe}\ (\mathrm{inv}\ \alpha)\ b)
\end{array}
$$

$$
\begin{aligned}
&\mathrm{HEq}_4 \\
&\quad : \forall\, \{\ell\}\, (\boldsymbol{A} : \mathrm{Type}\ \ell)(\boldsymbol{B} : \mathrm{Type}\ \ell) \\
&\quad \to (\alpha : \boldsymbol{A}\ \mathbf{==}\ \boldsymbol{B}) \\
&\quad \to (\boldsymbol{a} : \boldsymbol{A})(\boldsymbol{b} : \boldsymbol{B}) \\
&\quad \text{------------} \\
&\quad \to \mathrm{Type}\ \ell \\
&\mathrm{HEq}_4\ \ \boldsymbol{A}\ .\boldsymbol{A}\ \mathrm{idp}\ \ \boldsymbol{a}\,\boldsymbol{b} = \mathrm{Path}\ \ \boldsymbol{a}\,\boldsymbol{b}
\end{aligned}
$$

Here and below, the definition for Heterogeneous equality will be the HEq_1.

$$\mathrm{HEq}\ = \mathrm{HEq}_1$$

Equivalence between HEq₁ and HEq₂

```
-- HEq₁-≃-HEq₂
module _ {ℓ}(A : Type ℓ) (B : Type ℓ) where

-- Outgoing functions
  HEq₁-to-HEq₂
    : {α : A == B}{a : A}{b : B}
    → HEq₁ A B α a b
    → HEq₂ A B α a b

  HEq₁-to-HEq₂ { idp } idp = idp

  HEq₂-to-HEq₁
    : {α : A == B}{a : A}{b : B}
    → HEq₂ A B α a b
    → HEq₁ A B α a b

  HEq₂-to-HEq₁ { idp } idp = idp
```

Finally, we provide the evidence of the equivalence.

```
-- Equivalence
HEq₁−≃−HEq₂
  : {α : A == B}{a : A}{b : B}
  → HEq₁ A B α a b ≃ HEq₂ A B α a b

HEq₁−≃−HEq₂ { idp } {a} {b} =
  qinv−≃ HEq₁−to−HEq₂ ( HEq₂−to−HEq₁ , HEq₁−∼−HEq
  where
    HEq₁−∼−HEq₂ : ( p : HEq₂ A B idp a b)
                  → ( HEq₁−to−HEq₂ ( HEq₂−to−HEq₁ p )
    HEq₁−∼−HEq₂ idp = idp

    HEq₂−∼−HEq₁ : ( p : HEq₁ A B idp a b)
                  → ( HEq₂−to−HEq₁ ( HEq₁−to−HEq₂ p )
    HEq₂−∼−HEq₁ idp = idp
```

Paths in the total space (Dependent paths)

Pathover can be defined in at least five different ways:

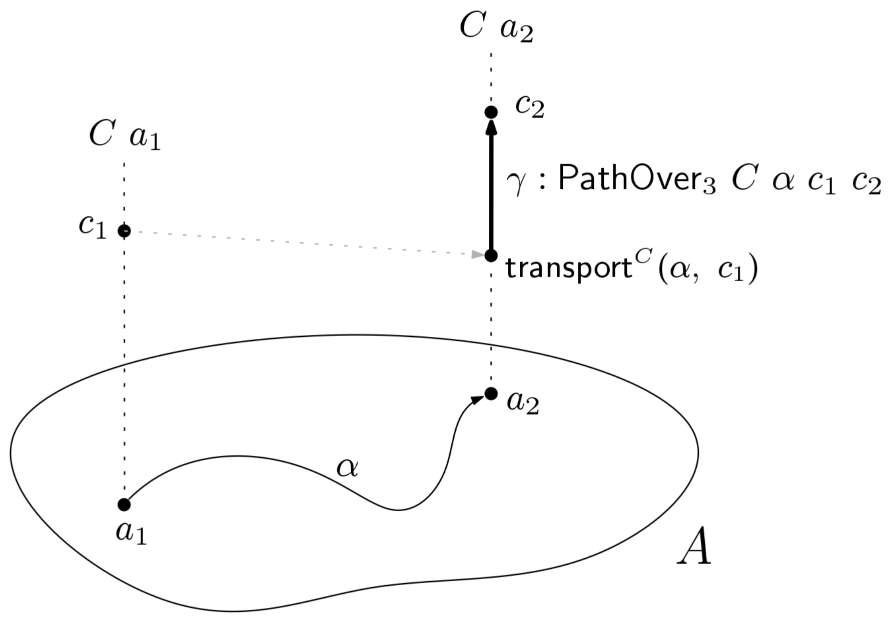
- Inductive type:

$$\begin{array}{l}
\mathrm{data}\ \mathrm{PathOver}_1\ \{\ell_i\,\ell_j\} \\
\quad \{A : \mathrm{Set}\ \ell_i\}\ (C : A \to \mathrm{Type}\ \ell_j)\ \{a_1 : A\} \\
\quad : \{a_2 : A\}\ (\alpha : a_1 == a_2) \\
\quad \to (c_1 : C\ a_1)\ (c_2 : C\ a_2) \to \mathrm{Type}\ \ell_j\ \mathrm{where} \\
\quad \mathrm{idp}\ : \forall\ \{c_1 : C\ a_1\} \to \mathrm{PathOver}_1\ C\ \mathrm{idp}\ c_1\ c_1
\end{array}$$

- Using Heterogeneous equality:

$$
\begin{aligned}
&\mathrm{PathOver}_2\\
&\quad : \forall \{\ell_i\ \ell_j\}\ \{A : \mathrm{Type}\ \ell_i\}\\
&\quad \to (C : A \to \mathrm{Type}\ \ell_j)\ \{a_1\ a_2 : A\}\\
&\quad \to (\alpha : a_1 == a_2)\\
&\quad \to (c_1 : C\ a_1)\\
&\quad \to (c_2 : C\ a_2)\\
&\quad \text{- - - - - - - - - - - - - - - - - - - -}\\
&\quad \to \mathrm{Type}\ \ell_j
\end{aligned}
$$

$$
\begin{aligned}
&\mathrm{PathOver}_2\ \{A = A\}\ C\ \{a_1\}\ \{a_2\}\ \alpha\ c_1\ c_2\\
&\quad = \mathrm{HEq}\ (C\ a_1)\ (C\ a_2)\ (\mathrm{ap}\ C\ \alpha)\ c_1\ c_2
\end{aligned}
$$



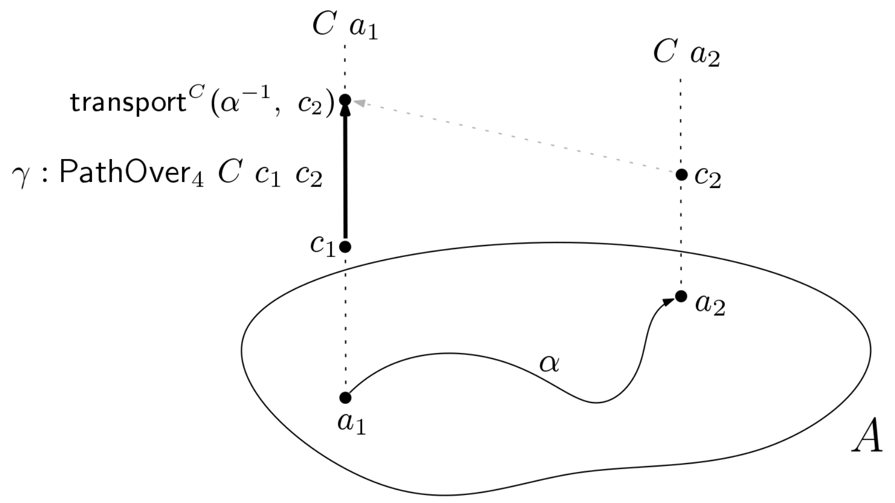
$C\ a_1$
c_1
$C\ a_2$
c_2
γ : PathOver$_3$ $C\ \alpha\ c_1\ c_2$
transport$^C(\alpha,\ c_1)$
a_2
α
a_1
A

Figure 1: *PathOver*$_3$

- Using Transport as in HoTT Book:

```
PathOver₃
  : ∀ {ℓᵢ ℓⱼ} {A : Type ℓᵢ}
  → (C : A → Type ℓⱼ) {a₁ a₂ : A}
  → (α : a₁ == a₂)
  → (c₁ : C a₁)
  → (c₂ : C a₂)
  --------------------
  → Type ℓⱼ

PathOver₃ C α c₁ c₂ = transport C α c₁ == c₂
```



$C\ a_1$
$C\ a_2$
$\mathsf{transport}^C(\alpha^{-1},\ c_2)$
$\gamma : \mathsf{PathOver}_4\ C\ c_1\ c_2$
c_1
c_2
a_2
α
a_1
A

Figure 2: *PathOver*$_4$

- Using Transport by reversing the path:

```
PathOver₄
  : ∀ {ℓᵢ ℓⱼ} {A : Type ℓᵢ}
  → (C : A → Type ℓⱼ) {a₁ a₂ : A}
  → (α : a₁ == a₂)
  → (c₁ : C a₁)
  → (c₂ : C a₂)
  --------------------
  → Type ℓⱼ

PathOver₄ C α c₁ c₂ = c₁ == transport C (α ⁻¹ ) c₂
```

- Using path-induction and an Identity type:

$$
\begin{aligned}
&\mathrm{PathOver}_5\\
&\quad : \forall\, \{\ell_i\ \ell_j\}\ \{A : \mathrm{Type}\ \ \ell_i\}\\
&\quad \to (C : A \to \mathrm{Type}\ \ \ell_j)\ \{a_1\ a_2 : A\}\\
&\quad \to (\alpha : a_1\ == \ a_2)\\
&\quad \to (c_1 : C\ a_1)\\
&\quad \to (c_2 : C\ a_2)\\
&\quad \text{-------------------}\\
&\quad \to \mathrm{Type}\ \ \ell_j\\
&\mathrm{PathOver}_5\ \_\ \mathrm{idp}\ \ c_1\ c_2 = c_1\ \ ==\ \ c_2
\end{aligned}
$$

Pathover Equivalences:

```
-- PathOver₁-≃-PathOver₂
module _ {ℓ} (A : Type ℓ) (C : A → Type ℓ) where

-- Outgoing functions
  PathOver₁-to-PathOver₂
    : ∀ {a₁ a₂ : A} {α : a₁ == a₂} {c₁ : C a₁}{c₂ : C a₂}
    → PathOver₁ C α c₁ c₂
    → PathOver₂ C α c₁ c₂

  PathOver₁-to-PathOver₂ {α = idp } idp = idp

  PathOver₂-to-PathOver₁
    : ∀ {a₁ a₂ : A} {α : a₁ == a₂} {c₁ : C a₁}{c₂ : C a₂}
    → PathOver₂ C α c₁ c₂
    → PathOver₁ C α c₁ c₂

  PathOver₂-to-PathOver₁ {α = idp } idp = idp
```

Finally, we provide the evidence of the equivalence.

```
-- Equivalence

  PathOver₁-≃-PathOver₂
    : {a₁ a₂ : A} {α : a₁ == a₂}
    → {c₁ : C a₁}{c₂ : C a₂}
    → PathOver₁ C α c₁ c₂ ≃ PathOver₂ C α c₁ c₂
  PathOver₁-≃-PathOver₂ {α = idp }{c₁}{c₂} =
    qinv-≃
      PathOver₁-to-PathOver₂
      ( PathOver₂-to-PathOver₁
        , PathOver₁~PathOver₂ , PathOver₂~PathOver₁)
    where
      PathOver₁~PathOver₂ : (p : PathOver₂ C idp c₁ c₂)
        → PathOver₁-to-PathOver₂ ( PathOver₂-to-PathOver₁ p) == p
```

$\mathrm{PathOver}_1\text{\textasciitilde}\mathrm{PathOver}_2\ \mathrm{idp}\ = \mathrm{idp}$

$\mathrm{PathOver}_2\text{\textasciitilde}\mathrm{PathOver}_1 : (p : \mathrm{PathOver}_1\ \ C\ \mathrm{idp}\ \ c_1\ c_2)$
$\quad \to \mathrm{PathOver}_2\text{-to-}\mathrm{PathOver}_1\ (\ \mathrm{PathOver}_1\text{-to-}\mathrm{PathOver}_2\ p)\ \ ==\ \ p$
$\mathrm{PathOver}_2\text{\textasciitilde}\mathrm{PathOver}_1\ \mathrm{idp}\ = \mathrm{idp}$

By default, we use the third definition because it is the same definition used in [3] in Section 2.3. The syntax sugar for pathovers is used in [1].

```
PathOver  = PathOver₃

infix 30 PathOver
syntax PathOver C α c1 c2 = c1 == c2 [ C ↓ α ]
```

Total spaces

Theorem

Let be $A : \mathcal{U}$, a path $\alpha : a_1 == a_2$ of two terms $a_1, a_2 : A$ and a type family $C : A \to \mathcal{U}$. If $c_2 : Ca_1$ and $c_2 : Ca_2$ then the type of the pathovers between c_1 and c_1 over the path α is equivalent to the sigma type of $(a_1, c_1) == (a_2, c_2)$ such that $\mathrm{ap}\ \pi_1 q == \alpha$, that is the following equivalence,

$$\sum_{q:(a_1,c_1)=(a_2,c_2)} (\mathrm{ap}\ \pi_1\ q\ = \alpha) \simeq PathOver\ \ C\ \alpha\ c_1\ c_2.$$

Proof

```
module _ {ℓᵢ ℓⱼ}{A : Type ℓᵢ}{C : A → Type ℓⱼ}{a₁ a₂ : A} where
```

We prove this equivalence by the quasi-inverse function $\Sigma - to- == [\downarrow]$. Therefore, we define its inverse, the function $== [\downarrow] - to - \Sigma$ and we show the respective homotopies, $\Sigma - to- == [\downarrow] \circ == [\downarrow] - to - \Sigma \; id$ and $== [\downarrow] - to - \Sigma \circ \Sigma - to- == [\downarrow] \; id$.

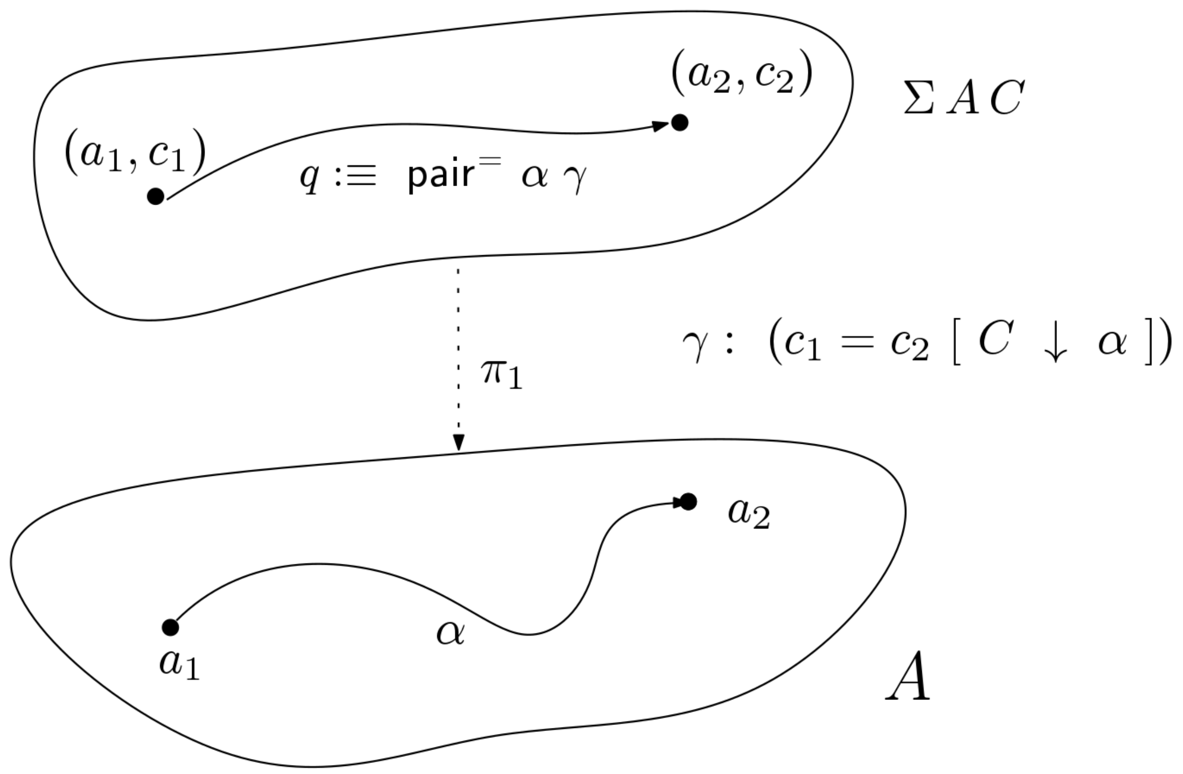

(a_1, c_1)
(a_2, c_2)
$q :\equiv \mathsf{pair}^= \ \alpha \ \gamma$
$\Sigma\, A\, C$
π_1
$\gamma : \ (c_1 = c_2 \ [\ C \ \downarrow \ \alpha \ ])$
a_1
α
a_2
A

Figure 3: Pathovers and paths in the total space.

- The function $\Sigma-to- == [\downarrow]$ maps a term of the sigma type in the equation above to the pathover $c_2 == c_2[C \downarrow \alpha]$. In its construction, we use Σ-induction followed by two path-inductions on each of the its sigma components. As result, we only have to provide a term of the identity type $c_2 == c_2$ where c_1 and c_2 are judgementally equal, which is idp.

```
-- Def.
Σ–to–==[↓]
  : {α : a₁ == a₂}{c₁ : C a₁}{c₂ : C a₂}
  → Σ ((a₁ , c₁) == (a₂ , c₂)) ( q → ( ap ₁ q) == α)
  → c₁ == c₂ [ C ↓ α ]
Σ–to–==[↓] ( idp , idp ) = idp
```

- The respective inverse function is $== [\downarrow] - to - \Sigma$, which maps terms of the pathover $c_2 == c_2[C \downarrow \alpha]$ to pairs in $\Sigma((a_1, c_1) == (a_2, c_2))(\lambda q \to (ap \pi_1 q) == \alpha)$. In its construction, we use path-induction on the path α in the base space follows by the induction on the pathover γ. As result, we define this function as a pair of reflexivity proofs.

```
-- Def.
==[↓]–to–Σ
  : {α : a₁ == a₂}{c₁ : C a₁}{c₂ : C a₂}
  → (γ : c₁ == c₂ [ C ↓ α ] )
  → Σ ((a₁ , c₁) == (a₂ , c₂)) ( q → ( ap  ₁ q) == α)
==[↓]–to–Σ { idp } idp = ( idp  , idp )
```

However, we do not get any benefit as far as we know of the latter definition against the former definition. Therefore, we have preferred the former which is simpler, elegant and exploits the pattern matching of Agda as well as in the following homotopies.

```
-- Homotopy: Σ-to-==[↓] ∘ ==[↓]-to-Σ ~ id
  private
    H₁
      : {α : a₁ == a₂}{c₁ : C a₁}{c₂ : C a₂}
      → (γ : c₁ == c₂ [ C ↓ α ] )
      → Σ–to–==[↓] {α = α} ( ==[↓]–to–Σ γ) == γ
    H₁ {α = idp } idp = idp

-- Homotopy: ==[↓]-to-Σ ∘ Σ-to-==[↓] ~ id
  private
    H₂ : {α : a₁ == a₂}{c₁ : C a₁}{c₂ : C a₂}
       → (pair : Σ ((a₁ , c₁) == (a₂ , c₂)) ( q → ( ap ₁ q)
       → ==[↓]–to–Σ ( Σ–to–==[↓] pair) == pair
    H₂ ( idp , idp ) = idp
```

Our remaining step now is to show the respective equivalence. To show that, we have used the function *qinv*$- \simeq$ that provides us a way to convert a quasi-inverse function into the equivalence between its domain and codomain. Since the function $\Sigma - to- == [\downarrow]$ is quasi-inverse by definition using $== [\downarrow] - to - \Sigma$, H_1 and H_2 hence the equivalence follows.

```
-- Equivalence
 private
  Σ-≃-==[↓]
   : {α : a₁ == a₂}{c₁ : C a₁}{c₂ : C a₂} →
   ( Σ ((a₁ , c₁) == (a₂ , c₂)) ( q → ( ap ₁ q) == α))
    ≃ (c₁ == c₂ [ C ↓ α ] )

  Σ-≃-==[↓] =
   qinv-≃
    Σ-to-==[↓]  -- the quasi-inverse
    ( ==[↓]-to-Σ -- its inverse
    , H₁         -- homotopy: Σ-to-==[↓] ∘ ==[↓]-to-Σ ~ id
    , H₂         -- homotopy: ==[↓]-to-Σ ∘ Σ-to-==[↓] ~ id
    )
```

In the remaining of this section, we prove some useful results about sigma types that allow us to give a shorter proof of the equivalence proved above.

Lemma 1

If $A\,,\ B\ :\ U$ and $C\ :\ A \to U$ and $e\ :\ B \simeq A$, then

$$\Sigma\, A\, C \simeq \Sigma\, B\, (C \circ e).$$

Proof. Our context:

```
module Lemma₁ {ℓi}{ℓj}
  {A : Type ℓi} {B : Type ℓi}
  (e : B ≃ A) {C : A → Type ℓj} where
```

We extract the functions and the homotopies from the equivalence $e : B \simeq A$ to use them later.

```
-- Def.
  private
    f : B → A
    f = lemap e

    ishaef : ishae f
    ishaef = ≃-ishae e

    f⁻¹ : A → B
    f⁻¹ = ishae.g ishaef

      : f ∘ f⁻¹ ∼ id
      = ishae. ishaef

      : f⁻¹ ∘ f ∼ id
      = ishae. ishaef
    : (b : B) → ap f ( b) == ( f b)
    = ishae. ishaef
```

Now, we proceed to define the outgoing functions from $\Sigma\, A\, C$ to $\Sigma\, B\, (C \circ e)$ and conversely.

```
-- Def.
ΣAC–to–ΣBCf : Σ A C → Σ B ( b → C ( f b))
ΣAC–to–ΣBCf (a , c) = f⁻¹ a , c′
  where
  c′ : C ( f ( f⁻¹ a))
  c′ = transport C (( a) ⁻¹ ) c

-- Def.
ΣBCf–to–ΣAC : Σ B ( b → C ( f b)) → Σ A C
ΣBCf–to–ΣAC (b , c′) = f b , c′
```

Evidence of the homotopies necessary to show the equivalence:

```
-- Homotopies
```

```
private
  H₁ : ΣAC−to−ΣBCf ∘ ΣBCf−to−ΣAC ∼ id
  H₁ (b , c′) = pair= (  b , patho )
    where
    c″ : C ( f ( f⁻¹ ( f b)))
    c″ = transport C ((  ( f b)) ⁻¹ ) c′

    -- patho : c” == c’ [ (C ∘ f) ↓ (β b) ]
    patho : transport ( x → C ( f x)) (  b) c″ == c′
    patho =
      begin
        transport ( x → C ( f x)) (  b) c″
          ==⟨ transport−family (  b) c″ ⟩
        transport C ( ap f (  b)) c″
          ==⟨ ap ( γ → transport C γ c″ ) (  b) ⟩
        transport C (  ( f b)) c″
      ==⟨ transport−comp−h ((  ( f  b)) ⁻¹ ) (   ( f  b)) c′
    transport  C ( ((   ( f  b)) ⁻¹ )  ·    ( f  b)) c′
      ==⟨ ap  (  γ → transport  C γ c′) ( ·−linv (   ( f  b)))
    transport  C idp  c′
      ==⟨⟩
    c′
  ∎

private
  H₂  : ΣBCf−to−ΣAC ∘ ΣAC−to−ΣBCf  ∼  id
  H₂  (a  ,  c) = pair=  (    a  ,  patho )
    where
    patho  : transport  C (    a) ( transport  C ((    a) ⁻¹ ) c)  =:
    patho  =
      begin
        transport  C (    a) ( transport  C ((    a)  ⁻¹ ) c)
          ==⟨ transport−comp−h (((    a)  ⁻¹ )) (    a) c  ⟩
        transport  C ( ((   a)  ⁻¹ )  ·  (   a) ) c
          ==⟨ ap  (  γ → transport  C γ c) ( ·−linv (   a))  ⟩
        transport  C idp  c
          ==⟨⟩
        c
      ∎
```

Finally, we now are able to prove the equivalence using the terms defined above.

```
  -- Equivalence
   lemma₁  :   Σ  A C  ≃  Σ  B (  b → C ( f  b))
   lemma₁  = qinv−≃
              ΣAC−to−ΣBCf  -- the quasi-inverse
              ( ΣBCf−to−ΣAC -- its inverse
              , H₁           -- ΣAC-to-ΣBCf ∘ ΣBCf-
              , H₂           -- ΣBCf-to-ΣAC ∘ ΣAC-t
              )
```

open Lemma_1 public

$\square$

Lemma 2:

If $A : U$ and $C : A \to U$ and $a : A$ then

$$\Sigma_{(w\,:\,\Sigma A C)} (\pi_1\, w =_A a) \simeq C\, a.$$

Proof:

```
module Lemma₂ {ℓ} {A : Type ℓ}{C : A → Type ℓ}(a : A) where

  ΣΣ–to–C : Σ (Σ A C) ( w → ₁ w == a) → C a
  ΣΣ–to–C ((a , c) , p) = transport C p c

  C–to–ΣΣ : C a → Σ (Σ A C) ( w → ₁ w == a)
  C–to–ΣΣ c = (a , c) , idp

private
  H₁ : ΣΣ–to–C ∘ C–to–ΣΣ ∼ id
  H₁ c = idp

  H₂ : C–to–ΣΣ ∘ ΣΣ–to–C ∼ id
  H₂ ((a′ , c) , p) = pair= ( paireq , patho )
    where
      c′ : transport C ( inv p) ( transport C p c) == c
      c′ = begin
          transport C ( inv p) ( transport C p c)
            ==⟨ transport–comp–h p (( inv p)) c ⟩
    transport C (p · ( inv p)) c
      ==⟨ ap ( γ → transport C γ c) ( ·−rinv p) ⟩
    transport C idp c
      ==⟨⟩
    c
  ∎

paireq : a , transport C p c == a′ , c
paireq = pair= ( inv p , c′ )

patho : transport ( w → 1 w == a) paireq idp == p
patho
  = begin
    transport ( w → 1 w == (( _ → a) w)) paireq idp
      ==⟨ transport−eq−fun 1 ( _ → a) paireq idp ⟩
    inv ( ap 1 paireq ) · idp · ap ( _ → a) paireq
      ==⟨ ap ( γ → inv ( ap 1 paireq ) · idp · γ)
      ( ap−const paireq ) ⟩
    inv ( ap 1 paireq ) · idp · idp
      ==⟨ ·−runit−infer ⟩
    inv ( ap 1 paireq ) · idp
      ==⟨ ·−runit−infer ⟩
    inv ( ap 1 paireq )
      ==⟨ ap ( p → inv p) ( ap− 1−pair= ( inv p) c′ ) ⟩
    inv ( inv p)
      ==⟨ involution ⟩
      p
    ∎

lemma2 : Σ ( Σ A C) ( w → 1 w == a) ≃ C a
```

```
lemma₂ = qinv-≃ ΣΣ-to-C ( C-to-ΣΣ , H₁ , H₂ )

open Lemma₂ public
```

Lemma 3:

If $A : U$ and for two type families $C,\ D : A \to U$. If we have $e : \Pi\,(a : A)\ C\ a \simeq D\ a$, then

$$\Sigma\, A\, C \ \simeq\ \Sigma\, A\, D.$$

Proof:

```
module Lemma₃ {ℓ} {A : Type ℓ}{C : A → Type ℓ}{D : A → Type ℓ}
  (e : (a : A) → C a ≃ D a) where

 private
  f : (a : A) → C a → D a
  f a = lemap (e a)

  f⁻¹ : (a : A) → D a → C a
  f⁻¹ a = remap (e a)

   : (a : A) → ( f a) ∘ ( f⁻¹ a) ∼ id
   a x = lrmap−inverse (e a)

   : (a : A) → ( f⁻¹ a) ∘ ( f a) ∼ id
   a x = rlmap−inverse (e a)
```

```
ΣAC−to−ΣAD : Σ  A C → Σ  A D
ΣAC−to−ΣAD (a , c) = (a , ( f  a) c)

ΣAD−to−ΣAC : Σ  A D → Σ  A C
ΣAD−to−ΣAC (a , d) = (a , ( f⁻¹  a) d)

H₁  : ΣAC−to−ΣAD ∘ ΣAD−to−ΣAC ∼ id
H₁  (a , d) = pair= ( idp  ,   a d)

H₂  : ΣAD−to−ΣAC ∘ ΣAC−to−ΣAD ∼ id
H₂  (a , c) = pair= ( idp ,   a c)

lemma₃  : Σ  A C  ≃ Σ  A D
lemma₃  = qinv−≃ ΣAC−to−ΣAD ( ΣAD−to−ΣAC , H₁  , H₂ )

open Lemma₃  public
```

Alternative proof:

Let us recall the equivalence.

$$\sum_{q\,:\,(a_1,c_1)=(a_2,c_2)} (\mathrm{ap}\ \pi_1\ q\ = \alpha) \simeq PathOver\ \ C\ \alpha\ c_1\ c_2.$$

Using the previous lemmas, the following is an alternative

proof of the theorem Σ− ≃ − == [↓].
Our context for this proof is:
Proof:

```
module _ {ℓ}
  {A : Type ℓ}
  {C : A → Type ℓ}
  {a₁ a₂ : A}
  (α : a₁ == a₂)
  {c₁ : C a₁}
  {c₂ : C a₂} where

-- Theorem.
  private

    Σ−≃−==[↓] :
      Σ ((a₁ , c₁) == (a₂ , c₂))
        ( q → ap ₁ q == α) ≃ PathOver C α c₁ c₂

    Σ−≃−==[↓] =
      begin≃
        Σ ((a₁ , c₁) == (a₂ , c₂)) ( q → ap ₁ q == α)
          ≃⟨ lemma₁ pair=Equiv ⟩
        Σ ( Σ (a₁ == a₂) ( β → transport C β c₁ == c₂))
          ( γ → ap ₁ ( pair= γ) == α)
          ≃⟨ lemma₃ ( ap− ₁−pair=Equiv α) ⟩
        Σ ( Σ (a₁ == a₂) ( β → transport C β c₁ == c₂))
          ( γ → ₁ γ == α)
          ≃⟨ lemma₂ α ⟩
  transport  C α c₁  ==  c₂
    ≃⟨⟩
  PathOver  C α c₁ c₂
≃█
```

Bibliography

[1] Guillaume Brunerie, Kuen-Bang Hou (Favonia), Evan Cavallo, Eric Finster, Jesper Cockx, Christian Sattler, Chris Jeris, Michael Shulman, and Others. Homotopy Type Theory in Agda, 2018. URL: `https://github.com/HoTT/HoTT-Agda`.

[2] Daniel R. Licata and Guillaume Brunerie. A cubical approach to synthetic homotopy theory. *Proceedings - Symposium on Logic in Computer Science*, 2015-July:92–103, 2015. `doi:10.1109/LICS.2015.19`.

[3] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013. URL:

http://saunders.phil.cmu.edu/book/
hott-online.pdf.