A Survey on Homotopy Type Theory with the Proof Assistant Agda

윤상진

석사과정 학위논문 심사

jiniy09@snu.ac.kr

January 8, 2024

Overview

Introduction

- HoTT는 유형론의 framework 를 가지는 새로운 형식 체계
- 그 자체로 호모토피적으로 해석 가능한 구조를 가지고 있음
- 유형론에 기반하여 증명보조기로의 이식이 매우 용이함
- 요약하자면, HoTT는 컴퓨터로의 이식이 용이한, 호모토피 이론을 공리적으로 탐구할 수 있는 새로운 형식체계이다.
- Voevodsky's essay

Basic terminologies and structural rules

- 4 kinds of judgements:

$$\Gamma \vdash A \text{ type} \qquad \Gamma \vdash A \equiv A' \qquad \Gamma \vdash a : A \qquad \Gamma \vdash a \equiv a' : A$$

- Γ is a **context**, a finite tuple of the forms $(x_i : A_i)_{i<n}$, where (x_i) are distinctive free variables, with an additional sequence of $(n-1)$-judgements which is that for each $i < n$, $(x_j : A_j)_{j<i} \vdash A_i$ type.

- Structural rules - Judgemental equality:

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma \vdash A \equiv A \text{ type}} \quad \frac{\Gamma \vdash A \equiv B \text{ type}}{\Gamma \vdash B \equiv A \text{ type}} \quad \frac{\Gamma \vdash A \equiv B \text{ type} \quad \Gamma \vdash B \equiv C \text{ type}}{\Gamma \vdash A \equiv C \text{ type}}$$

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

$$\frac{\Gamma \vdash A \equiv B \text{ type} \quad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, x : B, \Delta \vdash \mathcal{J}}$$

- Structural rules - Variable, weakening, and substitution rules:

$$\frac{\vdash \Gamma, x : A, \Delta \mathrm{ctxt}}{\Gamma, x : A, \Delta \vdash x : A}\text{V} \quad \frac{\Gamma \vdash A \text{ type} \quad \Gamma, \Delta \vdash \mathcal{J}}{\Gamma, x : A, \Delta \vdash \mathcal{J}}\text{W} \quad \frac{\Gamma \vdash a : A \quad \Gamma, x : A, \Delta \vdash \mathcal{J}}{\Gamma, \Delta[a/x] \vdash \mathcal{J}[a/x]}\text{S}$$

$$\frac{\Gamma \vdash a \equiv a' : A \quad \Gamma, x : A, \Delta \vdash B \text{ type}}{\Gamma, \Delta[a/x] \vdash B[a/x] \equiv B[a'/x] \text{ type}}\text{S-cong-type}$$

$$\frac{\Gamma \vdash a \equiv a' : A \quad \Gamma, x : A, \Delta \vdash b : B}{\Gamma, \Delta[a/x] \vdash b[a/x] \equiv b[a'/x] : B[a/x]}\text{S-cong-term}$$

Type constructors - Π

$$\frac{\Gamma, x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Pi_{x:A} B(x) \text{ type}}\Pi\text{-form} \qquad \frac{\Gamma, x : A \vdash B(x) \text{ type} \qquad \Gamma, x : A \vdash b(x) : B(x)}{\Gamma \vdash \lambda x.b(x) : \Pi_{x:A} B(x)}\Pi\text{-i}$$

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x) \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B(a)}\Pi\text{-elim}$$

$$\frac{\Gamma, x : A \vdash B(x) \text{ type} \qquad \Gamma, x : A \vdash b(x) : B(x) \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda x : A.b(x))(a) \equiv b(a) : B(a)}\Pi\text{-comp-}\beta$$

$$\frac{\Gamma \vdash f : \Pi_{x:A} B(x)}{\Gamma \vdash \lambda x.f(x) \equiv f : \Pi_{x:A} B(x)}\Pi\text{-comp-}\eta$$

For the special case when B is non-dependent over A, that is, if we have two judgements $\Gamma \vdash A$ type and $\Gamma \vdash B$ type, we write $A \to B$ for $\Pi_{x:A} B(x)$.
In Agda, a dependent product type $\Pi_{x:A}\mathcal{P}(x)$ is represented as $(x : A) \to \mathcal{P}\ x$.

Type constructors - Σ

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma,\ x : A \vdash B(x) \text{ type}}{\Gamma \vdash \Sigma_{x:A} B(x) \text{ type}}\ \Sigma\text{-form}$$

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma,\ x : A \vdash B(x) \text{ type}}{\Gamma,\ x : A,\ y : B(x) \vdash \text{pair}(x, y) : \Sigma_{x:A} B(x)}\ \Sigma\text{-intro}$$

$$\frac{\Gamma,\ z : \Sigma_{x:A} B(x) \vdash \mathcal{P}(z) \text{ type} \qquad \Gamma,\ x : A,\ y : B(x) \vdash f(x, y) : \mathcal{P}(\text{pair}(x, y))}{\Gamma,\ z : \Sigma_{x:A} B(x) \vdash \Sigma\text{elim}(f, z) : \mathcal{P}(z)}\ \Sigma\text{-eli}$$

$$\frac{\Gamma,\ z : \Sigma_{x:A} B(x) \vdash \mathcal{P}(z) \text{ type} \qquad \Gamma,\ x : A,\ y : B(x) \vdash f(x, y) : \mathcal{P}(\text{pair}(x, y))}{\Gamma,\ x : A,\ y : B(x) \vdash \Sigma\text{elim}(f, \text{pair}(x, y)) \equiv f(x, y) : \mathcal{P}(\text{pair}(x, y))}\ \Sigma\text{-co}$$

For the case where B is not dependent over A, we write $A \times B$ for $\Sigma_{x:A} B(x)$.

Type constructors - $+$

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma \vdash A + B \text{ type}} \; +\text{-form}$$

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma,\ x : A \vdash \mathsf{inl}(x) : A + B} \; +\text{-intro-l}$$

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma \vdash B \text{ type}}{\Gamma,\ y : B \vdash \mathsf{inr}(y) : A + B} \; +\text{-intro-r}$$

$$\frac{\Gamma,\ z : A + B \vdash \mathcal{P}(z) \text{ type} \qquad \Gamma, x : A \vdash l(x) : \mathcal{P}(\mathsf{inl}(x)) \qquad \Gamma, y : B \vdash r(y) : \mathcal{P}(\mathsf{inr}(y))}{\Gamma,\ z : A + B \vdash +\mathsf{elim}(l, r, z) : \mathcal{P}(z)} \; +\text{-elim}$$

$$\frac{\Gamma,\ z : A + B \vdash \mathcal{P}(z) \text{ type} \qquad \Gamma, x : A \vdash l(x) : \mathcal{P}(\mathsf{inl}(x)) \qquad \Gamma, y : B \vdash r(y) : \mathcal{P}(\mathsf{inr}(y))}{\Gamma,\ x : A \vdash +\mathsf{elim}(l, r, \mathsf{inl}(x)) \equiv l(x) : \mathcal{P}(\mathsf{inl}(x))} \; +\text{-com}$$

$$\frac{\Gamma,\ z : A + B \vdash \mathcal{P}(z) \text{ type} \qquad \Gamma, x : A \vdash l(x) : \mathcal{P}(\mathsf{inl}(x)) \qquad \Gamma, y : B \vdash r(y) : \mathcal{P}(\mathsf{inr}(y))}{\Gamma,\ y : B \vdash +\mathsf{elim}(l, r, \mathsf{inr}(y)) \equiv r(y) : \mathcal{P}(\mathsf{inr}(y))} \; +\text{-com}$$

Type constructors - $\mathbb{0}$

$$\frac{\vdash \Gamma \ \mathsf{ctxt}}{\Gamma \vdash \mathbb{0} \ \mathsf{type}} \ \mathbb{0}\text{-form}$$

$$\frac{\Gamma, \ x : \mathbb{0} \vdash \mathcal{P}(x) \ \mathsf{type}}{\Gamma, x : \mathbb{0} \vdash \mathbb{0}\mathsf{elim}(x) : \mathcal{P}(x)} \ \mathbb{0}\text{-elim}$$

We define the negation $\neg$ of types:

```
¬_ : type ℓ → type ℓ
¬ X = X → 𝟘
```

We call a type theory with the above empty type rules **inconsistent** if it is possible to construct a term in $\mathbb{0}$ from the empty context(i.e. it is a useless system). If not, we call the theory **consistent**.

Type constructors - $\mathbb{1}$

$$\frac{\vdash \Gamma\ \mathsf{ctxt}}{\Gamma \vdash \mathbb{1}\ \mathsf{type}}\ \mathbb{1}\text{-form}$$

$$\frac{\vdash \Gamma\ \mathsf{ctxt}}{\Gamma \vdash \star : \mathbb{1}}\ \mathbb{1}\text{-intro}$$

$$\frac{\Gamma,\ x : \mathbb{1} \vdash \mathcal{P}(x)\ \mathsf{type} \qquad \Gamma \vdash c : \mathcal{P}(\star)}{\Gamma, x : \mathbb{1} \vdash \mathbb{1}\mathsf{elim}(c, x) : \mathcal{P}(x)}\ \mathbb{1}\text{-elim}$$

$$\frac{\Gamma,\ x : \mathbb{1} \vdash \mathcal{P}(x)\ \mathsf{type} \qquad \Gamma \vdash c : \mathcal{P}(\star)}{\Gamma, x : \mathbb{1} \vdash \mathbb{1}\mathsf{elim}(c, \star) \equiv c : \mathcal{P}(x)}\ \mathbb{1}\text{-comp}$$

Type constructors - Inductive type $\mathbb{N}$

$$\frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash \mathbb{N} \text{ type}} \ \mathbb{N}\text{-form}$$

$$\frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash 0 : \mathbb{N}} \ \mathbb{N}\text{-intro-0}$$

$$\frac{\vdash \Gamma \text{ ctxt}}{\Gamma \vdash \mathsf{suc} : \mathbb{N} \to \mathbb{N}} \ \mathbb{N}\text{-intro-suc}$$

$$\frac{\Gamma,\ n : \mathbb{N} \vdash \mathcal{P}(n) \text{ type} \qquad \Gamma \vdash c_0 : \mathcal{P}(0) \qquad \Gamma,\ n : \mathbb{N} \vdash c_s : \mathcal{P}(n) \to \mathcal{P}(\mathsf{suc}(n))}{\Gamma, n : \mathbb{N} \vdash \mathbb{N}\mathsf{elim}(c_0, c_s, n) : \mathcal{P}(n)} \ \mathbb{N}\text{-elim}$$

$$\frac{\Gamma,\ n : \mathbb{N} \vdash \mathcal{P}(n) \text{ type} \qquad \Gamma \vdash c_0 : \mathcal{P}(0) \qquad \Gamma, n : \mathbb{N} \vdash c_s : \mathcal{P}(n) \to \mathcal{P}(\mathsf{suc}(n))}{\Gamma \vdash \mathbb{N}\mathsf{elim}(c_0, c_s, 0) \equiv c_0 : \mathcal{P}(0)} \ \mathbb{N}\text{-comp-0}$$

$$\frac{\Gamma,\ n : \mathbb{N} \vdash \mathcal{P}(n) \text{ type} \qquad \Gamma \vdash c_0 : \mathcal{P}(0) \qquad \Gamma, n : \mathbb{N} \vdash c_s : \mathcal{P}(n) \to \mathcal{P}(\mathsf{suc}(n))}{\Gamma, n : \mathbb{N} \vdash \mathbb{N}\mathsf{elim}(c_0, c_s, \mathsf{suc}(n)) \equiv c_s(n, \mathbb{N}\mathsf{elim}(c_0, c_s, n)) : \mathcal{P}(\mathsf{suc}(n))} \ \mathbb{N}\text{-comp-suc}$$

Type constructors - Identity types _=_

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma,\ x, y : A \vdash x =_A y \text{ type}} =\text{-form}$$

$$\frac{\Gamma \vdash A \text{ type}}{\Gamma,\ x : A \vdash \mathsf{refl}_A(x) : x =_A x} =\text{-intro}$$

$$\frac{\Gamma,\ x, y : A,\ p : x =_A y \vdash \mathcal{P}(x, y, p) \text{ type} \qquad \Gamma,\ x : A \vdash c(x) : \mathcal{P}(x, x, \mathsf{refl}_A(x))}{\Gamma,\ x, y : A,\ p : x =_A y \vdash = \mathsf{elim}(c, x, y, p) : \mathcal{P}(x, y, p)} =\text{-elim}$$

$$\frac{\Gamma,\ x, y : A,\ p : x =_A y \vdash \mathcal{P}(x, y, p) \text{ type} \qquad \Gamma,\ x : A \vdash c(x) : \mathcal{P}(x, x, \mathsf{refl}_A(x))}{\Gamma,\ x, y : A,\ p : x =_A y \vdash = \mathsf{elim}(c, x, x, \mathsf{refl}_A(x)) \equiv c(x) : \mathcal{P}(x, x, \mathsf{refl}_A(x))} =\text{-comp}$$

Groupoid structure of types

```
_•_ : {X : type ℓ} {x y z : X} → x = y → y = z → x = z

_⁻¹ : {A : type ℓ} {x y : A} → x = y → y = x
sym = _⁻¹

•-refl-l : {A : type ℓ} {x y : A}
         → (p : x = y) → refl x • p = p

•-refl-r : {A : type ℓ} {x y : A}
         → (p : x = y ) → p • refl y = p

•-sym-l : {A : type ℓ} {x y : A}
        → (p : x = y ) → p ⁻¹ • p = refl y

•-sym-r : {A : type ℓ} {x y : A}
        → (p : x = y ) → p • p ⁻¹ = refl x

•-assoc : {A : type ℓ} {x y z w : A}
        → (p : x = y) (q : y = z) (r : z = w)
        → (p • q) • r = p • (q • r)
```

Functions are groupoid functors

```
ap : {X : type ℓ} {Y : type ℓ'} {x x' : X}
   → (f : X → Y) → (x = x') → f x = f x'

ap-refl : {X Y : type ℓ} {x : X}
        → (f : X → Y) → ap f (refl x) = refl (f x)

ap-∙ : {X : type ℓ} {Y : type ℓ'} {x y z : X}
     → (f : X → Y) (p : x = y) (q : y = z)
     → ap f (p ∙ q) = ap f p ∙ ap f q

ap-sym : {X : type ℓ} {Y : type ℓ'} {x y : X}
       → (f : X → Y) (p : x = y)
       → ap f (p ⁻¹) = (ap f p) ⁻¹

ap-id : {X : type ℓ} {x y : X}
      → (p : x = y) → ap id p = p

ap-∘ : {X : type ℓ} {Y : type ℓ'} {Z : type ℓ''} {x y : X}
     → (g : Y → Z) (f : X → Y) (p : x = y)
     → ap (g ∘ f) p = (ap g (ap f p))
```

The action of paths

```
tr : {A : type ℓ} {x y : A}
   → (𝓟 : A → type ℓ') → x = y → 𝓟 x → 𝓟 y

tr-• : {X : type ℓ} {x y z : X}
     → (𝓟 : X → type ℓ') (p : x = y) (q : y = z)
     → tr 𝓟 (p • q) = tr 𝓟 q ∘ tr 𝓟 p

tr-∘ : {X : type ℓ} {Y : type ℓ'} {x x' : X}
     → (𝓟 : Y → type ℓ'') (f : X → Y) (p : x = x')
     → tr (𝓟 ∘ f) p = tr 𝓟 (ap f p)

tr-const : {X : type ℓ} {Y : type ℓ'} {x x' : X}
         → (p : x = x') (y : Y)
         → tr (λ - → Y) p y = y

pathover : {A : type ℓ} (𝓟 : A → type ℓ') {a b : A}
       → (x : 𝓟 a) (y : 𝓟 b) (p : a = b) → type ℓ'
pathover 𝓟 x y p = tr 𝓟 p x = y
syntax pathover 𝓟 x y p =   x =↑ y [ p ]over 𝓟
```

Some applications of transport

```
tr-fibmap : {X : type ℓ} {x y : X}
          → (𝒫 : X → 𝒰) (𝒬 : X → 𝒰) (p : x ≡ y) (f : 𝒫 x → 𝒬 x)
          → tr (λ - → 𝒫 - → 𝒬 -) p f ≡ (tr 𝒬 p) ∘ f ∘ (tr 𝒫 (p ⁻¹))

tr-path-lfix : {X : type ℓ} {x y : X}
             → (s : X) (p : x ≡ y)
             → (r : s ≡ x) → tr (λ - → s ≡ -) p r ≡ r ∙ p

tr-path-rfix : {X : type ℓ} {x y : X}
             → (t : X) (p : x ≡ y)
             → (r : x ≡ t) → tr (λ - → - ≡ t) p r ≡ (p ⁻¹) ∙ r

tr-path-btwmaps : {X : type ℓ} {Y : type ℓ'} {x y : X}
                → (f g : X → Y) (p : x ≡ y) (γ : f x ≡ g x)
                → tr (λ - → f - ≡ g -) p γ ≡ (ap f p) ⁻¹ ∙ γ ∙ (ap g p)

apd : {X : type ℓ} {𝒫 : X → type ℓ'} {x y : X}
    → (f : Π 𝒫) (p : x ≡ y) → tr 𝒫 p (f x) ≡ f y

apd-const : {X : type ℓ} {Y : type ℓ'} {x y : X}
          → (f : X → Y) (p : x ≡ y)
          → apd f p ≡ (tr-const p (f x)) ∙ (ap f p)
```

Homotopy between functions

```
_~_ : {X : type ℓ} {𝒫 : X → type ℓ'}
    → Π 𝒫 → Π 𝒫 → type (ℓ ⊔ ℓ')
_~_ {X = X} f g = (x : X) → f x = g x

ht-refl : {A : type ℓ} {𝒫 : A → type ℓ'}
        → (f : Π 𝒫) → f ∼ f

_ʰⁱ :{A : type ℓ} {𝒫 : A → type ℓ'} {f g : Π 𝒫}
    → f ∼ g → g ∼ f
ht-sym = _ʰⁱ

_●ₕ_ : {A : type ℓ} {𝒫 : A → type ℓ'} {f g h : Π 𝒫}
     → f ∼ g → g ∼ h → f ∼ h
ht● = _●ₕ_
```

Natural square of a homotopy

$$\begin{tikzcd}
f(x) \arrow[r, "ap_f\, p"] \arrow[d, "H(x)"'] & f(y) \arrow[d, "H(y)"] \\
g(x) \arrow[r, "ap_g\, p"'] \arrow[ur, phantom, "\Nearrow"] & g(y)
\end{tikzcd}$$

```
ht-nat : {A : type ℓ} {B : type ℓ'} {f g : A → B} {x y : A}
       → (H : f ∼ g) (p : x = y) → (H x) ● (ap g p) = (ap f p) ● (H y)

ht-nat-u : {A : type ℓ} {B : type ℓ'} {f g : A → B} {x y : A}
       → (H : f ∼ g) (p : x = y) → (H x) ● (ap g p) ● (H y) ⁻¹ = ap f p

ht-nat-d : {A : type ℓ} {B : type ℓ'} {f g : A → B} {x y : A}
       → (H : f ∼ g) (p : x = y) → (H x) ⁻¹ • (ap f p) • (H y) = ap g p
```

Whiskering

```
_∘l_ : {A : type ℓ} {B : type ℓ'} {C : type ℓ''} {f g : A → B}
  → (h : B → C) → f ∼ g → h ∘ f ∼ h ∘ g

_∘r_ : {A : type ℓ} {B : type ℓ'} {C : type ℓ''} {g h : B → C}
  → g ∼ h → (f : A → B) → g ∘ f ∼ h ∘ f
```

Section and retraction

```
has-sec : {A : type ℓ} {B : type ℓ'} (f : A → B) → type (ℓ ⊔ ℓ')
has-sec f = Σ (λ g → f ∘ g ∼ id)

has-ret : {A : type ℓ} {B : type ℓ'} (f : A → B) → type (ℓ ⊔ ℓ')
has-ret f = Σ (λ g → g ∘ f ∼ id)


record _◁_  (A : type ℓ) (B : type ℓ') : type (ℓ ⊔ ℓ')  where
  constructor  ◁pf
  field
    ret : B → A
    retpf : has-sec (ret)
```

Invertible map

```
record  ivtbl  {A : type ℓ} {B : type ℓ'} (f : A → B) : type (ℓ ⊔ ℓ') where
  constructor  Ivtbl
  field
    inv : B → A
    inv-s : f ∘ inv ∼ id
    inv-r : inv ∘ f ∼ id


record  _≅_  (A : type ℓ) (B : type ℓ') : type (ℓ ⊔ ℓ') where
  constructor ≅pf
  field
    ivt : A → B
    ivtpf : ivtbl ivt
```

Equivalence

```
record  equiv  {A : type ℓ} {B : type ℓ'} (f : A → B) : type (ℓ ⊔ ℓ') where
  constructor Equiv
  field
    sec : B → A
    sec-h : f ∘ sec ∼ id
    ret : B → A
    ret-h : ret ∘ f ∼ id

record  _≃_  (A : type ℓ) (B : type ℓ') : type (ℓ ⊔ ℓ') where
  constructor ≃pf
  field
    eqv : A → B
    eqvpf : equiv eqv
```

Invertible $\iff$ Equivalence

```
ivtbl-equiv :  {A : type ℓ} {B : type ℓ'} {f : A → B}
            → (I : ivtbl f) → equiv f

equiv-ivtbl : {A : type ℓ} {B : type ℓ'} {f : A → B}
            → (E : equiv f) → ivtbl f


≅-≃ : {A : type ℓ} {B : type ℓ'} → A ≅ B → A ≃ B

≃-≅ : {A : type ℓ} {B : type ℓ'} → A ≃ B → A ≅ B
```

Contractible type

```
contr : type ℓ → type ℓ
contr A = Σ c : A , (Π x : A , (c = x))
```

```
contr-closed-above : {A : type ℓ}
     → (contr A) → ((x y : A) → contr (x = y))
```

h-level

```
_has-hlv_ : type ℓ → ℕ → type ℓ
A has-hlv 0 = contr A
A has-hlv (suc n) = (x y : A) → (x ≡ y) has-hlv n

prop : type ℓ → type ℓ
prop A = A has-hlv 1

set : type ℓ → type ℓ
set A = A has-hlv 2
```

Another description of set

```
set' : type ℓ → type ℓ
set' A = (x y : A) → (p q : x = y) → p = q

set→set' : {A : type ℓ} → set A → set' A

set'→set : {A : type ℓ} → set' A → set A
```

Properties of h-level

```
hlv-closed-above : (n : ℕ) {A : type ℓ}
                 → (A has-hlv n) → (A has-hlv (suc n))

hlv-closed-ret : (n : ℕ) {A : type ℓ} {B : type ℓ'}
               → A ◁ B → (B has-hlv n) → (A has-hlv n)
```

The type of integers as an inductive type

```
data ℤ : 𝒰 where
 pos : ℕ → ℤ
 negsuc : ℕ → ℤ

succℤ : ℤ → ℤ
succℤ (pos x) = pos (suc x)
succℤ (negsuc 0) = pos 0
succℤ (negsuc (suc x)) = negsuc x

predℤ : ℤ → ℤ
predℤ (pos 0) = negsuc 0
predℤ (pos (suc x)) = pos x
predℤ (negsuc x) = negsuc (suc x)

succ-≅ : ℤ ≅ ℤ

ℤ≅ℕ+ℕ : ℤ ≅ ℕ + ℕ
```

ℤ is a set

```
ℕ-is-set : set ℕ

set-closed-+ : {A : type ℓ} {B : type ℓ'}
             → set A → set B → set (A + B)

ℤ-is-set : set ℤ
```

Function extensionality

```
happ : {X : type ℓ} {𝒫 : X → type ℓ'} {f g : Π 𝒫}
     → f = g → f ∼ g
happ p x = ap (λ - → - x) p
fext' = happ

postulate
  FEXT : {X : type ℓ} {𝒫 : X → type ℓ'} {f g : Π 𝒫}
       → equiv (happ {f = f} {g = g})
```

Function extensionality

```
FEXT-ivtbl : {X : type 𝓁} {𝓟 : X → type 𝓁'} {f g : Π 𝓟}
           → ivtbl (happ {f = f} {g = g})
FEXT-ivtbl = equiv-ivtbl (FEXT)

fext : {X : type 𝓁} {𝓟 : X → type 𝓁'} {f g : Π 𝓟}
     →  f ∼ g → f  =  g
fext = ivtbl.inv (FEXT-ivtbl)

fext-s : {X : type 𝓁} {𝓟 : X → type 𝓁'} {f g : Π 𝓟}
       → (happ {f = f} {g = g}) ∘ (fext {f = f} {g = g}) ∼ id
fext-s = ivtbl.inv-s (FEXT-ivtbl)

fext-r : {X : type 𝓁} {𝓟 : X → type 𝓁'} {f g : Π 𝓟}
       → (fext {f = f} {g = g}) ∘ (happ {f = f} {g = g}) ∼ id
fext-r {f} {g} = ivtbl.inv-r (FEXT-ivtbl)
```

Univalence axiom

```
tr-id : {A B : type ℓ} → A ≡ B → (A → B)
tr-id = tr id

≡-≅ : {A B : type ℓ} → A ≡ B → A ≅ B
≡-≅ p = (≅pf (tr-id p)
             (Ivtbl (tr-id (p ⁻¹)) H K))

≡-≃ : {A B : type ℓ} → A ≡ B → A ≃ B
≡-≃ = ≅-≃ ∘ ≡-≅
uaⁱ = ≡-≃

postulate
  UA : {A B : 𝒰} → equiv (≡-≃ {A = A} {B = B})
```

Univalence axiom

```
UA-ivtbl : {A B : 𝒰} → ivtbl (=-≃ {A = A} {B = B})
UA-ivtbl = equiv-ivtbl (UA)

ua : {A B : 𝒰} → A ≃ B → A  = B
ua = ivtbl.inv (UA-ivtbl)

ua-s : {A B : 𝒰} → (ua' {A = A} {B = B}) ∘ (ua {A = A} {B = B}) ∼ id
ua-s = ivtbl.inv-s (UA-ivtbl)

ua-r : {A B : 𝒰} → (ua {A = A} {B = B}) ∘ (ua' {A = A} {B = B}) ∼ id
ua-r = ivtbl.inv-r (UA-ivtbl)
```

Univalence axiom

```
tr-ua : {X Y : 𝓤} (E : X ≃ Y) → tr-id (ua E) ＝ (_≃_.eqv E)

≃sym : {X : type ℓ} {Y : type ℓ'} → X ≃ Y → Y ≃ X

＝-≃-sym : {X Y : 𝓤} → (p : X ＝ Y) → ＝-≃ (sym p) ＝ ≃sym (＝-≃ p)

ua-sym : {X Y : 𝓤} → (E : X ≃ Y) → ua (≃sym E) ＝ sym (ua E)
```

HIT - Homotopy pushout

$$
\begin{tikzcd}
X \arrow[r, "f_2"] \arrow[d, "f_1"'] & A_2 \arrow[d, "in_2"'] \arrow[ddr, bend left, "\varphi_2"] & \\
A_1 \arrow[r, "in_1"] \arrow[ur, phantom, "\nearrow glue"] \arrow[drr, bend right, "\varphi_1"'] & P \arrow[dr, dashed, "\varphi"] \arrow[r, phantom, "\circlearrowleft"] \arrow[d, phantom, "\circlearrowleft"] & {} \\
& {} & Y
\end{tikzcd}
$$

HIT - Homotopy pushout

```
postulate
  pushout : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂) → type (ℓ ⊔ ℓ' ⊔ ℓ'')

postulate
  in₁ : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''} {f₁ : X → A₁} {f₂ : X → A₂}
      → A₁ → pushout f₁ f₂

  in₂ : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''} {f₁ : X → A₁} {f₂ : X → A₂}
      → A₂ → pushout f₁ f₂

  glue : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''} {f₁ : X → A₁} {f₂ : X → A₂}
      → (x : X) → in₁ {f₁ = f₁} {f₂ = f₂} (f₁ x) ＝ in₂ (f₂ x)

postulate
  pushout-elim : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (𝓟 : pushout f₁ f₂ → type ℓ''')
          → (φ₁ : Π (𝓟 ∘ in₁)) → (φ₂ : Π (𝓟 ∘ in₂))
          → (I : (x : X) → (φ₁ (f₁ x)) ＝↑ φ₂ (f₂ x) [ glue x ]over 𝓟 )
          → Π 𝓟
```

HIT - Homotopy pushout

```
postulate
  pushout-comp-1 : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (𝒫 : pushout f₁ f₂ → type ℓ''')
          → (φ₁ : Π (𝒫 ∘ in₁)) → (φ₂ : Π (𝒫 ∘ in₂))
          → (I : (x : X) → (φ₁ (f₁ x)) ≡↑ φ₂ (f₂ x) [ glue x ]over 𝒫 )
          → (a1 : A₁)
          → (pushout-elim f₁ f₂ 𝒫 φ₁ φ₂ I) (in₁ a1) ≡ (φ₁ a1)

  pushout-comp-2 : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (𝒫 : pushout f₁ f₂ → type ℓ''')
          → (φ₁ : Π (𝒫 ∘ in₁)) → (φ₂ : Π (𝒫 ∘ in₂))
          → (I : (x : X) → (φ₁ (f₁ x)) ≡↑ φ₂ (f₂ x) [ glue x ]over 𝒫 )
          → (a2 : A₂)
          → (pushout-elim f₁ f₂ 𝒫 φ₁ φ₂ I) (in₂ a2) ≡ (φ₂ a2)

{-# REWRITE pushout-comp-1 #-}
{-# REWRITE pushout-comp-2 #-}
```

HIT - Homotopy pushout

```
postulate
  pushout-comp-glue : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (𝒫 : pushout f₁ f₂ → type ℓ''')
          → (φ₁ : Π (𝒫 ∘ in₁)) → (φ₂ : Π (𝒫 ∘ in₂))
          → (I : (x : X) → (φ₁ (f₁ x)) ≡↑ φ₂ (f₂ x) [ glue x ]over 𝒫 )
          → (x : X) → apd (pushout-elim f₁ f₂ 𝒫 φ₁ φ₂ I) (glue x) ≡ I x
```

HIT - Homotopy pushout

```
pushout-rec : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (Y : type ℓ''')
          → (φ₁ : A₁ → Y) → (φ₂ : A₂ → Y)
          → (I : (x : X) → (φ₁ (f₁ x)) ＝ φ₂ (f₂ x) )
          → (pushout f₁ f₂) → Y

pushout-rec-comp-1 : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (Y : type ℓ''')
          → (φ₁ : A₁ → Y) → (φ₂ : A₂ → Y)
          → (I : (x : X) → (φ₁ (f₁ x)) ＝ φ₂ (f₂ x) )
          → (a1 : A₁) → (pushout-rec f₁ f₂ Y φ₁ φ₂ I) (in₁ a1) ＝ φ₁ a1

pushout-rec-comp-2 : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (Y : type ℓ''')
          → (φ₁ : A₁ → Y) → (φ₂ : A₂ → Y)
          → (I : (x : X) → (φ₁ (f₁ x)) ＝ φ₂ (f₂ x) )
          → (a2 : A₂) → (pushout-rec f₁ f₂ Y φ₁ φ₂ I) (in₂ a2) ＝ φ₂ a2
```

HIT - Homotopy pushout

```
pushout-rec-comp-glue : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''}
          → (f₁ : X → A₁) (f₂ : X → A₂)
          → (Y : type ℓ''')
          → (φ₁ : A₁ → Y) → (φ₂ : A₂ → Y)
          → (I : (x : X) → (φ₁ (f₁ x)) = φ₂ (f₂ x) )
          → (x : X) → ap (pushout-rec f₁ f₂ Y φ₁ φ₂ I) (glue x) = I x
```

HIT - Homotopy pushout

```
pushout-∃ : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''} {Y : type ℓ'''}
          → (f₁ : X → A₁) (f₂ : X → A₂) (φ₁ : A₁ → Y) (φ₂ : A₂ → Y)
          → square f₁ f₂ φ₁ φ₂
          → (pushout f₁ f₂) → Y
pushout-∃ {Y = Y} f₁ f₂ φ₁ φ₂ H = pushout-rec f₁ f₂ Y φ₁ φ₂ H

pushout-! : {X : type ℓ} {A₁ : type ℓ'} {A₂ : type ℓ''} {Y : type ℓ'''}
          → (f₁ : X → A₁) (f₂ : X → A₂) (φ₁ : A₁ → Y) (φ₂ : A₂ → Y)
          → (H : square f₁ f₂ φ₁ φ₂)
          → (α : (pushout f₁ f₂) → Y)
          → (K₁ : φ₁ ∼ α ∘ in₁) → (K₂ : α ∘ in₂ ∼ φ₂)
          → (K₁ ∘ᵣ f₁) •ₕ (α ∘ₗ (glue {f₁ = f₁} {f₂ = f₂})) •ₕ (K₂ ∘ᵣ f₂) ∼ H
          → α ∼ (pushout-∃ f₁ f₂ φ₁ φ₂ H)
```

HIT - Suspension, spheres, and the circle

```
Σ : (X : type ℓ) → type ℓ
Σ X = pushout {X = X} const⋆ const⋆

S^_ : ℕ → 𝓤
S^ 0 = 2
S^ suc n = Σ (S^ n)

north south : S^ 1
north = in₁ ⋆
south = in₂ ⋆

west east : north = south
west = glue ₁
east = glue ₂
```

```
postulate
  S¹ : 𝒰
  base : S¹
  loop : base ≡ base

postulate
  S¹elim : (𝒫 : S¹ → type ℓ) (x : 𝒫 base) (ℓ : x ≡↑ x  [ loop ]over 𝒫 )
        → (z : S¹) → 𝒫 z

postulate
  S¹comp-base :  (𝒫 : S¹ → type ℓ) (x : 𝒫 base) (ℓ : x ≡↑ x  [ loop ]over 𝒫)
              →  ((S¹elim 𝒫 x ℓ) base)  ≡ x
{-#  REWRITE S¹comp-base  #-}

postulate
  S¹comp-loop :  (𝒫 : S¹ → type ℓ) (x : 𝒫 base) (ℓ : x ≡↑ x [ loop ]over 𝒫)
              → apd (S¹elim 𝒫 x ℓ) loop  ≡ ℓ
```

HIT - Suspension, spheres, and the circle

```
S¹rec : (A : type ℓ) (a : A) (p : a = a)
      → S¹ → A

S¹rec-comp-base : (A : type ℓ) (a : A) (p : a = a)
          → ((S¹rec A a p) base) = a

S¹rec-comp-loop : (A : type ℓ) (a : A) (p : a = a)
          → ap (S¹rec A a p) loop = p

S¹rec-loop-sym :  (A : type ℓ) (a : A) (p : a = a)
      → ap (S¹rec A a p) (loop ⁻¹) = p ⁻¹


S^1≅S¹ : S^ 1 ≅ S¹
```

The Fundamental Group of The Circle

```
Ω¹S¹ = base ≡ base

loop^_ : ℤ → Ω¹S¹
loop^ pos 0 = refl base
loop^ pos (suc n) = (loop^ (pos n)) • loop
loop^ negsuc 0 = loop ⁻¹
loop^ negsuc (suc n) = (loop^ (negsuc n)) • loop ⁻¹
```

```
cover : S¹ → 𝒰
cover = S¹rec 𝒰 ℤ (ua succ-≃)

loopact=succ : tr cover loop = succℤ

loop⁻¹act=pred : tr cover (loop ⁻¹) = predℤ
```

The Fundamental Group of The Circle

```
encode : (z : S¹) → (base = z → cover z)
encode z p = tr cover p (pos 0)

decode : (z : S¹) → (cover z → base = z)
decode = S¹elim
          𝒫
          loop^_
          loop^toloop^-over-loop

S¹-fiberwise-eqv : (z : S¹) → (base = z) ≃ (cover z)

Ω¹S¹≃ℤ : Ω¹S¹ ≃ ℤ
Ω¹S¹≃ℤ = S¹-fiberwise-eqv base
```

Conclusion

Identity types can be interpreted as path spaces and related inference rules are strong enough to construct various tools for the homotopy theory. By the univalence axiom and higher inductive types, we could introduce objects with interesting higher structures and give them fibrations to analyze them.
Recall the fact that the slight modification of the univalence axiom (namely, postulate `equiv` $(=\text{-}\cong)$ instead of `equiv` $(=\text{-}\simeq)$) makes our system inconsistent. By finding a model of HoTT in 'classical mathematics' we can bring the authority and belief in the consistency of mathematics to our theory.

The End