Playing With Homotopy Type Theory in Coq

Alexander Berenbeim

Today

Categories (Jumping to the `working` definition)

A category C consists of

- a class of objects denoted $|\mathsf{C}|$
- a class of arrows (morphisms) denoted by C
- two maps $s, t : \mathsf{C} \to |\mathsf{C}|$, a map $1 : |\mathsf{C}| \to \mathsf{C}$ and a composition relation $; \subseteq \mathsf{C} \times \mathsf{C} \times \mathsf{C}$

such that for all $X, Y, Z \in |\mathsf{C}|$

1. $\mathsf{C}(X, Y) := \{f \in \mathsf{C} \mid s(f) = X \wedge t(f) = Y\}$
2. (identity) $1(X) \equiv 1_X \in \mathsf{C}$
3. when $\mathsf{C}(X, Y)$ and $\mathsf{C}(Y, Z)$ are inhabited, then $(-);(-) : \mathsf{C}(X, Y) \times \mathsf{C}(Y, Z) \to \mathsf{C}(X, Z)$ defined by $(f, g) \mapsto f; g$,an associative operation such that the *identity* arrows are left and right identities.

Example: Pred

Not only are sets categories, but we can treat *predicates* on sets as a category as follows:

1. objects are pairs (I, X) such that $X \subseteq I$. We say that "X is a predicate of Y" and write $X(i)$ for $i \in X$. This choice of notation is intended to emphasize that $i \in I$ may be chosen as a *free variable*
2. morphisms $(I, X) \to (J, Y)$ are functions $u \in \mathrm{Sets}(I, J)$ such that for all $i \in I$, $X(i)$ implies $Y(u(i))$

Example : Rel

Just as we may turn predicates on sets into a category, we may also turn relations on sets into a category. We present the category of binary relations Rel as follows:

1. objects are pairs (I, R) where $I \in |\mathrm{Sets}|$ and $RI \times I$
2. morphisms $(I, R) \to (J, S)$ are set functions $u \in \mathrm{Sets}(I, J)$ such that for all $i, j \in I$, $R(i, j)$ implies that $S(u(i), u(j))$

Functors (Jumping to the `working` definition)

A functor $\mathcal{F} : \mathsf{C} \to \mathsf{D}$ consists of mappings $|\mathsf{C}| \to |\mathsf{D}|$ and $\mathcal{F}_{X,Y} : \mathsf{C}(X, Y) \to \mathsf{D}(\mathcal{F}(X), \mathcal{F}(Y))$ for all objects $X, Y \in |\mathsf{C}|$ such that

1. $\mathcal{F}(1_X) = 1_{\mathcal{F}(X)}$
2. $\mathcal{F}(f; g) = \mathcal{F}(f); \mathcal{F}(g)$ for all composable arrows $f, g \in \mathsf{C}$.

A Very Important Example

Given a functor $\mathcal{F} : \mathsf{D} \to \mathsf{C}$ we can define for any $X \in \mathsf{C}$ the comma category $X/\mathcal{F}$ whose arrows are $f \in \mathsf{C}(X, \mathcal{F}(Y))$, often denoted as (f, B), and arrows are $g \in \mathsf{D}(Y, Z)$ such that $f ; \mathsf{F}(g) =: f'$, where $f' \in \mathsf{C}(X, \mathcal{F}(Z))$.

Another Very Important Example: The Power Set

$\mathcal{P}$: Sets $\to$ Sets maps each set X to the set of its subsets $\{Y \mid Y \subset X\}$, and each $f : X \to Z$ is sent to $\mathcal{P}(f) : \mathcal{P}(X) \to \mathcal{P}(Z)$ such that $\mathcal{P}(f)(Y) = f(Y) = \{f(x) \mid x \in Y\}$.

Another Very Important example : Cartesian products

- Let's work in Sets for now
- For any fixed $X \in |\text{Sets}|$ we may define a functor $X \times - : \text{Sets} \to \text{Sets}$ which maps each $Y \in |\text{Sets}|$ to $X \times Y$ and each $f \in \text{Sets}(Y, Z)$ for any such Z to $(1_X \times f) : X \times Y \to X \times Z$ defined by $(1_X \times f)(a, b) = (a, f(b))$

Another Very Important example : Internal Hom functors

In any locally small category C, and for any object $X \in |C$, we can define

$$\mathrm{C}(X, -) : \mathrm{C} \to \mathrm{Sets}$$

by sending every $Y \in |\mathrm{C}|$ to the set of arrows $\mathrm{C}(X, Y)$ and every $f : Y \to Z$ is sent to $\mathrm{C}(X, f) : \mathrm{C}(X, Y) \to \mathrm{C}(X, Z)$ by pre-compostion, i.e. this is defined as $\mathrm{C}(X, f) := g; f$

Functor Properties

- A functor $\mathcal{F} : \mathsf{C} \to \mathsf{D}$ is full when for every $X, Y \in |\mathsf{C}|$, the mapping on arrows $\mathcal{X}, \mathcal{Y} : \mathsf{C}(\mathcal{X}, \mathcal{Y}) \to \mathsf{D}(\mathcal{F}(\mathcal{X}), \mathcal{F}(\mathcal{Y}))$ is surjective
- $\mathcal{F}$ is faithful if each $\mathcal{F}_{X,Y}$ is injective
- $\mathsf{C} \subseteq \mathsf{D}$, ie. C is a subcategory of D, if $|\mathsf{C}| \subseteq |\mathsf{D}|$ and $\mathsf{C}(X, Y) \subset \mathsf{D}(X, Y)$ for all $X, Y \in |\mathsf{C}|$, and composition in C is a restriction of composition in D
- A subcategory $\mathsf{C} \subseteq \mathsf{D}$ is broad when $|\mathsf{C}| = |\mathsf{D}|$.

- Given a functor $p : \mathrm{E} \to \mathrm{B}$, we can define a *new* category

with respect to every object in the image of p. Let $I \in |\mathrm{B}|$, and define $\mathrm{E}_I := p^{-1}(I)$ such that

1. objects are $X \in |\mathrm{E}|$ such that $p(X) = I$
2. morphisms are $f \in \mathrm{E}(X, Y)$ such that $p(f) = 1_I \in \mathrm{B}$

- E_I is the fibre category over I
- We say that $X \in |\mathrm{E}_I|$ is above I and similarly $f \in \mathrm{E}$ such that $p(f) = u$ is said to be above u.

Functors: Fibrations

- Given a functor $p : \mathsf{E} \to \mathsf{B}$, we say $f \in \mathsf{E}$ is Cartesian over and $u \in \mathsf{B}(I, J)$ if $p(f) = u$ and for every $g \in \mathsf{E}(Z, Y)$ such that $p(g) = u \circ w$ for some $w \in \mathsf{B}(p(Z), I)$ there is a uniquely determined $h \in \mathsf{E}(Z, X)$ *above* w with $f \circ h = g$
- $f \in \mathsf{E}(X, Y)$ is a Cartesian if it is Cartesian over its underlying map $p(f)$.
- p is a fibration if for every $Y \in |\mathsf{E}|$ and $u \in \mathsf{B}(I, p(Y))$ there is a cartesian morphism $f \in \mathsf{E}(X, Y)$ over u
- Practically understood, fibrations capture *indexing* and *substitution*

Fibrations Example

Let $I \in |\mathrm{Sets}|$.

- The fibre category Pred_I is the subcategory of predicates on I identified with the poset category $(\mathtt{P}(P)(I), \subseteq)$ ordered by inclusion
- Given any $u \in \mathrm{Sets}(I, J)$ we can define a substitution functor $u^* : \mathtt{P}(P)(J) \to \mathtt{P}(P)(I)$ via

$$(Y \subseteq J) \mapsto (\{i \mid u(i) \in Y\} \subseteq I)$$

- If $u = \pi : I \times J \to I$, then π^* is called weakening as it is given by $X \mapsto \{(i,j) \mid i \in X \wedge j \in J\}$ by adding a dummy variable $j \in J$ to the predicate X
- If $u = \Delta : I \to I \times I$, then Δ^* is called contraction as it is given by $\mathtt{P}(P)(I \times I) \ni Y \mapsto \{i \in I \mid (i,i) \in Y\}$, and thus replaces two variables of I with a single variable.

- A natural transformation is a map α between functors

$\mathcal{F}, \mathcal{G} : \mathsf{C} \to \mathsf{D}$ which consists of the following

1. for each $X \in \mathsf{C}$, there is a component $\alpha(X) \equiv \alpha_X \in \mathsf{D}(\mathcal{F}(X), \mathcal{G}(X))$
2. for every $f \in \mathsf{C}(X, Y)$, $\alpha_X ; \mathcal{G}(f) = \mathcal{F}(f ; \alpha_X$

- We denote a natural transformation by $\alpha : \mathcal{F} \Rightarrow \mathcal{G}$

Terminal Objects

- In a category C, we say 0 is initial if for all objects $X \in |C|$, there exist a unique arrow in $C(0, X)$, e.g. $C(0, X)$ is a singleton.
- Dually, we say 1 is a final object (sometimes terminal) if for all $X \in |C|$, $C(X, 1)$ is a singleton, e.g. there exists a unique arrow from X to 1.
- In Sets these are the empty set and any singleton set respectively.

Universal Arrows

- Not surprisingly, we can generalize this to arbitrary functorial objects.
- Universal arrows are initial objects in a some comma category, i.e. given $\mathcal{F} : \mathsf{C} \to \mathsf{D}$, and for each $X \in |\mathsf{D}|$, the universal arrow from X to $\mathcal{F}$ is the initial object is the comma category $X/\mathcal{F}$

Example: Diagonal Functor

For an *index* category J, and any category C, the diagonal functor $\Delta : \mathrm{C} \to \mathrm{Cat}(J, \mathrm{C})$ maps all objects $X \in |\mathrm{C}|$ to a *functor* denoted $X_\Delta : J \to \mathrm{C}$ such that:

- $(X\Delta(j) = X$ for all $j \in |J|$
- $X\Delta(u) = 1_X$ for all $u \in J$
- Any $f \in \mathrm{C}(X, Y)$ is mapped to the natural transformation $f\Delta : X\Delta \Rightarrow Y\Delta$ such that $(f\Delta)_j = f$ for all $|J|$

Example: Co-limits

- A co-cone to a functor $\mathcal{D} : J \to \mathsf{C}$ is an object in D/Δ.
- A co-limit of $\mathcal{D}$ consists of a family of arrows $\{\mu_i\}_{|J|}$ such $\mu_i = \mathcal{D}(u); \mu_j$ for every $u \in J(i,j)$
- co-limits are unique up to isomorphism, i.e. for any other family $\{\tau_i\}_{|J|}$ such that $\tau_i = \mathcal{D}(u); \tau_j$ for all $u \in J(i,j)$, then there exists a unique f such that $\mu_i; f = \tau_i$
- That is to say, the <u>colimit of $\mathcal{D}$ is a universal arrow from $\mathcal{D}$ to Δ</u>.
- If J is a directed partially ordered set, then J co-limits are directed colimits , and if J is a total order, then the J co-limits are called inductive colimits
- The dual construction here is a limit

Examples of colimits

- Coproducts
  - Disjoint unions in Sets
  - Free groups in Grp
  - Direct sums in Ab
- Co-equalizers
- Pushouts

Example : (Co)-equalizers

- If J is a category with two objects and two parallel arrows between them, then J limits are equalizers and J colimits are co-equalizers
- In Sets the equalizer of any pair of parallel arrows $f, g : X \to Y$ would be the subset inclusion $\{x | f(x) = g(x)\}$
- In Sets the co-equalizer k is the quotient of Y by the equivalence relation generated by $\{(f(x), g(x)) \mid x \in X\}$

Adjunctions: Galois Connections

- For two pre-orders $(P, \leq)$ and $(Q, \leq)$, let

  L: (P,$\leq$) $\rightarrow (Q, \leq)^{op}$

  R: (Q,$\leq$)$^{op} \rightarrow (P, \leq)$be order preserving functions
- We say (L, R) is a Galois connection or an an adjunction if for all $p \in P, q \in Q$,

$$L(p) \geq q \iff p \leq R(q)$$

- Right adjoints preserve limits and dually left adjoints preserve all colimits

Some logic (back to Pred and Rel)

- Substitution can be defined functorially
- We work in the *fibre* category Pred_I for a specific set I
- Pred_I is the subcategory of Pred whose objects are the predicates $X \subset I$ and whose morphisms are mapped onto the identity function on I, e.g. this is the poset category $(\mathcal{P}(I),)$
- For any $u \in \mathrm{Sets}(I, J)$, the substitution functor $u^* : \mathcal{P}(J) \to \mathcal{P}(I)$ is given by the mapping

$$(Y \subseteq J) \mapsto (\{i \mid u(i) \in Y\} \subseteq I$$

Weakening and Contraction

- Let $\pi : I \times J \to I$, then $\pi^* : \mathcal{P}(I) \to \mathcal{P}(I \times J)$ by sending $X \mapsto \{(i,j) \mid i \in X \wedge j \in J\}$
- Let $\delta : I \to I \times I$ be the cartesian diagonal. Then $\delta^* : \mathcal{P}(I \times I) \to \mathcal{P}(I)$ is given by $Y \mapsto \{i \in I \mid (i,i) \in Y\}$; this replaces two variables of *type* I with a single one, and hence is called contraction

Quantifiers

- Let $Y \subseteq I \times J$. Then
  1. $\exists(Y) := \{i \in I \mid \exists j \in J, (i,j) \in Y\} (\subseteq I)$
  2. $\forall(Y) := \{i \in I \mid \forall j \in J, (i,j) \in Y\} (\subseteq I)$
- The assignments $Y \mapsto \exists(Y)$, $Y \mapsto \forall(Y)$ are functorial on $\mathcal{P}(I \times J) \to \mathcal{P}(I)$

Logical Adjoints To Keep In Mind

- $\exists() \vdash \pi^* \vdash \forall()$
- $Eq \vdash \delta^*$
- $\top \vdash \{-\}$
- $Q \vdash Eq$

The Point of Types

- Types in type theory are a "theory of sorts" if one studies this from a classical point of view
- We are going to identify these Types with CCCs
- The total category captures the logic, which is *fibred* over another category capturing the type theory.
- Guiding Principle: An operation in logic or type theory should correspond to an adjoint, and these provide canonical introduction, elimination, and conversion rules

Reminder: CCCs

A Cartesian closed category C has :

1. finite products
2. exponential objects
3. these are adjoint, e.g. $- \times X \vdash \mathrm{C}(X, -)$ with a co-unit ε^X indicating *evaluation* (*application*) so that for each pair of objects X, Y and $f : Z \times X \to Y$, there is a unique $f' : Z \to \mathrm{C}(X, Y)$ such that $f = (f' \times 1_X); \varepsilon^X_Y$

Type Formers: Product Types

We already have encountered these, i.e.

- (×-Form) If $\Gamma \vdash A : \mathfrak{T}$ and $\Gamma \vdash B : \mathfrak{T}$ then $\Gamma \vdash A \times B \mathfrak{T}$
- (×-Intro)

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

- (×-Elim) If we have $t : A \times B$, then we have $\pi^L(t) : A$ and $\pi^R(t) : B$
- (×-Comp) We have that $\pi^L((a, b)) \equiv a$ and

$\pi^R((a, b)) \equiv b$.

```
Inductive prod (A B : Type) : Type :=
| pair : A -> B -> prod A B.
```

Type Formers: Dependent function types

- We model family of types via $\beta : \alpha \to \mathfrak{T}$; such β are dependent types (or `families of types`)
- This construction is generalized as a Π-type.
- The entities of a Π-type are functions whose codomain type varies on the domain to which the *dependent function* is applied

Type Formers: Dependent function types

- (Π-form) If $\Gamma \vdash A : \mathfrak{T}$ and $\Gamma, x : A \vdash B : \mathfrak{T}$ then $\Gamma \vdash (\Pi_{x:A} B) : \mathfrak{T}$
- (Π-intro) If $\Gamma, x : A \vdash b : B$ then $\Gamma \vdash (\lambda(x : A).(b : B) : (\Pi_{(x:A)} B)$
- (Π-Elim) If $\Gamma \vdash f : \prod_{(x:A)} B$ and $\Gamma \vdash (a : A)$, then $\Gamma \vdash f(a) : B[x := a]$
- (Π-Comp) If $\Gamma, x : A \vdash b : B$ and $\Gamma \vdash a : A$ then

$$\Gamma \vdash (\lambda(x : A).(b : B))(a) \equiv b[x := a] : B[x := a]$$

- (Π-Uniq) If $\Gamma \vdash f : \prod_{(x:A)} B$ then $\Gamma \vdash f \equiv (\lambda x.f(x)) : \prod_{(x:A)} B$
- The ordinary function type $A \to B := \prod_{(x:A)} B$ is attained when x does not freely occur in B

Type Formers: Dependent pair types

- (Σ-Form) If $\Gamma \vdash A : \mathfrak{T}$ and $\Gamma, x : A \vdash B : \mathfrak{T}$, then $\Gamma \vdash \sum_{(x:A)} B : \mathfrak{T}$
- (Σ-Intro)

$$\frac{\Gamma, x : A \vdash B : \mathfrak{T} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[x := a]}{\Gamma \vdash (a, b) : \sum_{(x:A)} B}$$

- (Σ-Elim)

$$\frac{\begin{array}{c}\Gamma, z : \sum_{x:A} B \vdash C : \mathfrak{T} \\ \Gamma, x : A, y : B \vdash g : C[z := (x, y)] \qquad \Gamma \vdash p : \sum_{x:A} B\end{array}}{\Gamma \vdash \mathtt{ind}_{\sum_{x:A} B}(z.C, x.y.g, p) : C[z := p]}$$

Type Formers: Dependent pair types (Cont'd)

- (Σ-Comp)

$$\frac{\Gamma, z : \sum_{x:A} B \vdash C : \mathfrak{T} \qquad \Gamma, x : A, y : B \vdash g : C[z := (x,y)] \qquad \Gamma \vdash a' : A \qquad \Gamma \vdash b' : B[x := a']}{\Gamma \vdash \mathtt{ind}_{\sum_{(x:A)} B}(z.C, x.y.g, (a', b')) \equiv g[x := a', y := b')] : C[z := (a', b'}$$

- Similarly to the product type, in $\sum_{(x:A)} B$, if x does not freely occur in B, then $A \times B :\equiv \sum_{(x:A)} B$

Type formers: Coproduct types

- (+-form) If $\Gamma \vdash A : \mathfrak{T}$ and $\Gamma \vdash B : \mathfrak{T}$, then $\Gamma \vdash A + B : \mathfrak{T}$
- (+-Intro1)

$$\frac{\Gamma \vdash A : \mathfrak{T} \qquad \Gamma \vdash a : A \qquad \Gamma \vdash B : \mathfrak{T}}{\Gamma \vdash \mathtt{inl}(a) : A + B}$$

and similarly we introduce inr

- (+-elim)

$$\frac{\begin{array}{cc} \Gamma, z : (A+B) \vdash C : \mathfrak{T} & \Gamma, x : A \vdash c : C[z := \mathtt{inl}(x)] \\ \Gamma \vdash e : A + B & \Gamma, y : B \vdash d : C[z := \mathtt{inr}(x)] \end{array}}{\Gamma \vdash \mathtt{ind}_{A+B}(z.C, x.c, y.d, e) : C[z := e]}$$

- There are two computation rules defined with respect to the two introduction rules, e.g.

$$\mathtt{ind}_{A+B}(z.C, x.c, y.d, \mathtt{inl}(a)) \equiv c[x := a] : C[z := \mathtt{inl}(a)]$$

$$\mathtt{ind}_{A+B}(z.C, x.c, y.d, \mathtt{inr}(b)) \equiv c[y := b] : C[z := \mathtt{inr}(b)]$$

Type Formers: The Empty and Unit Types

- (0-Form) Given any Γ, $\Gamma \vdash \mathbf{0} : \mathfrak{T}$
- (0-Elim) If $\Gamma, x : \mathbf{0} \vdash C : \mathfrak{T}$ and $\Gamma \vdash a : \mathbf{0}$ then $\Gamma \vdash \mathtt{ind}_{\mathbf{0}}(\mathtt{x.C}, \mathtt{a}) : \mathtt{C}[\mathtt{x} := \mathtt{a}]$
- In the induction rule $\mathtt{ind}_0$, x is bound in C. Importantly, there are no introduction or computation rules on the empty type
- (1-Form) Given any Γ, $\Gamma \vdash \mathbf{1} : \mathfrak{T}$
- (1-Intro) Given any Γ, $\Gamma \vdash \star : \mathbf{1}$
- (1-Elim)

$$\frac{\Gamma, x : \mathbf{1} \vdash C : \mathfrak{T} \qquad \Gamma, y : \mathbf{1} \vdash c : C[x := y] \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathtt{ind}_{\mathbf{1}}(x.C, y.c, a) : C[x := a]}$$

- (1-Comp)

$$\frac{\Gamma, x : \mathbf{1} \vdash C : \mathfrak{T} \qquad \Gamma, y : \mathbf{1} \vdash c : C[x := y]}{\Gamma \vdash \mathtt{ind}_{\mathbf{1}}(x.C, y.c, \star) \equiv c[y := \star] : C[x := \star]}$$

Type formers: Natural Numbers

- ($\mathbb{N}$ -Form) Given any Γ, $\Gamma \vdash \mathbb{N} : \mathfrak{T}$
- ($\mathbb{N}$ -Intro$_1$) Given any Γ, $\Gamma \vdash 0 : \mathbb{N}$
- ($\mathbb{N}$ -Intro$_2$) Given $\Gamma \vdash n : \mathbb{N}$, then $\Gamma \vdash S(n) : \mathbb{N}$
- ($\mathbb{N}$ Elim)

$$\frac{\begin{array}{cc} \Gamma, x : \mathbb{N} \vdash C : \mathfrak{T} & \Gamma \vdash c_0 : C[x := 0] \\ \Gamma \vdash n : \mathbb{N} & \Gamma, x : \mathbb{N}, y : C \vdash c_s : C[x := S(x)] \end{array}}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n) : C[x := n]}$$

- There are two computation rules, one defined on our fixed point 0, and the other defined on our successor map $S : \mathbb{N} \to \mathbb{N}$.

- (N-Comp$_1$)

$$\frac{\Gamma, x:\mathbb{N} \vdash C : \mathfrak{T} \qquad \Gamma \vdash c_0 : C[x := 0] \qquad \Gamma \vdash n : \mathbb{N} \qquad \Gamma, x:\mathbb{N}, y:C \vdash c_s : C[x := S(x)]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, 0) \equiv c_0 : C[x := 0]}$$

- (N-Comp$_2$)

$$\frac{\Gamma, x:\mathbb{N} \vdash C : \mathfrak{T} \qquad \Gamma \vdash c_0 : C[x := 0] \qquad \Gamma \vdash n : \mathbb{N} \qquad \Gamma, x:\mathbb{N}, y:C \vdash c_s : C[x := S(x)]}{\Gamma \vdash \mathtt{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, S(n)) \equiv}$$

$$c_s[x := n, y := \mathtt{ind}_{\mathbb{N}}(x.C, c_0, x..c_s, n)] : C[x := S(n)]$$

- What if x is free in B? Then $f : \prod_{(x:A)} B(x)$ is the name of a dependent function with family $B : A \to \mathfrak{T}$, such that there is some $\Phi : B(x)$ that may involve $x : A$.
- Applying a dependent function f to an argument $a : A$ is equivalent to an element $f(a) : B(a)$.
- We may consider the product $A \times B$ to be the left adjoint of the *exponential* $B \to C$.
- Let us define the recursor $\mathtt{rec}_{A\times B} : \prod_{C:\mathfrak{T}}(A \to B \to C) \to (A \times B \to C)$ with the defining equation

$$\mathtt{rec}_{A\times B}(C, g, (a, b)) :\equiv g(a)(b)$$

and

$$\pi^L_{A,B} :\equiv \mathtt{rec}_{A\times B}(A, \lambda(a : A).\lambda(b : B).b$$

$$\pi^R_{A,B} :\equiv \mathtt{rec}_{A\times B}(B, \lambda(a : A).\lambda(b : B).b$$

Revisiting The Dependent Functions and the Product Type

Let's check that $\sum_{x:A} B \equiv A \times B$. The recursion principle says in order to define a non-dependent function

$$f : (\sum_{(x:A)} B(x)) \to C$$

we provide

$$g : \prod_{(x:A)} B(x) \to C$$

so that $f((a,b)) :\equiv g(a)(b)$. By the defining equation

$$\pi^L_{A,B}((a,b)) :\equiv a$$

we derive $\pi^L_{A,B} : (\sum_{(x:A)} B(x)) \to A$ and given (a,b), $b : B(a)$, the second projection is a dependent function

$$\pi^R_{A,B} : \prod_{p:\sum_{(x:A)} B(x)} B(\pi^L_{A,B}(p))$$

Revisiting The Dependent Functions and the Product Type

- The *induction principle* (i.e. the *dependent eliminator*) then says to build a dependent function *out of* a Σ-type into a family $C : (\sum_{(x:A)} B(x)) \to \mathfrak{T}$, we need

$$g : \prod_{(a:A)} \prod_{(b:B(a))} C((a, b))$$

- We will defining an inhabitant $f : \prod_{p:\sum_{(x:A)} B(x)} C(p)$ using

$$C(p) :\equiv B(\pi^L(p))$$

to define

$$\pi^R_{A,B} : \prod_{p:\sum_{(x:A)} B(x)} B(\pi^L_{A,B}(p))$$

via $\pi^R_{A,B}((a, b)) :\equiv b$, so that $f \equiv \pi^R$, and $B(\pi^L((a, b))) \equiv B(a)$

Key Ideas

- Path Spaces
- Fibrations
- Equivalences
- Higher Inductive Types
- Flattening Lemma
- The Fundamental Group of the circle

Homotopy Hypothesis/Theorem

- Every topological space X has a *fundamental* ω *groupoid* whose k-morphisms are the k-dimensional paths in X.
- Depending on how we define an ω-groupoid, there is a homotopy theory preserving adjunction between the fundamental ω-groupoid of a space X and the geometric realization of a ω-groupoid as a space.

Higher Groupoid Structure

- An element $p : x =_A y$ is a path from x to y
- $p, q : x =_A y$ are parallel, and $r : p =_{x=_A y} q$ can be thought of as a 2-path or a homotopy, and $r =_{p_{x=_A y} q} s$ is a 3-path,and so on...
- The higher groupoid structure arises from the induction principle for identity types
- The induction principle for identity types says if we want to construct an object (or prove a statement) depending on a path $p : x =_A y$, then it will suffice to construct an object (argument) in the case where $x \equiv y$ and $p \equiv \mathtt{refl}_x : x = x$
- The induction principle also endows each type with the structure of an ω functor.

Realizing the higher groupoid structure

Recall this Induction Principle of Identity Types amounts to: If

- for every $x, y : A$, and every $p : x =_A y$, we have a type $D(x, y, p)$
- for every $a : A$ we have an element $d(a) : D(a, a, \mathtt{refl}_a)$

then

- there exists $\mathtt{ind}_{=_A}(D, d, x, y, p) : D(x, y, p)$ for every $x, y : A$ and $p : x =_A y$ such that

$$\mathtt{ind}_{=_A}(D, d, a, a, \mathtt{refl}_a) \equiv d(a)$$

Coq-HoTT

- So far we've flirted with Coq and the Homotopy Type Theory fork of the language
- Now we'll look at the actual code in conjunction with the slides (after all the whole point of this endeavor is to learn about *proving* things using HoTT, and the *use* of HoTT instantiated in Coq is one way to do that)
- I recommend downloading the HoTT code from github

Theorem: $\prod_{A:\mathfrak{T}} \prod_{x,y:A} (x =_A y) \to (y =_A x)$

- For each $x, y : A$ and $p : x =_A y$, we want to construct $p^{-1} : y =_A x$
- By induction, it will suffice to do this in the case of $y \equiv x$ and p is $\texttt{refl}_x$.
- In this case, $x =_A x \equiv x =_A y \equiv y =_A x$, and so $\texttt{refl}_x^{-1} :\equiv \texttt{refl}_x$.
- The general case follows by the induction principle and the conversion $\texttt{refl}_x^{-1} \equiv \texttt{refl}_x$, specifically

$$\lambda A.\lambda x.\lambda y.\lambda p.\texttt{ind}_{=_A}((\lambda x.\lambda y.\lambda p.(y =_A x)), (\lambda x.\texttt{refl}_x), x, y, p)$$

$$: \prod_{A:\mathfrak{T}} \prod_{x,y:A} (x =_A y) \to (y =_A x)$$

- In particular, we defined a dependent path, i.e. a *path lying over other paths*

Constructing path inverses as a dependent path

- We may assume that $A : \mathfrak{T}$ and that we have the type family

$$D : \prod_{(x,y:A)} \prod_{(p:x=_A y)} \mathfrak{T}$$

defined by $D(x, y, p) :\equiv (y =_A x)$.

- We may consider D to be a function assigning any $x, y : A$ and $p : x =_A y$ to a *type*, here $y =_A x$.
- We have that $d :\equiv \lambda x.\mathtt{refl}_x : \prod_{x:A} D(x, x, \mathtt{refl}_x)$ so that for each $p : (x =_A y)$ the induction principle for identity types gives us an element

$$\mathtt{ind}_{=_A}(D, d, x, y, p) : (y =_A)$$

- We define the desired inverse function $(-)^{-1} :\equiv \lambda p.\mathtt{ind}_{=_A}(D, d, x, y, p)$ with $\mathtt{refl}_x^{-1} \equiv \mathtt{refl}_x$ following from the conversion rule

$$\mathtt{ind}_{=_A}(D, d, a, a, \mathtt{refl}_a) \equiv d(a)$$

Tale of Two Induction Principles: $\prod_{A:\mathfrak{T}} \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$

- We are going to build a witness that concatenates paths, e.g. $p \centerdot q : x =_A z$
- We introduce $A : \mathfrak{T}$ and then define family $D : \prod_{x,y:A} \prod_{p:x=_y} \mathfrak{T}$ such that
$$D(x, y, p) :\equiv \prod_{z:A} \prod_{q:y=_A z} (x =_A z)$$
- To apply the induction principle of identity types to D, we need to construct a witness of type $\prod_{x:A} D(x, x, \mathtt{refl}_x)$
- To do this we will define a simpler type family E and apply the induction principle of identity types to that first.

$\prod_{A:\mathfrak{T}} \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$

- Define $E : \prod_{x:A} D(x, x, \mathtt{refl}_x)$ by type family $E(x, z, q) :\equiv (x =_A z)$.
- $e(x) :\equiv \mathtt{refl}_x : E(x, x, \mathtt{refl}_x)$ since $E(x, x, \mathtt{refl}_x) \equiv (x =_A x)$
- Applying IPIT to (E,e), we have $d(x, z, q) : \prod_{x,z:A} \prod_{q:x=_A z} E(x, z, q)$, i.e.
$$d(x, z, q) : \prod_{x,z:A} \prod_{q:x=_A z} (x =_A z)$$
- Applying IPIT to (D,d), we have $\lambda x.\lambda y.\lambda z.\mathtt{ind}_{=_A}(D, \mathtt{ind}_{=_A}(E, e, x, z, q), x, y, p) : (x =_A y) \to (y =_A z) \to (x =_A z)$
- In particular $\mathtt{refl}_x \centerdot \mathtt{refl}_x \equiv \mathtt{refl}_x$ by the double induction on paths p, q; if we only inducted on q, we would have a proof that $p \centerdot \mathtt{refl}_x \equiv p$.

The BIG table

So far, we have this picture

Equality	Homotopy	ω Groupoid
reflexivity	constant path	identity morphism
symmetry	inversion of paths	inverse morphism
transitivity	concatenation of paths	composition of morphisms

Other Groupoid properties

- For all types A, and for all $x, y, z, q : A$ and paths $p : x = y$, $q : y = z$ and $r : z = w$, we have
- (unit laws) $\mathtt{ru}_p : p = p \centerdot \mathtt{refl}_y$ and $\mathtt{lu}_p : \mathtt{refl}_x \centerdot p$
- $p^{-1} \centerdot p = \mathtt{refl}_y$ and $p \centerdot p^{-1} = \mathtt{refl}_x$
- $(p^{-1})^{-1} = p$
- $p \centerdot (q \centerdot r) = (p \centerdot q) \centerdot r$
- These defined paths also satisfy their own coherence laws which are higher paths, and so on, all the way up to ω (this notion is made precise via a globular operad)
- Homotopy type theory has that all this structure can be *proven* starting from the inductive properties of identity types

Loop space $\Omega(A, a)$

- Unlike set theory, the proposition $a = a$ carries a lot of information since the proposition is a path from a point to itself, e.g. a *loop*
- Given a type A and an element of A, we define the loop space $\Omega(A, a)$ to be the type $a =_A a$.
- That's right, the loop space is identified with the identity type.
- Any two elements of $\Omega(A, a)$ are paths with the same start and endpoints, so they can be concatenated, thus we have a binary operation $\Omega(A, a) \times \Omega(A, a) \to \Omega(A, a)$.
- We let $\Omega^2(A, a)$ denote the space of \$2-\$dimensional loops on the identity loop, i.e. $\texttt{refl}_a =_{a =_A a} \texttt{refl}_a$

Eckmann-Hilton: Composition in $\Omega^2(A)$ is abelian

- Composition of 1-loops induces *horizontal composition* $\star : \Omega^2(A) \times \Omega^2(A) \to \Omega^2(A)$, such that $\alpha \star \beta : p \centerdot q = q \centerdot s$ with $a, b, c : A$ and

$$p : a = b, q : a = b, r : b = c, s : b = c, \alpha : p = q, \beta : r = s$$

- Define

$$\alpha \centerdot_r r : p \centerdot r = q \centerdot r$$

  by path induction on r so that $\alpha \centerdot_r \mathtt{refl}_b \equiv \mathtt{ru}_p^{-1} \centerdot \alpha \centerdot \mathtt{ru}_q$
- Similarly induct on q for $q \centerdot_l \beta : q \centerdot r = q \centerdot s$ with $\mathtt{lu}_s$

Eckmann-Hilton (Cont'd)

- The $\cdot_l, \cdot_r$ operations are called whiskering, so that $\alpha \cdot_r r$ and $q \cdot_l \beta$ are composable 2-paths from which we define $\alpha \star \beta :\equiv (\alpha \cdot_r r) \cdot (q \cdot_l \beta)$
- Supposing $a \equiv b \equiv c$ so that $p, q, r, s \in \Omega(A, a)$, and further $p \equiv q \equiv r \equiv s \equiv \mathtt{refl}_a$, then $\alpha, \beta : \mathtt{refl}_a = \mathtt{refl}_a$ are composable in both orders,i.e.

$$\alpha\beta \equiv (\alpha \cdot_r \mathtt{refl}_a) \cdot (\mathtt{refl}_a \cdot_l \beta) = \mathtt{ru}^{-1}_{\mathtt{refl}_a} \cdot \alpha \cdot \mathtt{ru}_{\mathtt{refl}_a} \cdot \mathtt{lu}^{-1}_{\mathtt{refl}_a} \cdot \beta \cdot \mathtt{lu}_{\mathtt{refl}_a}$$

$$\equiv \mathtt{refl}^{-1}_{\mathtt{refl}_a} \cdot \alpha \cdot \mathtt{refl}_{\mathtt{refl}_a} \cdot \mathtt{refl}^{-1}_{\mathtt{refl}_a} \cdot \beta \cdot \mathtt{refl}_{\mathtt{refl}_a} = \alpha \cdot \beta$$

Pointed Types and their Loop spaces

- A pointed type (A,a)$ is a type $A : \mathfrak{T}$ with a basepoint

$a : A$, and we write $\mathfrak{T}_\bullet :\equiv \sum_{(A:\mathfrak{T})} A$ for the type of pointed types in $\mathfrak{T}$

- $\Omega(A, a) :\equiv ((a =_A a), \mathtt{refl}_a)$ so that an element of it will be a loop at a.
- For each $n : \mathbb{N}$, the n-fold iterated loop space $\Omega^n(A, a)$ of (A,a) is defined recursively

$$\Omega^0(A, a) :\equiv (A, a)$$

$$\Omega^{n+1}(A, a) :\equiv \Omega^n(\Omega(A, a))$$

Functions are Functors (on paths)

- This amounts to saying that *functions respect equality* in Type theory
- Let's define $\mathsf{ap}_f : (x =_A y) \to (f(x) =_B f(y))$ as applying functions to paths by induction
- Suppose that p is $\mathtt{refl}_x$, and define $\mathsf{ap}_f(p) :\equiv \mathtt{refl}_{fx} : f(x) = f(x)$. By path induction we're done
- In fact
- $\mathsf{ap}_f(p \centerdot q) = \mathsf{ap}_f(p) \centerdot \mathsf{ap}_f(q)$
- $\mathsf{ap}_f(p^{-1}) = \mathsf{ap}_f(p)^{-1}$
- $\mathsf{ap}_g(\mathsf{ap}_f(p)) = \mathsf{ap}_{g \circ f}(p)$
- $\mathsf{ap}_{\mathsf{id}_A(p)=p}$

Type Families are Fibrations

- Given a type family P over A, and a path $p : x =_A y$, there is a function $p_* : P(x) \to P(y)$ defined through $\texttt{transport}^P$
- By path induction again, it suffices to assume that $p \equiv \texttt{refl}_x$, and in turn $(\texttt{refl}_x)_* : P(x) \to P(x)$ by the identity function.
- Topologically, this is *path lifting* in a *fibration*, if we think of $P : A \to \mathfrak{T}$ as a *fibration* with base space A and $P(x)$ as the fibre over x, so that $\sum_{x:A} P(x)$ is the total space of the fibration, with the first projection as the natural projection
- We can define $\texttt{lift}(u, p) : (x, u) = (y, p_*(u))$ in $\sum\limits_{x:A} P(x)$
- We can regard $f : \prod\limits_{x:A} P(x)$ as a section of the fibration P, as f shows that P is fiberwise inhabited

- Under the propositions-as-types interpretation, two (dependently) typed functions f, g are *the same* if $\prod_{x:A}(f(x) = g(x))$ is inhabited, e.g. there is a functorial equivalence (continuous path)
- Such a functorial equivalence is a type of natural isomorphism or homotopy, i.e. $(f \sim g) :\equiv \prod_{x:A}(f(x) = g(x))$; this is not the same thing as identifying $f = g$
- Homotopies are automatically natural transformations, as for any $H : f \sim g$ and $p : x =_A y$, $H(x) \centerdot g(p) = f(p) \centerdot H(y)$ by induction on p, and noting that ap_f and ap_g will commute on reflexivity, e.g.
$$H(x) = H(x) \centerdot \mathtt{refl}_{g(x)} = \mathtt{refl}_{f(x)} \centerdot H(x) = H(x)$$
- $f : A \to B$ has a quasi-inverse (adjoint equivalence) if $(g : B \to A, \alpha : f \circ g \sim \mathrm{id}_{B, \beta : g \circ f \sim \mathrm{id}_{A)}}$. $\mathtt{qinv}(f)$ denotes the type of these *adjoints*.

Equivalences

- Given $f : A \to B$, define

$$\mathtt{isequiv}(f) :\equiv (\sum_{g:B\to A} (f \circ g \sim \mathrm{id}_{B}) \times (\sum_{h:B\to A} (h \circ f \sim \mathrm{id}_{A})$$

- An equivalence from A to B is some $f : A \to B$ with an inhabitant of $\mathtt{isequiv}(f)$, e.g. a proof that f is an equivalence.
- Let $(A \simeq B) :\equiv \sum_{f:A\to B} \mathtt{isequiv}(f)$.
- In HoTT we use *equivalence* in general and *isomorphism* when the types behave like sets.

The Higher Groupoid Structure of Type Formers

- If $P : a \to \mathfrak{T}$ is built up fiberwise via type forming rules, then $\mathtt{transport}^P(p, -)$ is characterized up to homotopy via the operations on the data that went into P
- If P(x)$\equiv$ B(x)$\times$ C(x)\$, then

$$\mathtt{transport}^P(p,(b,c)) = (\mathtt{transport}^B(p,b), \mathtt{transport}^C(p,c))$$

- A deficiency: the characterizations of identity type, transport, etc, are not necessarily judgemental equalities in other type theories
- Not all identity types can be determined by induction over the construction of types (e.g. most nontrivial higher inductive types)
- An axiom is an 'atomic' element declared to inhabit some specified type, whereas a theorem has to be declared and constructed.

Dependent product types and function extensionality

- *The* equivalence axiom for Π-types is *function extensionality*, i.e. for any A,B,f,g the function

$$\texttt{happly} : (f = g) \to \prod_{x:A} (f(x) =_{B(x)} g(x))$$

is an equivalence

- This axiom can be turned into a theorem (using the univalence axiom and defining the interval type later)
- The quasi-inverse $\texttt{funext} : (\prod_{x:A} (f(x) = g(x))) \to (f = g)$ can be regarded as an introduction rule, `happly` as an elimination rule, and the homotopies witnessing that `funext` as a quasi-inverse to `happly` become propositional computation rules

Universes

- So far we've elided something crucial. There isn't quite one single

universe type $\mathfrak{T}$, where a universe is a type whose elements are types.

- To avoid paradoxes, we introduce a hierarchy on the universes

$$\mathfrak{T}_0 : \mathfrak{T}_1 : \mathfrak{T}_2 : \cdots$$

but we don't even need to have this be a strict hierarchy; any poset will do so long as the universes are cumulative, e.g. $A : \mathfrak{T}_i$ then $A : \mathfrak{T}_{i+1}$

- Given $A, B : \mathfrak{T}$, it makes sense to form the identity type $A =_{\mathfrak{T}} B$.
- What we mean by univalence is the identification of $A =_{\mathfrak{T}} B$ with the type $A \simeq B$.

- Consequently, *equivalent types may be identified*
- And $\mathtt{idtoeqv} : (A =_{\mathfrak{U}} B) \to (A \simeq B)$ defined by $\mathtt{idtoeqv}(p) :\equiv p_*$ is an equivalence
- (Intro) For $A =_{\mathfrak{U}} B$, $\mathtt{ua} : (A \simeq B) \to (A =_{\mathfrak{U}} B)$
- (Elim) $\mathtt{idtoeqv} \equiv \mathtt{transport}^{X \mapsto X} : (A =_{\mathfrak{U}} B) \to (A \simeq B)$
- (Comp) $\mathtt{transport}^{X \mapsto X}(\mathtt{ua}(f), x) = f(x)$
- (Uniqueness) for any $p : A = B$, $p = \mathtt{ua}(\mathtt{transport}^{X \mapsto X(p)})$ with

$$\mathtt{refl}_A = \mathtt{ua}(\mathrm{id}_{A),\mathtt{ua}(f).\mathtt{ua}(g) = \mathtt{ua}(g \circ f), \mathtt{ua}(f)^{-1} = \mathtt{ua}(f^{-1})})$$

Sets

- *Discrete groupoids* behave like sets, e.g. groupoids which are determined by a set of objects and only identity morphisms are the higher morphisms
- Formally, for any $A : \mathfrak{T}$, $\mathtt{isSet}(A) :\equiv \prod_{x,y:A} \prod_{p,q:x=y} p = q$.
- Note, there is no global membership predicate $\in$ as in ZF
- The definining property of a set (a 0-type) is that there are no non-trivial paths; the defining property of a 1-type is that there are no non-trivial paths between paths, e.g.

$$\prod_{x,y:A} \prod_{p,q:x=y} \prod_{r,s:p=q} (r = s)$$

n-Types

- *Any* type universe $\mathfrak{T}$ is not a set under this set up- just exhibit a type A and a path $p : A = A$ which is not equal to $\texttt{refl}_A$, say $A = \mathbf{2}$ and $f : A \to A$ which switches the elements, so $\texttt{ua}(f)$ is a path which is not equal to $\texttt{refl}_A$ as otherwise $\texttt{ua}(f) = \mathrm{id}_A$.
- If $\texttt{isSet}(A)$ is inhabited, then A is a 1-type

Propositions-as-Types Revisited

- Statements like LEM or LDN are incompatible with the univalence axiom, e.g. there are types such that $\neg(\neg A) \to A$ is not inhabited
- When types are viewed as propositions, they can contain more information than mere truth or falsity
- The logical constructions on propositions as type must respect this additional information
- A type P is a mere proposition if $\texttt{isProp}(P) :\equiv \forall_{x,y:P}(x = y)$ is inhabited. In this case, P has no *higher* information.

Pointed Mere Propositions $P \simeq \mathbf{1}$

- Define $f : P \to \mathbf{1}$ by $f(x) :\equiv *$ and $g : \mathbf{1} \to P$ by $g(*) :\equiv x_0$, where $x_0 : P$.
- The unit type is a mere proposition
- Since for any $x : P$, $g(f(x)) = x$ since P is a mere proposition (and obviously $f(g(*)) = *$), we have that f and g are quasi-inverses
- A space that is homotopically equivalent to the unit type is contractible
- Moreover, every mere proposition is a set
- A is decidable if $A + \neg A$ is inhabited; $B : A \to \mathfrak{T}$ is decidable if $\prod_{a:A}(B(a) + \neg B(a))$; and A has decidable equality if $\prod_{a,b:A}((a = b) + \neg(a = b))$
- Since we're working in an intuitionistic setting, we don't have LEM

Propositional Truncation

- Some type formers do not preserve mere propositions ($\mathbf{1}$ is mere but $\mathbf{2} = \mathbf{1} + \mathbf{1}$ is not)
- We introduce the type former of propositional truncation (of (-1)-truncation) to truncate a type down to a mere proposition $\|A\|$ has two constructors:
- For any $a : A$, $|a| : \|A\|$
- For any $x, y : \|A\|$, $x = y$
- And the recursion principle states that if B is a mere proposition and $f : A \to B$, then there is an induced $g : \|A\| \to B$ such that $g(|a|) \equiv f(a)$ for all $a : A$

Higher Inductive Types

- In the classical setting, we use CW complexes to inductively define spaces by the collection of points, paths, and higher paths
- *Higher inductive types* are a general schema for defining new types generated by constructors, so that in addition to the points generated in ordinary inductive types, we may also generate paths and so on of the HIT
- Ordinary constructors are known as point constructors while the other constructors are path constructors (or higher constructors)
- Path constructors must specify the starting and ending points of the path

Type Formers: Integers

- The Integers can be found defined HoTT/theories/Spaces/Int.v
- Notice that this is inductive in the sense that we have the natural numbers type encoded as the type Pos

Coequalizers

- Can be found in HoTT/theories/HIT/Coeq.v
- Recall that coequalizers are the colimits of a diagram consisting of two parallel morphisms on two objects X,Y where the object universal construction takes $q : Y \to Q$ can be thought of the smallest equivalence relation such that the two morphisms are identified when working in `Sets`
- In the category of topological spaces, S^1 is the coequalizer of the two inclusion maps from the standard 0-simplex into the 1-simplex

Flattening Lemma

- Can be found HoTT/theories/HIT/Flattening.v
- The flattening lemma says that for such $P : W \to \mathfrak{T}$, the total space $\sum_{x:W} P(x)$ is equivalent to a *flattened* HIT whose constructors are deduced from W and the definition of P
- For instance, let $X : \mathfrak{T}$ and $e : X \simeq X$. We can define a type family $P : S^1 \to \mathfrak{U}$ using this S^1 recursion:

$$P(\texttt{base}) :\equiv X \text{ and } P(\texttt{loop}) := \texttt{ua}(e)$$

so X appears as the fibre $P(\texttt{base})$ of P at the base point and the self-equivalence can be extracted by transporting along `loop`

- Categorically, $\sum_{x:W} P(x)$ is the Grothendieck construction of P, and expresses its UMP as a *lax colimit*

- Let $f, g : B \to A$ and suppose W is an inductive type formed by $c : A \to W$ and $p : \prod_{b:B}(c(f(b)) =_W c(g(b)))$, e.g. W is the (homotopy) coequalizer of f and g. Further, let $C : A \to \mathfrak{T}$ and $D : \prod_{b:B} C(f(b)) \simeq C(g(b))$
- Then we define $P : W \to \mathfrak{T}$ inductively by $P(c(a)) :\equiv C(a)$ and $P(p(b)) :\equiv \mathrm{ua}(D(b))$. Further, let $\tilde{}\{W\} by the HIT generated by $\tilde{c} : \prod_{a:A} C(a) \to \tilde{W}$ and $\tilde{p} : \prod_{b:B} \prod_{y:C(f(b))} (\tilde{c}(f(b), y) =_{\tilde{W}} \tilde{c}(g(b), D(b)(y)))$.
- (Flattening) $\tilde{W} \simeq \sum_{w:W} P(x)$
- This is a taste of the powers of combining HIT with univalence: when W is HIT and $\mathfrak{T}$ is a type universe, we can use the recursion principle of W to define a type family $P : W \to \mathfrak{T}$

FINALLY, $\mathbb{Z} = \pi(S^1)$

- There are in fact several ways to prove this.
- By definition $\pi_1(S^1) = \|\Omega(S^1)\|_0$, so if $\Omega(S^1) = \mathbb{Z}$, and $\mathbb{Z}$ is a set, the desired result follows by congruence.
- Recall that classically, the proof uses the the winding map $w : \mathbb{R} \to S^1$, which is a fibration, e.g. the universal cover, of S^1
- A map of fibrations over B which is a homotopy equivalence induces a homotopy equivalence on all fibers
- By contractibility of $\mathbb{R}$ and $P_{\mathtt{base}}S^1$ are both contractible, they are homotopy equivalent and their fibres $\mathbb{Z}$ and $\Omega(S^1)$ over the basepoint are homotopy equivalent
- In particular, the type family defined by $x \mapsto (x_0 = x)$ corresponds to a *path fibration* $P_{x_0}B \to B$ is contractible.
- We'll prove this in `Coq` via the `code,encode,decode` method

Defining S^1 and universal covering

- One way to define S^1 as a HIT is to specify a base point and the path from that point
- In the interest of applying the Flattening lemma, we define S^1 as the coequalizer of two copies of the identity map on the `Unit` type
- We define the universal cover $\mathtt{code} : S^1 \to \mathfrak{T}$ by circle recursion so that $\mathtt{code}(\mathtt{base}) :\equiv \mathbb{Z}$ and $\mathtt{ap}_{\mathtt{code}}(\mathtt{loop}) :\equiv \mathtt{ua}(\mathtt{succ})$
- The loop we choose is the successor/predecessor isomorphism on $\mathbb{Z}$
- Elements here are combinatorial data that act as codes for paths on the circle, so that the integer n codes for the path looping around n times

Transporting along code

- We're claiming that for all integers, $\texttt{transport}^{\mathsf{code}}(\texttt{loop}, x) = x + 1$ and $\texttt{transport}^{\mathsf{code}}(\texttt{loop}^{-1}, x) = x - 1$
- Equationally, we're showing

$$\texttt{transport}^{\mathsf{code}}(\texttt{loop}, x) = \texttt{transport}^{\mathsf{id}}((\mathtt{code}(\texttt{loop}, x))$$

$$= \texttt{transport}^{\mathsf{id}}(\mathtt{ua}(\mathtt{succ}), x) = x + 1$$

and similarly for the inverse loop.

- The idea of this proof is to define equivalences of a map that sends paths to codes
- We define encode : $\prod_{x:S^1} (\mathtt{base} = x) \to \mathtt{code}(x)$ by $\mathtt{encode}\,p :\equiv \mathtt{transport}^{\mathtt{code}}(p, 0)$
- We define decode : $\prod_{x:S^1} \mathtt{code}(x) \to (\mathtt{base} = x)$ by circle induction
- In particular, when proving that this is a well-defined inhabitant, we check that $\mathtt{loop}^{-1}$ respects loop, i.e. there is a path from $\mathtt{loop}^{-1}$ to $\mathtt{loop}^{-1}$ over loop, i.e. that there is a path $\mathtt{transport}^{x \mapsto \mathtt{code}(x) \to (\mathtt{base} = x)}(\mathtt{loop}, \mathtt{loop}^{-1})$ to $\mathtt{loop}^{-1}$

Decode Is Well Defined

We build the define the path by applying the characterization of transport when the outer connective of the fibration is $\to$ so that transport reduces to pre and post composition with transport at the domain and codomain types

$$\mathtt{transport}^{x \mapsto \mathtt{code}(x) = (\mathtt{base} = x)}(\mathtt{loop}, \mathtt{loop}^{-1}) =$$

$$\mathtt{transport}^{x \mapsto (base = x)(\mathtt{loop}) \circ \mathtt{loop}^{-1} \circ \mathtt{transport}^{\mathtt{code}(\mathtt{loop}^{-1})}}$$

$$= (- \,.\, \mathtt{loop}) \circ (\mathtt{loop}^{-1}) \circ \mathtt{transport}^{\mathtt{code}(\mathtt{loop}^{-1})}$$

$$= (- \,.\, \mathtt{loop}) \circ (\mathtt{loop}^{-1}) \circ (\mathtt{pred}) = (n \mapsto \mathtt{loop}^{n-1} \,.\, \mathtt{loop})$$

Wrapping Up

- For all $x : S^1$ and $p : \mathtt{base} = x$, $\mathtt{decode}_x(\mathtt{encode}_x(p)) = p$
- For all $x : S^1$ and $c : \mathtt{code}(x)$, $\mathtt{encode}_x(\mathtt{decode}_x(c)) = c$
- There is a family of equivalences $\prod_{x:S^1} ((\mathtt{base} = x) \simeq \mathtt{code}(x))$
- Instantiating at base gives $\Omega(S^1, \mathtt{base}) \simeq \mathbb{Z}$
- Consequently, $\pi_1(S^1) = \mathbb{Z}$ and for $n > 1$, $\pi_n(S^1) = 0$ since

$$\|\Omega^n(S^1)\|_0 = \|\Omega^{n-1}(\Omega S^1)\|_0 = \|\Omega^{n-1}(\mathbb{Z})\|_0 = \{*\}$$

as $\mathbb{Z}$ is a set and therefore is contractible.