Path induction is a key tool in HoTT.

Path induction - Roughly

To prove $C(x, y, p)$ where $x, y : A$ and $p : x =_A y$ it suffices to prove $C(x, x, \mathrm{refl}_x)$ for all $x : A$

Path induction

Suppose $C(x, y, p)$ is a (dependent) type for each $x, y : A$ and $p : x =_A y$, and suppose there is a function $c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$, then there is a function $f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$.

Does this make sense?

Path Induction

Path induction is a key tool in HoTT.

Path induction - Roughly

To prove $C(x, y, p)$ where $x, y : A$ and $p : x =_A y$ it suffices to prove $C(x, x, \mathrm{refl}_x)$ for all $x : A$

Path induction

Suppose $C(x, y, p)$ is a (dependent) type for each $x, y : A$ and $p : x =_A y$, and suppose there is a function $c : \prod_{x:A} C(x, x, \mathrm{refl}_x)$, then there is a function $f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$.

Does this make sense?

Path Induction

Path induction is a key tool in HoTT.

Path induction - Roughly

To prove $C(x, y, p)$ where $x, y : A$ and $p : x =_A y$ it suffices to prove $C(x, x, \mathsf{refl}_x)$ for all $x : A$

Path induction

Suppose $C(x, y, p)$ is a (dependent) type for each $x, y : A$ and $p : x =_A y$, and suppose there is a function $c : \prod_{x:A} C(x, x, \mathsf{refl}_x)$, then there is a function $f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$.

Does this make sense?

Path Induction

Path induction is a key tool in HoTT.

Path induction - Roughly

To prove $C(x, y, p)$ where $x, y : A$ and $p : x =_A y$ it suffices to prove $C(x, x, \mathsf{refl}_x)$ for all $x : A$

Path induction

Suppose $C(x, y, p)$ is a (dependent) type for each $x, y : A$ and $p : x =_A y$, and suppose there is a function $c : \prod_{x:A} C(x, x, \mathsf{refl}_x)$, then there is a function $f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$.

Does this make sense?

Path Induction

Path induction is a key tool in HoTT.

Path induction - Roughly

To prove $C(x, y, p)$ where $x, y : A$ and $p : x =_A y$ it suffices to prove $C(x, x, \mathsf{refl}_x)$ for all $x : A$

Path induction

Suppose $C(x, y, p)$ is a (dependent) type for each $x, y : A$ and $p : x =_A y$, and suppose there is a function $c : \prod_{x:A} C(x, x, \mathsf{refl}_x)$, then there is a function $f : \prod_{x,y:A} \prod_{p:x=_A y} C(x, y, p)$.

Does this make sense?

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\mathrm{refl}_x : Id_A(x, x)$ (”unit element”)
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ (“Composition”)
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ (“Inverses”)
- Several “group-like” laws hold

“Groupoid” structure

Every type A is a “(weak) ∞-groupoid” with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

- If $x : A$ then there is a special element $\mathrm{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\mathrm{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\mathrm{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\text{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\mathrm{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

The Structure of $Id_A(x, y)$

- If $x : A$ then there is a special element $\text{refl}_x : Id_A(x, x)$ ("unit element")
- If $p : Id_A(x, y)$ and $q : Id_A(y, z)$ then there is an element $r : Id_A(x, z)$ ("Composition")
- For every element $p : Id_A(x, y)$ there is an element $p^{-1} : Id_A(y, x)$ ("Inverses")
- Several "group-like" laws hold

"Groupoid" structure

Every type A is a "(weak) ∞-groupoid" with objects $x : A$ and morphisms $A(x, y) := Id_A(x, y)$.

Proof of inverses on board.

(Lower) Inductive types

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$0 : \mathbb{N} \qquad \text{(element)}$$
$$\mathrm{succ} : \mathbb{N} \to \mathbb{N} \qquad \text{(function)}$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0,f)} : \mathbb{N} \to A$ such that

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathrm{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\ A \arrow[r, "f"'] & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0,f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$

(Lower) Inductive types

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$\begin{aligned} 0 &: \mathbb{N} && \text{(element)} \\ \mathrm{succ} &: \mathbb{N} \to \mathbb{N} && \text{(function)} \end{aligned}$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0, f)} : \mathbb{N} \to A$ such that

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathrm{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\ A \arrow[r, "f"'] & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0, f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$
\begin{aligned}
0 &: \mathbb{N} && \text{(element)} \\
\mathrm{succ} &: \mathbb{N} \to \mathbb{N} && \text{(function)}
\end{aligned}
$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0,f)} : \mathbb{N} \to A$ such that

$$
\begin{tikzcd}
\mathbb{N} \arrow[r, "\mathrm{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\
A \arrow[r, "f"'] & A
\end{tikzcd}
$$

commutes, and such that $rec_{(a_0,f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$

(Lower) Inductive types

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$\begin{aligned} 0 &: \mathbb{N} && \text{(element)} \\ \mathsf{succ} &: \mathbb{N} \to \mathbb{N} && \text{(function)} \end{aligned}$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0,f)} : \mathbb{N} \to A$ such that

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathsf{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\ A \arrow[r, "f"'] & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0,f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$

(Lower) Inductive types

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$\begin{aligned} 0 &: \mathbb{N} && \text{(element)} \\ \mathsf{succ} &: \mathbb{N} \to \mathbb{N} && \text{(function)} \end{aligned}$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0,f)} : \mathbb{N} \to A$ such that

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathsf{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\ A \arrow[r, "f"'] & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0,f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathsf{succ} : \mathbb{N} \to \mathbb{N}$

(Lower) Inductive types

Types can be *inductively* defined. For example, $\mathbb{N}$, is defined by

$$\begin{aligned} 0 &: \mathbb{N} && \text{(element)} \\ \mathrm{succ} &: \mathbb{N} \to \mathbb{N} && \text{(function)} \end{aligned}$$

Universal property / Recursion principle for $\mathbb{N}$

Given any type A, and $a_0 : A$, and a map $f : A \to A$, we get a unique map $rec_{(a_0, f)} : \mathbb{N} \to A$ such that

$$\begin{tikzcd} \mathbb{N} \arrow[r, "\mathrm{succ}"] \arrow[d, dashed, "rec"'] & \mathbb{N} \arrow[d, dashed, "rec"] \\ A \arrow[r, "f"'] & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0, f)}(0) = a_0$.

- We say $\mathbb{N}$ is "constructed freely" from $0 : \mathbb{N}$ and $\mathrm{succ} : \mathbb{N} \to \mathbb{N}$

Higher inductive types

- A new way to construct new types in HoTT is to construct "free types on some generators", called *higher inductive types*
- Whereas "normal" inductive definitions (eg. $\mathbb{N}$) use elements and functions, we allow *paths*, i.e. elements of $Id_A(x, y)$.

Example: Interval

The interval I is constructed freely from $0_I : I$, $1_I : I$ and the *path* $\mathrm{seg} : Id_I(0_I, 1_I)$

- Each higher inductive type comes equipped with its own universal property, similar to the one for $\mathbb{N}$

- A new way to construct new types in HoTT is to construct "free types on some generators", called *higher inductive types*
- Whereas "normal" inductive definitions (eg. $\mathbb{N}$) use elements and functions, we allow *paths*, i.e. elements of $Id_A(x, y)$.

Example: Interval

The interval I is constructed freely from $0_I : I$, $1_I : I$ and the *path* $\mathrm{seg} : Id_I(0_I, 1_I)$

- Each higher inductive type comes equipped with its own universal property, similar to the one for $\mathbb{N}$

Higher inductive types

- A new way to construct new types in HoTT is to construct "free types on some generators", called *higher inductive types*
- Whereas "normal" inductive definitions (eg. $\mathbb{N}$) use elements and functions, we allow *paths*, i.e. elements of $Id_A(x, y)$.

Example: Interval

The interval I is constructed freely from $0_I : I$, $1_I : I$ and the *path* $\mathrm{seg} : Id_I(0_I, 1_I)$

- Each higher inductive type comes equipped with its own universal property, similar to the one for $\mathbb{N}$

Higher inductive types

- A new way to construct new types in HoTT is to construct "free types on some generators", called *higher inductive types*
- Whereas "normal" inductive definitions (eg. $\mathbb{N}$) use elements and functions, we allow *paths*, i.e. elements of $Id_A(x, y)$.

Example: Interval

The interval I is constructed freely from $0_I : I$, $1_I : I$ and the *path* $\text{seg} : Id_I(0_I, 1_I)$

- Each higher inductive type comes equipped with its own universal property, similar to the one for $\mathbb{N}$

Higher inductive types

- A new way to construct new types in HoTT is to construct "free types on some generators", called *higher inductive types*
- Whereas "normal" inductive definitions (eg. $\mathbb{N}$) use elements and functions, we allow *paths*, i.e. elements of $Id_A(x, y)$.

Example: Interval

The interval I is constructed freely from $0_I : I$, $1_I : I$ and the *path* $\mathrm{seg} : Id_I(0_I, 1_I)$

- Each higher inductive type comes equipped with its own universal property, similar to the one for $\mathbb{N}$

Higher Inductive definition of the Circle - $\mathbb{S}^1$

$\mathbb{S}^1$ is constructed freely from

$$\begin{aligned} \mathit{base} &: \mathbb{S}^1 && \text{(element)} \\ \mathrm{loop} &: \mathit{Id}_{\mathbb{S}^1}(\mathit{base}, \mathit{base}) && (\textit{path}!) \end{aligned}$$

$\mathbb{S}^1$ is the "free ∞-groupoid" on these generators.

- There is already a lot of structure implied by these generators for example
  - $\mathrm{refl}_{\mathrm{base}} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\mathrm{loop} \centerdot \mathrm{loop} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\mathrm{loop}^{-1} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\alpha : \mathrm{loop}^{-1} \centerdot \mathrm{loop} =_{\mathrm{base} =_{\mathbb{S}^1} \mathrm{base}} \mathrm{refl}_{\mathrm{base}}$

Higher Inductive definition of the Circle - $\mathbb{S}^1$

$\mathbb{S}^1$ is constructed freely from

$$\begin{aligned} &\mathit{base} : \mathbb{S}^1 && \text{(element)} \\ &\mathrm{loop} : \mathit{Id}_{\mathbb{S}^1}(\mathit{base}, \mathit{base}) && (\mathit{path}!) \end{aligned}$$

$\mathbb{S}^1$ is the "free ∞-groupoid" on these generators.

- There is already a lot of structure implied by these generators for example
  - $\mathrm{refl}_{\mathrm{base}} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\mathrm{loop} \centerdot \mathrm{loop} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\mathrm{loop}^{-1} : \mathit{Id}_{\mathbb{S}^1}(\mathrm{base}, \mathrm{base})$
  - $\alpha : \mathrm{loop}^{-1} \centerdot \mathrm{loop} =_{\mathrm{base} =_{\mathbb{S}^1} \mathrm{base}} \mathrm{refl}_{\mathrm{base}}$

$\mathbb{S}^1$ is constructed freely from

$$\begin{aligned} &\textit{base} : \mathbb{S}^1 && \text{(element)} \\ &\text{loop} : \textit{Id}_{\mathbb{S}^1}(\textit{base}, \textit{base}) && (\textit{path}!) \end{aligned}$$

$\mathbb{S}^1$ is the "free ∞-groupoid" on these generators.

- There is already a lot of structure implied by these generators for example
  - $\text{refl}_{\text{base}} : \textit{Id}_{\mathbb{S}^1}(\text{base}, \text{base})$
  - $\text{loop} \centerdot \text{loop} : \textit{Id}_{\mathbb{S}^1}(\text{base}, \text{base})$
  - $\text{loop}^{-1} : \textit{Id}_{\mathbb{S}^1}(\text{base}, \text{base})$
  - $\alpha : \text{loop}^{-1} \centerdot \text{loop} =_{\text{base} =_{\mathbb{S}^1} \text{base}} \text{refl}_{\text{base}}$

Universal property / Recursion principle for $\mathbb{S}^1$

As for $\mathbb{N}$, since we defined $\mathbb{S}^1$ freely we get a universal mapping property, for mapping out of $\mathbb{S}^1$.

Universal property / Recursion principle for $\mathbb{S}^1$

Given any type A, and $a_0 : A$, and a *path* $p : Id_A(a_0, a_0)$, we get a unique map $rec_{(a_0,p)} : \mathbb{S}^1 \to A$, such that

$$\begin{tikzcd} 1 \arrow[r, "\text{base}"] \arrow[dr, "a_0"'] & \mathbb{S}^1 \arrow[d, dashed, "\text{rec}"] \\ & A \end{tikzcd}$$

commutes, and such that $rec_{(a_0,p)_*}(loop) = p$

As for $\mathbb{N}$, since we defined $\mathbb{S}^1$ freely we get a universal mapping property, for mapping out of $\mathbb{S}^1$.

Universal property / Recursion principle for $\mathbb{S}^1$

Given any type A, and $a_0 : A$, and a *path* $p : Id_A(a_0, a_0)$, we get a unique map $rec_{(a_0,p)} : \mathbb{S}^1 \to A$, such that

$$\begin{tikzcd} 1 \arrow[r, "\text{base}"] \arrow[dr, "a_0"'] & \mathbb{S}^1 \arrow[d, dashed, "\text{rec}"] \\ & A \end{tikzcd}$$

commutes, and such that ${rec_{(a_0,p)}}_*(loop) = p$

As for $\mathbb{N}$, since we defined $\mathbb{S}^1$ freely we get a universal mapping property, for mapping out of $\mathbb{S}^1$.

Universal property / Recursion principle for $\mathbb{S}^1$

Given any type A, and $a_0 : A$, and a *path* $p : Id_A(a_0, a_0)$, we get a unique map $rec_{(a_0,p)} : \mathbb{S}^1 \to A$, such that

$$\begin{tikzcd} 1 \arrow[r, "\mathrm{base}"] \arrow[dr, "a_0"'] & \mathbb{S}^1 \arrow[d, dashed, "\mathrm{rec}"] \\ & A \end{tikzcd}$$

commutes, and such that ${rec_{(a_0,p)}}_*(loop) = p$

Universal property / Recursion principle for $\mathbb{S}^1$

As for $\mathbb{N}$, since we defined $\mathbb{S}^1$ freely we get a universal mapping property, for mapping out of $\mathbb{S}^1$.

Universal property / Recursion principle for $\mathbb{S}^1$

Given any type A, and $a_0 : A$, and a *path* $p : Id_A(a_0, a_0)$, we get a unique map $rec_{(a_0,p)} : \mathbb{S}^1 \to A$, such that

$$\begin{tikzcd} 1 \arrow[r, "\text{base}"] \arrow[dr, "a_0"'] & \mathbb{S}^1 \arrow[d, dashed, "\text{rec}"] \\ & A \end{tikzcd}$$

commutes, and such that ${rec_{(a_0,p)}}_*(loop) = p$

Truncation

We can "rank" our types into a hierarchy according to which level is homotopically trivial.

Propositional truncation

Let A be a type. Then the *propositional truncation* of A, written $||A||_{-1}$, is A "reduced to a logical proposition"

The type $||A||_{-1}$ is freely generated by the function $|a| : A \to ||A||_{-1}$ and the paths $\prod_{a,b:A} a =_A b$.

- Writing the usual axiom of choice:
$$\left(\prod_{(X:F)} \left|\left|\sum_{(x:A(X))} P(x,X)\right|\right|_{-1}\right) \to$$
$$\left|\left|\sum_{(g:\prod_{(X:F)} A(X))} \prod_{(X:F)} P(g(X),X)\right|\right|_{-1}$$

We can "rank" our types into a hierarchy according to which level is homotopically trivial.

Propositional truncation

Let A be a type. Then the *propositional truncation* of A, written $||A||_{-1}$, is A "reduced to a logical proposition"

The type $||A||_{-1}$ is freely generated by the function $|a| : A \to ||A||_{-1}$ and the paths $\prod_{a,b:A} a =_A b$.

- Writing the usual axiom of choice:
$$\left(\prod_{(X:F)} \left|\left|\sum_{(x:A(X))} P(x,X)\right|\right|_{-1}\right) \to$$
$$\left|\left|\sum_{(g:\prod_{(X:F)} A(X))} \prod_{(X:F)} P(g(X),X)\right|\right|_{-1}$$

We can "rank" our types into a hierarchy according to which level is homotopically trivial.

Propositional truncation

Let A be a type. Then the *propositional truncation* of A, written $||A||_{-1}$, is A "reduced to a logical proposition"

The type $||A||_{-1}$ is freely generated by the function $|a| : A \to ||A||_{-1}$ and the paths $\prod_{a,b:A} a =_A b$.

- Writing the usual axiom of choice:

$$\left(\prod_{(X:F)} \left|\left|\sum_{(x:A(X))} P(x,X)\right|\right|_{-1}\right) \to \left|\left|\sum_{(g:\prod_{(X:F)} A(X))} \prod_{(X:F)} P(g(X),X)\right|\right|_{-1}$$

Truncation

We can "rank" our types into a hierarchy according to which level is homotopically trivial.

Propositional truncation

Let A be a type. Then the *propositional truncation* of A, written $||A||_{-1}$, is A "reduced to a logical proposition"

The type $||A||_{-1}$ is freely generated by the function $|a| : A \to ||A||_{-1}$ and the paths $\prod_{a,b:A} a =_A b$.

- Writing the usual axiom of choice:
$$\left(\prod_{(X:F)} \left|\left|\sum_{(x:A(X))} P(x,X)\right|\right|_{-1}\right) \to \left|\left|\sum_{(g:\prod_{(X:F)} A(X))} \prod_{(X:F)} P(g(X),X)\right|\right|_{-1}$$

The hierarchy of types is divided into certain “h-level” accordingly

- At the -1-level we have types that are either contractible (“have only one point”) or empty.
- At the 0-level we have “sets”
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A “reduced to a set”, $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

The hierarchy of types is divided into certain "h-level" accordingly

- At the -1-level we have types that are either contractible ("have only one point") or empty.
- At the 0-level we have "sets"
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A "reduced to a set", $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

General Truncations

The hierarchy of types is divided into certain “h-level” accordingly

- At the -1-level we have types that are either contractible (“have only one point”) or empty.
- At the 0-level we have “sets”
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A “reduced to a set”, $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

General Truncations

The hierarchy of types is divided into certain "h-level" accordingly

- At the -1-level we have types that are either contractible ("have only one point") or empty.
- At the 0-level we have "sets"
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A "reduced to a set", $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

The hierarchy of types is divided into certain "h-level" accordingly

- At the -1-level we have types that are either contractible ("have only one point") or empty.
- At the 0-level we have "sets"
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A "reduced to a set", $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

General Truncations

The hierarchy of types is divided into certain "h-level" accordingly

- At the -1-level we have types that are either contractible ("have only one point") or empty.
- At the 0-level we have "sets"
- At the 1-level we have ordinary groupoids ...

Set truncation, π_0

Let A be a type. Then the *set truncation* is A "reduced to a set", $\pi_0(A)$ also denoted $||A||_0$.

$\pi_0(A)$ is freely generated by the function $|a| : A \to \pi_0(A)$ and paths $\prod_{x,y:A} \prod_{p,q:Id(x,y)} p = q$.

Fundamental groups

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathrm{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathrm{base})) = \mathbb{Z}.$$

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathsf{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathsf{base})) = \mathbb{Z}.$$

Fundamental groups

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathsf{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathsf{base})) = \mathbb{Z}.$$

Fundamental groups

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a\rangle$
- The *loop space* of a pointed type $\langle A, a\rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathrm{refl}_a\rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathrm{base})) = \mathbb{Z}.$$

Fundamental groups

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathrm{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathrm{base})) = \mathbb{Z}.$$

Fundamental groups

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathrm{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathrm{base})) = \mathbb{Z}.$$

- A *pointed type* is a type A along with some $a : A$, denoted $\langle A, a \rangle$
- The *loop space* of a pointed type $\langle A, a \rangle$, denoted $\Omega(A, a)$ is the pointed type $\langle Id_A(a, a), \mathrm{refl}_a \rangle$
- But $\Omega(A, a)$ is not a group! We need it to be a *set* first
- Thus $\pi_0(\Omega(A, a))$ is a set, and it can be shown to be a group

Theorem

The fundamental group of the circle is the integers:

$$\pi_0(\Omega(\mathbb{S}^1, \mathrm{base})) = \mathbb{Z}.$$

Thank you for listening!

- More info: HomotopyTypeTheory.org