Inductive and higher inductive types

Michael Shulman

13 April 2012

Homotopy invariance

Question

Suppose two model categories $\mathcal{M}$, $\mathcal{N}$ present the same $(\infty, 1)$-category $\mathscr{C}$. Do they have the same internal type theory?

Homotopy invariance

Question

Suppose two model categories $\mathcal{M}$, $\mathcal{N}$ present the same $(\infty, 1)$-category $\mathscr{C}$. Do they have the same internal type theory?

1. All type-theoretic operations are homotopy invariant (represent well-defined $(\infty, 1)$-categorical operations).
2. Therefore, any type-theoretic construction performed on equivalent data in $\mathcal{M}$ and $\mathcal{N}$ yields equivalent results.

Homotopy invariance

Question

Suppose two model categories $\mathcal{M}$, $\mathcal{N}$ present the same $(\infty, 1)$-category $\mathscr{C}$. Do they have the same internal type theory?

1. All type-theoretic operations are homotopy invariant (represent well-defined $(\infty, 1)$-categorical operations).
2. Therefore, any type-theoretic construction performed on equivalent data in $\mathcal{M}$ and $\mathcal{N}$ yields equivalent results.
3. All type-theoretic data is terms in (dependent) types, i.e. sections of fibrations. If all objects in $\mathcal{M}$ and $\mathcal{N}$ are cofibrant, any "section" in $\mathscr{C}$ can be represented in both $\mathcal{M}$ and $\mathcal{N}$.

Homotopy invariance

Question

Suppose two model categories $\mathcal{M}$, $\mathcal{N}$ present the same $(\infty, 1)$-category $\mathscr{C}$. Do they have the same internal type theory?

1. All type-theoretic operations are homotopy invariant (represent well-defined $(\infty, 1)$-categorical operations).
2. Therefore, any type-theoretic construction performed on equivalent data in $\mathcal{M}$ and $\mathcal{N}$ yields equivalent results.
3. All type-theoretic data is terms in (dependent) types, i.e. sections of fibrations. If all objects in $\mathcal{M}$ and $\mathcal{N}$ are cofibrant, any "section" in $\mathscr{C}$ can be represented in both $\mathcal{M}$ and $\mathcal{N}$.
4. The only trouble is with asserting computational equalities, e.g. "let G be a group with computationally associative multiplication". If we stick with properties that can be expressed in the type theory, we are fine.

Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

Positive types

Recall: positive types are characterized by their introduction rules.

In fact, any choice of introduction rule(s) determines a positive type in an algorithmic way.

- The derived eliminator literally does a case analysis on the introduction rules.
- We call these introduction rules constructors.

Positive types

Example (Coproduct types)

- Introduction: $\mathsf{inl}\colon A \to A + B$ and $\mathsf{inr}\colon B \to A + B$
- Elimination: If $(x\colon A) \vdash (c_A : C(\mathsf{inl}(x)))$ and $(y\colon B) \vdash (c_B : C(\mathsf{inr}(y)))$, then for $p\colon A + B$ we have $\mathsf{case}(p, c_A, c_B) : C(p)$.

Positive types

Example (Coproduct types)

- Introduction: $\mathrm{inl}\colon A \to A + B$ and $\mathrm{inr}\colon B \to A + B$
- Elimination: If $(x\colon A) \vdash (c_A : C(\mathrm{inl}(x)))$ and $(y\colon B) \vdash (c_B : C(\mathrm{inr}(y)))$, then for $p\colon A + B$ we have $\mathrm{case}(p, c_A, c_B) : C(p)$.

Example (Empty type)

- Introduction:
- Elimination: If (nothing), then for $p\colon \emptyset$ we have $\mathrm{abort}(p)\colon C(p)$.

The natural numbers

The natural numbers are a positive type.

1. Formation: There is a type $\mathbb{N}$.
2. Introduction: $0 \colon \mathbb{N}$, and $(x \colon \mathbb{N}) \vdash (\mathsf{s}(x) : \mathbb{N})$.

The natural numbers

The natural numbers are a positive type.

1. Formation: There is a type $\mathbb{N}$.
2. Introduction: $0\colon \mathbb{N}$, and $(x\colon \mathbb{N}) \vdash (\mathsf{s}(x) : \mathbb{N})$.

A new feature: the input of the constructor "s" involves something of the type $\mathbb{N}$ being defined!

We intend, of course, that all elements of $\mathbb{N}$ are generated by *successively* applying constructors.

$$0, s(0), s(s(0)), s(s(s(0))), \dots$$

The natural numbers

1. Formation: There is a type $\mathbb{N}$.
2. Introduction: $0 \colon \mathbb{N}$, and $(x \colon \mathbb{N}) \vdash (\mathsf{s}(x) : \mathbb{N})$.
3. Elimination? If $c_0 \colon C(0)$ and $(x \colon \mathbb{N}) \vdash (c_{\mathsf{s}} : C(\mathsf{s}(x)))$, then for $p \colon \mathbb{N}$ we have $\mathsf{match}(p, c_0, c_{\mathsf{s}}) : C(p)$.

The natural numbers

1. Formation: There is a type $\mathbb{N}$.
2. Introduction: $0\colon \mathbb{N}$, and $(x\colon \mathbb{N}) \vdash (\mathsf{s}(x) : \mathbb{N})$.
3. Elimination? If $c_0\colon C(0)$ and $(x\colon \mathbb{N}) \vdash (c_\mathsf{s} : C(\mathsf{s}(x)))$, then for $p\colon \mathbb{N}$ we have $\mathsf{match}(p, c_0, c_\mathsf{s}) : C(p)$.

But this is not much good; we need to recurse.

3. Elimination: If $c_0\colon C(0)$ and
$$(x\colon \mathbb{N}), \color{red}{(r\colon C(x))} \vdash (c_\mathsf{s} : C(\mathsf{s}(x)))$$
then for $p\colon \mathbb{N}$ we have $\mathsf{rec}(p, c_0, c_\mathsf{s}) : C(p)$.

The variable r represents the result of the recursive call at x, to be used the computation c_s of the value at $\mathsf{s}(x)$.

Example: Addition

We define addition by recursion on the first input.

$$\begin{aligned} \mathsf{plus}(0, m) &:= m \\ \mathsf{plus}(\mathsf{s}(n), m) &:= \mathsf{s}(\mathsf{plus}(n, m)) \end{aligned}$$

In terms of the rec eliminator, this is

$$(n\colon \mathbb{N}), (m\colon \mathbb{N}) \ \vdash \ \mathsf{plus}(n, m) := \mathsf{rec}(n, m, \mathsf{s}(r))$$

Example: Addition

We define addition by recursion on the first input.

$$\begin{aligned}\mathsf{plus}(0, m) &:= m \\ \mathsf{plus}(\mathsf{s}(n), m) &:= \mathsf{s}(\mathsf{plus}(n, m))\end{aligned}$$

In terms of the rec eliminator, this is

$$(n \colon \mathbb{N}), (m \colon \mathbb{N}) \vdash \mathsf{plus}(n, m) := \mathsf{rec}(n, \color{red}{m}, \mathsf{s}(r))$$

- When $n = 0$, the result is m.

Example: Addition

We define addition by recursion on the first input.

$$\begin{aligned} \mathsf{plus}(0, m) &:= m \\ \mathsf{plus}(\mathsf{s}(n), m) &:= \mathsf{s}(\mathsf{plus}(n, m)) \end{aligned}$$

In terms of the rec eliminator, this is

$$(n \colon \mathbb{N}), (m \colon \mathbb{N}) \vdash \mathsf{plus}(n, m) := \mathsf{rec}(n, m, \mathsf{s}(r))$$

- When $n = 0$, the result is m.
- When n is a successor $\mathsf{s}(x)$, the result is $\mathsf{s}(r)$.
  (As before, r is the result of the recursive call at x.)

The natural numbers

1. Formation: There is a type $\mathbb{N}$.
2. Introduction: $0\colon \mathbb{N}$, and $(x\colon \mathbb{N}) \vdash (\mathsf{s}(x) : \mathbb{N})$.
3. Elimination: If $c_0\colon C(0)$ and

   $$(x\colon \mathbb{N}), (r\colon C(x)) \vdash (c_{\mathsf{s}} : C(\mathsf{s}(x)))$$

   then for $p\colon \mathbb{N}$ we have $\mathsf{rec}(p, c_0, c_{\mathsf{s}}) : C(p)$.
4. Computation:
   - $\mathsf{rec}(0, c_0, c_{\mathsf{s}})$ computes to c_0.
   - $\mathsf{rec}(\mathsf{s}(n), c_0, c_{\mathsf{s}})$ computes to c_{s} with n substituted for x and $\mathsf{rec}(n, c_0, c_{\mathsf{s}})$ substituted for r.

Computing an addition

$$\begin{aligned}\mathsf{plus}(\mathsf{ss0}, \mathsf{sss0}) &:= \mathsf{rec}(\mathsf{ss0}, \mathsf{sss0}, \mathsf{s}(r)) \\ &\rightsquigarrow \mathsf{s}\Big(\mathsf{rec}(\mathsf{s0}, \mathsf{sss0}, \mathsf{s}(r))\Big) \\ &\rightsquigarrow \mathsf{s}\Big(\mathsf{s}\Big(\mathsf{rec}(0, \mathsf{sss0}, \mathsf{s}(r))\Big)\Big) \\ &\rightsquigarrow \mathsf{s}(\mathsf{s}(\mathsf{sss0})) = \mathsf{sssss0}\end{aligned}$$

Other recursive inductive types

Generalized positive types of this sort are called inductive types.

Example (Lists)

For any type A, there is a type $\mathsf{List}(A)$, with constructors

$$\begin{aligned} &\vdash \mathsf{nil}\colon \mathsf{List}(A) \\ (a\colon A), (\ell\colon \mathsf{List}(A)) &\vdash (\mathsf{cons}(a,\ell) : \mathsf{List}(A)) \end{aligned}$$

Other recursive inductive types

Generalized positive types of this sort are called inductive types.

Example (Lists)

For any type A, there is a type $\mathsf{List}(A)$, with constructors

$$\begin{aligned} &\vdash \mathsf{nil}\colon \mathsf{List}(A) \\ (a\colon A), (\ell\colon \mathsf{List}(A)) &\vdash (\mathsf{cons}(a,\ell) : \mathsf{List}(A)) \end{aligned}$$

Functional programming is built on defining functions by recursion over inductive datatypes.

$$\begin{aligned} \mathsf{length}(\mathsf{nil}) &:= 0 \\ \mathsf{length}(\mathsf{cons}(\mathsf{a},\ell)) &:= s(\mathsf{length}(\ell)) \end{aligned}$$

This is defined using the eliminator for $\mathsf{List}(A)$.

Proof by induction

③ If $c_0 \colon C(0)$ and

$$(x \colon \mathbb{N}), (r \colon C(x)) \vdash (c_{\mathsf{s}} : C(\mathsf{s}(x)))$$

then for $p \colon \mathbb{N}$ we have $\mathsf{rec}(p, c_0, c_{\mathsf{s}}) : C(p)$.

When C is a predicate, this is just proof by induction.

types	$\longleftrightarrow$	propositions
programming	$\longleftrightarrow$	proving
recursion	$\longleftrightarrow$	induction

Proof by induction

3. If $c_0 \colon C(0)$ and

$$(x \colon \mathbb{N}), (r \colon C(x)) \vdash (c_{\mathsf{s}} : C(\mathsf{s}(x)))$$

then for $p \colon \mathbb{N}$ we have $\mathsf{rec}(p, c_0, c_{\mathsf{s}}) : C(p)$.

When C is a predicate, this is just proof by induction.

types ⟷ propositions

programming ⟷ proving

recursion ⟷ induction

Conclusion

Proof by induction is not something special about the natural numbers; it applies to any inductive type.

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Theorem

$0 \neq 1$.

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Theorem

$0 \neq 1$.

Proof.

Define $P \colon \mathbb{N} \to \mathrm{Type}$ by “recursion”:

$$\begin{aligned} P(0) &\coloneqq \mathbf{1} \\ P(\mathsf{s}(n)) &\coloneqq \emptyset \end{aligned}$$

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Theorem

$0 \neq 1$.

Proof.

Define $P\colon \mathbb{N} \to \mathsf{Type}$ by “recursion”:

$$\begin{aligned} P(0) &:\equiv \mathbf{1} \\ P(\mathsf{s}(n)) &:\equiv \emptyset \end{aligned}$$

- Suppose $p\colon (0 = 1)$.

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Theorem

$0 \neq 1$.

Proof.

Define $P\colon \mathbb{N} \to \mathsf{Type}$ by “recursion”:

$$\begin{aligned} P(0) &:\equiv \mathbf{1} \\ P(\mathsf{s}(n)) &:\equiv \emptyset \end{aligned}$$

- Suppose $p\colon (0 = 1)$.
- Since $\star : P(0)$, we have $\mathsf{trans}(p, \star) : P(1) \equiv \emptyset$.

Recursively defined types

We can define dependent types as Type-valued recursive functions.

Theorem

$0 \neq 1$.

Proof.

Define $P\colon \mathbb{N} \to \text{Type}$ by "recursion":

$$\begin{aligned} P(0) &:\equiv \mathbf{1} \\ P(\mathsf{s}(n)) &:\equiv \emptyset \end{aligned}$$

- Suppose $p\colon (0 = 1)$.
- Since $\star : P(0)$, we have $\mathsf{trans}(p, \star) : P(1) \equiv \emptyset$.
- Thus, $\lambda p.\mathsf{trans}(p, tt) : ((0 = 1) \to \emptyset) \equiv \neg(0 = 1)$.

□

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.
- A is $(n+1)$-truncated $\iff$ each $(x = y)$ is n-truncated.

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.
- A is $(n+1)$-truncated $\iff$ each $(x = y)$ is n-truncated.
- A is an h-set $\iff$ each $(x = y)$ is an h-prop.
  Thus, it makes sense to call h-props "(-1)-truncated".

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.
- A is $(n+1)$-truncated $\iff$ each $(x = y)$ is n-truncated.
- A is an h-set $\iff$ each $(x = y)$ is an h-prop.
  Thus, it makes sense to call h-props "(-1)-truncated".
- A is an h-prop $\iff$ each $(x = y)$ is contractible.
  Thus, we call contractible spaces "(-2)-truncated".

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.
- A is $(n+1)$-truncated $\iff$ each $(x = y)$ is n-truncated.
- A is an h-set $\iff$ each $(x = y)$ is an h-prop.
  Thus, it makes sense to call h-props “(-1)-truncated”.
- A is an h-prop $\iff$ each $(x = y)$ is contractible.
  Thus, we call contractible spaces “(-2)-truncated”.
- After this, it’s “turtles all the way down”: (-3)-truncated is the same as (-2)-truncated.

Example: Truncation

Definition

An ∞-groupoid is *n-truncated* if it has no nontrivial k-morphisms for any $k > n$.

- h-sets are 0-truncated.
- A is $(n+1)$-truncated $\iff$ each $(x = y)$ is n-truncated.
- A is an h-set $\iff$ each $(x = y)$ is an h-prop.
  Thus, it makes sense to call h-props "(-1)-truncated".
- A is an h-prop $\iff$ each $(x = y)$ is contractible.
  Thus, we call contractible spaces "(-2)-truncated".
- After this, it's "turtles all the way down": (-3)-truncated is the same as (-2)-truncated.
- (Voevodsky) h-level n means $(n-2)$-truncated.

$$\mathrm{isHlevel}(0, A) :\equiv \mathrm{isContr}(A)$$

$$\mathrm{isHlevel}(\mathsf{s}(n), A) :\equiv \prod_{x\,:\,A} \prod_{y\,:\,A} \mathrm{isHlevel}(n, (x = y))$$

Inductive families

We can define dependent types inductively as well.

Example (Vectors)

For any A there is a dependent type $\mathsf{Vec}(A)\colon \mathbb{N} \to \mathsf{Type}$, with constructors

$$\begin{aligned} &\vdash\ \mathsf{nil}\colon \mathsf{Vec}(A,0) \\ (a\colon A), (n\colon \mathbb{N}), (\ell\colon \mathsf{Vec}(A,n)) &\vdash\ (\mathsf{cons}(a,\ell) : \mathsf{Vec}(A,\mathsf{s}(n))) \end{aligned}$$

(We build the length of a list into its type.)

Inductive families

We can define dependent types inductively as well.

Example (Vectors)

For any A there is a dependent type $\mathsf{Vec}(A) \colon \mathbb{N} \to \mathsf{Type}$, with constructors

$$\begin{aligned} &\vdash \ \mathsf{nil} \colon \mathsf{Vec}(A, 0) \\ (a \colon A), (n \colon \mathbb{N}), (\ell \colon \mathsf{Vec}(A, n)) &\vdash \ (\mathsf{cons}(a, \ell) : \mathsf{Vec}(A, \mathsf{s}(n))) \end{aligned}$$

(We build the length of a list into its type.)

Example (Equality!)

For any A there is a dependent type $\mathsf{Eq}_A \colon A \times A \to \mathsf{Type}$, with constructor

$$(a \colon A) \ \vdash \ (\mathsf{refl}_a \colon Eq_A(a, a))$$

Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

Natural numbers objects

The positive type $\mathbb{N}$ should have a left universal property.

Definition

A natural numbers object is N with $0\colon 1 \to N$, $s\colon N \to N$, s.t.

- For any object X with $0_X\colon 1 \to X$ and $s_X\colon X \to X$, there is a unique $r\colon N \to X$ such that

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[dr, "0_X"'] & N \arrow[r, "s"] \arrow[d, "r"] & N \arrow[d, "r"] \\
 & X \arrow[r, "s_X"'] & X
\end{tikzcd}$$

Natural numbers objects

The positive type $\mathbb{N}$ should have a left universal property.

Definition

A natural numbers object is N with $0\colon 1 \to N$, $s\colon N \to N$, s.t.

- For any object X with $0_X\colon 1 \to X$ and $s_X\colon X \to X$, there is a unique $r\colon N \to X$ such that

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[dr, "0_X"'] & N \arrow[r, "s"] \arrow[d, "r"] & N \arrow[d, "r"] \\
 & X \arrow[r, "s_X"'] & X
\end{tikzcd}$$

= an initial object in the category of triples $(X, 1 \to X, X \to X)$.

Natural numbers objects

The positive type $\mathbb{N}$ should have a left universal property.

Definition

A natural numbers object is N with $0\colon 1 \to N$, $s\colon N \to N$, s.t.

- For any object X with $0_X\colon 1 \to X$ and $s_X\colon X \to X$, there is a unique $r\colon N \to X$ such that

$$\begin{tikzcd}
1 \arrow[r, "0"] \arrow[dr, "0_X"'] & N \arrow[r, "s"] \arrow[d, "r"] & N \arrow[d, "r"] \\
 & X \arrow[r, "s_X"'] & X
\end{tikzcd}$$

$=$ an initial object in the category of triples $(X, 1 \to X, X \to X)$.

natural numbers type $\mathbb{N}$ $\longleftrightarrow$ natural numbers object

Algebras for endofunctors

Let F be a functor from a category to itself.

Definition

An F-algebra is an object X with a morphism $x\colon F(X) \to X$.
An F-algebra map is a map $f\colon X \to Y$ such that

$$\begin{tikzcd}
F(X) \arrow[r, "F(f)"] \arrow[d, "x"'] & F(Y) \arrow[d, "y"] \\
X \arrow[r, "f"'] & Y
\end{tikzcd}$$

An initial F-algebra is an initial object in the category of F-algebras and F-algebra maps.

Inductive types and endofunctors

inductive types $\longleftrightarrow$ initial algebras for endofunctors

inductive type		endofunctor
$\mathbb{N}$	$\longleftrightarrow$	$F(X) :\equiv 1 + X$
$\mathsf{List}(A)$	$\longleftrightarrow$	$F(X) :\equiv 1 + (A \times X)$
$A + B$	$\longleftrightarrow$	$F(X) :\equiv A + B$ (a *constant* endofunctor)

Inductive types and endofunctors

inductive types $\longleftrightarrow$ initial algebras for endofunctors

inductive type		endofunctor
$\mathbb{N}$	$\longleftrightarrow$	$F(X) := 1 + X$
$\mathsf{List}(A)$	$\longleftrightarrow$	$F(X) := 1 + (A \times X)$
$A + B$	$\longleftrightarrow$	$F(X) := A + B$
		(a *constant* endofunctor)

The eliminator directly asserts only weak initiality, but using the dependent eliminator one can prove:

Theorem (Awodey–Gambino–Sojakova)

Any inductive type W is a homotopy initial F-algebra: the space of F-algebra maps $W \to X$ is contractible.

Constructing initial algebras

We also have:

Theorem

If F is an accessible endofunctor of a locally presentable category, then there exists an initial F-algebra.

Sketch of proof.

Take the colimit of the transfinite sequence

$$\emptyset \to F(\emptyset) \to F(F(\emptyset)) \to \cdots \qquad \square$$

Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

Higher inductive types

Idea

- Inductive types are a good way to build sets: we specify the elements of a set by giving constructors.
- To build an space (or ∞-groupoid), we need to specify not only elements, but paths and higher paths.

Higher inductive types

Idea

- Inductive types are a good way to build sets: we specify the elements of a set by giving constructors.
- To build an space (or ∞-groupoid), we need to specify not only elements, but paths and higher paths.
- The iterative construction of initial algebras looks a lot like the small object argument.
- Is there an analogous notion of higher inductive type that described more general cell complexes?

Higher inductive types

Idea

- Inductive types are a good way to build sets: we specify the elements of a set by giving constructors.
- To build an space (or ∞-groupoid), we need to specify not only elements, but paths and higher paths.
- The iterative construction of initial algebras looks a lot like the small object argument.
- Is there an analogous notion of higher inductive type that described more general cell complexes?

Example

The circle S^1 should be inductively defined by two constructors

$$\mathsf{base} : S^1 \qquad \text{and} \qquad \mathsf{loop} : (\mathsf{base} = \mathsf{base})$$

Can we make sense of this?

The circle (first try)

1. Formation: There is a type S^1.
2. Introduction: $\mathsf{base} : S^1$ and $\mathsf{loop} : (\mathsf{base} = \mathsf{base})$.
3. Elimination: Given $b \colon C$ and $\ell \colon (b = b)$, for any $p \colon S^1$ we have $\mathsf{match}(p, b, \ell) : C$.
4. Computation: $\mathsf{match}(\mathsf{base}, b, \ell)$ computes to b, and $\mathsf{map}(\mathsf{match}(-, b, \ell), \mathsf{loop})$ computes to ℓ.

The circle (first try)

1. **Formation:** There is a type S^1.
2. **Introduction:** $\mathsf{base} : S^1$ and $\mathsf{loop} : (\mathsf{base} = \mathsf{base})$.
3. **Elimination:** Given $b\colon C$ and $\ell\colon (b = b)$, for any $p\colon S^1$ we have $\mathsf{match}(p, b, \ell) : C$.
4. **Computation:** $\mathsf{match}(\mathsf{base}, b, \ell)$ computes to b, and $\mathsf{map}(\mathsf{match}(-, b, \ell), \mathsf{loop})$ computes to ℓ.

What about a dependent eliminator?

Dependent loops

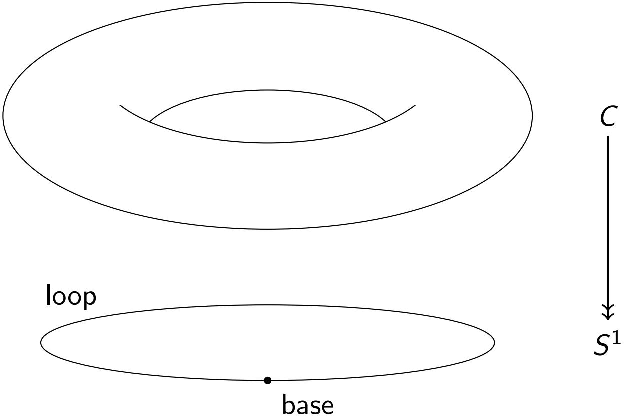
As hypotheses of the dependent eliminator for S^1, we need

1. A point $b\colon C(\mathsf{base})$.
2. A path ℓ from b to b lying over "loop".

Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

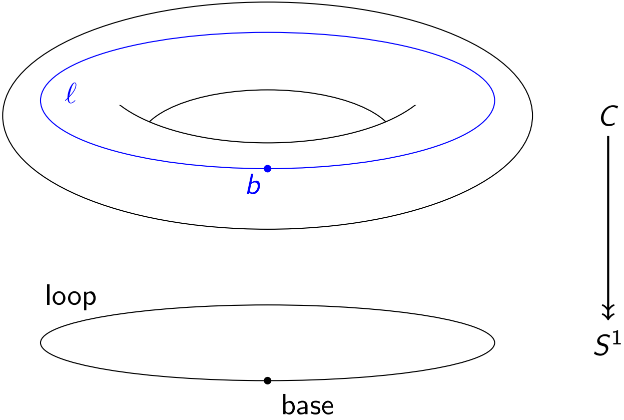
1. A point $b \colon C(\text{base})$.
2. A path ℓ from b to b lying over "loop".



Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

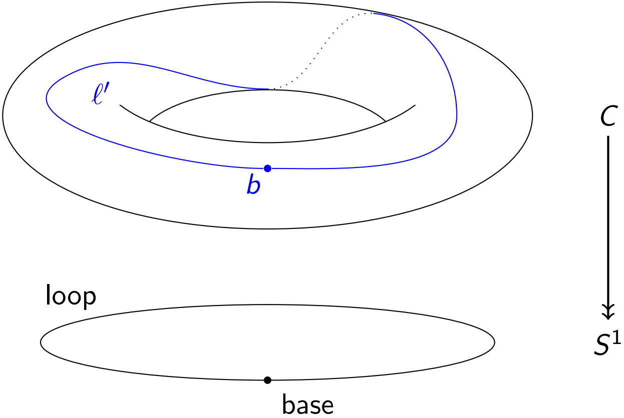
1. A point $b\colon C(\text{base})$.
2. A path ℓ from b to b lying over "loop".



Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

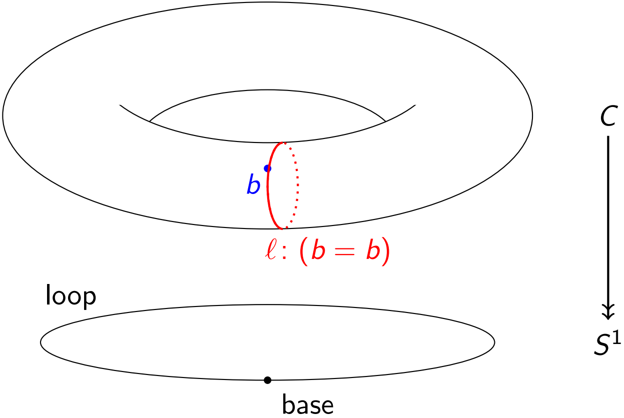
1. A point $b\colon C(\text{base})$.
2. A path ℓ from b to b lying over "loop".



Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

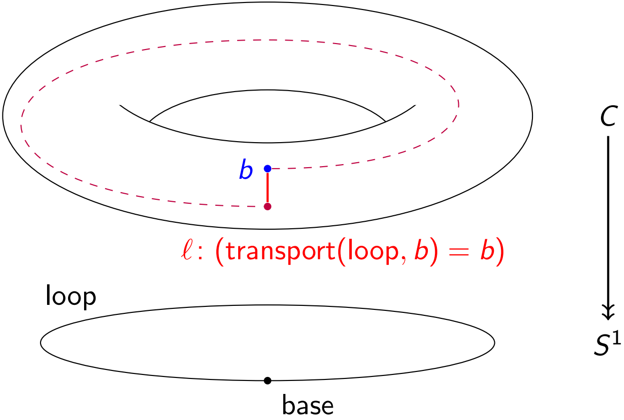
1. A point $b\colon C(\mathsf{base})$.
2. A path ℓ from b to b lying over "loop".



Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

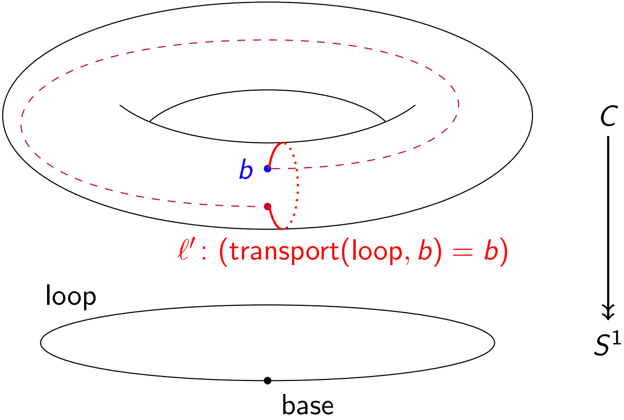
1. A point $b\colon C(\mathsf{base})$.
2. A path ℓ from b to b lying over "loop".



Dependent loops

As hypotheses of the dependent eliminator for S^1, we need

1. A point $b\colon C(\text{base})$.
2. A path ℓ from b to b lying over "loop".



The circle (final version)

1. Formation: There is a type S^1.
2. Introduction: $\mathsf{base} : S^1$ and $\mathsf{loop} : (\mathsf{base} = \mathsf{base})$.
3. Elimination: Given $b\colon C(\mathsf{base})$ and $\ell\colon (\mathsf{trans}(\mathsf{loop}, b) = b)$, for any $p\colon S^1$ we have $\mathsf{match}(p, b, \ell) : C(p)$.
4. Computation: $\mathsf{match}(\mathsf{base}, b, \ell)$ computes to b, and $\mathsf{map}(\mathsf{match}(-, b, \ell), \mathsf{loop})$ computes to ℓ.

The Interval

Example

The interval I is an inductive type with three constructors:

$$\mathsf{zero} : \mathrm{I} \qquad \mathsf{one} : \mathrm{I} \qquad \mathsf{segment} : (\mathsf{zero} = \mathsf{one})$$

The Interval

Example

The interval I is an inductive type with three constructors:

$$\mathsf{zero} : \mathrm{I} \qquad \mathsf{one} : \mathrm{I} \qquad \mathsf{segment} : (\mathsf{zero} = \mathsf{one})$$

- Unsurprisingly, this type is provably contractible.
- But surprisingly, it is not useless; it implies function extensionality.

The 2-sphere

Example

The 2-sphere S^2 has two constructors:

$$\mathsf{base2} : S^2 \qquad \mathsf{loop2} : (\mathsf{refl}_{\mathsf{base2}} = \mathsf{refl}_{\mathsf{base2}})$$

The 2-sphere

Example

The 2-sphere S^2 has two constructors:

$$\mathsf{base2} : S^2 \qquad \mathsf{loop2} : (\mathsf{refl}_{\mathsf{base2}} = \mathsf{refl}_{\mathsf{base2}})$$

OR:

$$\begin{aligned} \mathsf{northpole} &: S^2 \\ \mathsf{southpole} &: S^2 \\ \mathsf{greenwich} &: (\mathsf{northpole} = \mathsf{southpole}) \\ \mathsf{dateline} &: (\mathsf{northpole} = \mathsf{southpole}) \\ \mathsf{east} &: (\mathsf{greenwich} = \mathsf{dateline}) \\ \mathsf{west} &: (\mathsf{greenwich} = \mathsf{dateline}) \end{aligned}$$

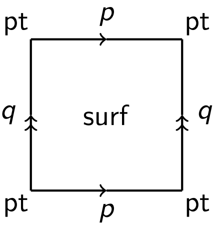
etc. . .

The torus

Example

The torus T^2 has four constructors:

$$
\begin{aligned}
\mathrm{pt} &: T^2 \\
p &: (\mathrm{pt} = \mathrm{pt}) \\
q &: (\mathrm{pt} = \mathrm{pt}) \\
\mathrm{surf} &: (p * q = q * p)
\end{aligned}
$$



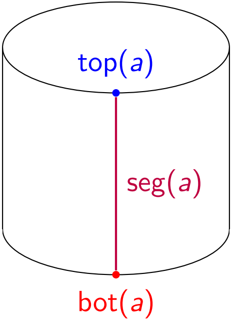
Cylinders

Example

The cylinder $\mathsf{Cyl}(A)$ on A has three constructors:

$$(a\colon A) \vdash (\mathsf{top}(a) : \mathsf{Cyl}(A)) \qquad (a\colon A) \vdash (\mathsf{bot}(a) : \mathsf{Cyl}(A))$$

$$(a\colon A) \vdash (\mathsf{seg}(a) : (\mathsf{top}(a) = \mathsf{bot}(a)))$$

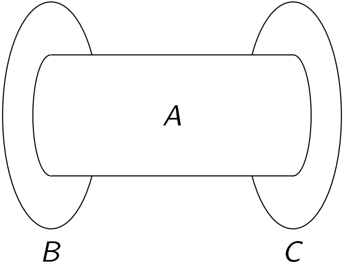


Homotopy pushouts

Example

The homotopy pushout of $f\colon A \to B$ and $g\colon A \to C$ has three constructors:

$$
\begin{aligned}
(b\colon B) &\;\vdash\; (\mathrm{left}(b) : \mathrm{pushout}(f,g))\\
(c\colon C) &\;\vdash\; (\mathrm{right}(c) : \mathrm{pushout}(f,g))\\
(a\colon A) &\;\vdash\; (\mathrm{glue}(a) : (\mathrm{left}(f(a)) = \mathrm{right}(g(a))))
\end{aligned}
$$

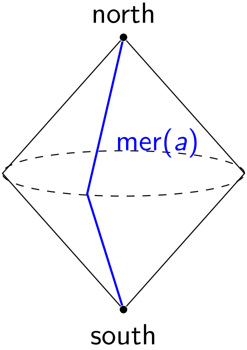


Suspension

Example

The suspension ΣA of A has three constructors:

$$\mathsf{north} : \Sigma A \qquad \mathsf{south} : \Sigma A$$
$$(a\colon A) \vdash (\mathsf{mer}(a) : (\mathsf{north} = \mathsf{south}))$$

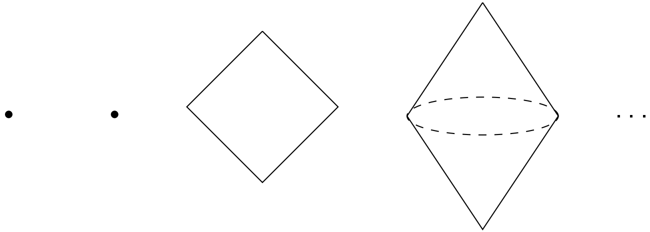


Higher spheres

Example

The *n*-sphere S^n is defined by recursion on *n*:

$$S^0 := 1 + 1$$
$$S^{\mathsf{s}(n)} := \Sigma(S^n)$$



Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

Nontriviality

Theorem

The type S^1 is contractible $\iff$ all types are h-sets.

Proof.

Easy; S^1 is the "universal loop". ☐

Nontriviality

Theorem

The type S^1 is contractible $\iff$ all types are h-sets.

Proof.

Easy; S^1 is the "universal loop". ☐

HITs by themselves don't guarantee the homotopy theory is nontrivial. We need something else, like univalence.

$\pi_1(S^1) \cong \mathbb{Z}$, classically

$$\pi_1(S^1) \cong \mathbb{Z}$$

How do we prove this classically?

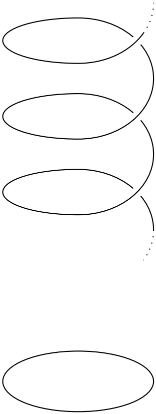
$\pi_1(S^1) \cong \mathbb{Z}$, classically

$$\pi_1(S^1) \cong \mathbb{Z}$$

How do we prove this classically?

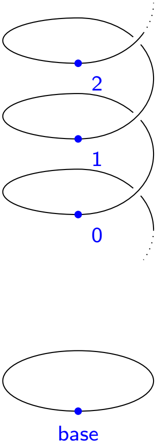
1. Consider the winding map $\mathbb{R} \to S^1$.
2. This is the universal cover of S^1.
3. Thus, its fiber over a point, namely $\mathbb{Z}$, is $\pi_1(S^1)$.

The universal cover of S^1



$$\begin{tikzcd} \mathbb{R} \arrow[d, two heads] \\ S^1 \end{tikzcd}$$

The universal cover of S^1



$$\begin{tikzcd}\mathbb{R} \arrow[d, two heads] \\ S^1\end{tikzcd}$$

$\pi_1(S^1) \cong \mathbb{Z}$, homotopically

$$\pi_1(S^1) \cong \mathbb{Z}$$

A more homotopy-theoretic way to phrase the classical proof:

1. We have a fibration $\mathbb{R} \to S^1$ with fiber $\mathbb{Z}$.

$\pi_1(S^1) \cong \mathbb{Z}$, homotopically

$$\pi_1(S^1) \cong \mathbb{Z}$$

A more homotopy-theoretic way to phrase the classical proof:

1. We have a fibration $\mathbb{R} \to S^1$ with fiber $\mathbb{Z}$.
2. We have a map $* \to S^1$, whose homotopy fiber is ΩS^1.

$\pi_1(S^1) \cong \mathbb{Z}$, homotopically

$\pi_1(S^1) \cong \mathbb{Z}$

A more homotopy-theoretic way to phrase the classical proof:

1. We have a fibration $\mathbb{R} \to S^1$ with fiber $\mathbb{Z}$.
2. We have a map $* \to S^1$, whose homotopy fiber is ΩS^1.
3. $\mathbb{R}$ is contractible, so we have an equivalence $* \simeq \mathbb{R}$ over S^1. By the short five lemma, the induced map on homotopy fibers is an equivalence.

$$\begin{tikzcd}
\Omega S^1 \arrow[r] \arrow[d, "\sim"'] & * \arrow[r] \arrow[d, "\sim"'] & S^1 \arrow[d, equal] \\
\mathbb{Z} \arrow[r] & \mathbb{R} \arrow[r] & S^1
\end{tikzcd}$$

$\pi_1(S^1) \cong \mathbb{Z}$, homotopically

$$\pi_1(S^1) \cong \mathbb{Z}$$

A more homotopy-theoretic way to phrase the classical proof:

1. We have a fibration $\mathbb{R} \to S^1$ with fiber $\mathbb{Z}$.
2. We have a map $* \to S^1$, whose homotopy fiber is ΩS^1.
3. $\mathbb{R}$ is contractible, so we have an equivalence $* \simeq \mathbb{R}$ over S^1. By the short five lemma, the induced map on homotopy fibers is an equivalence.

$$\begin{tikzcd}
\Omega S^1 \arrow[r] \arrow[d, "\sim"'] & * \arrow[r] \arrow[d, "\sim"'] & S^1 \arrow[d, equal] \\
\mathbb{Z} \arrow[r] & \mathbb{R} \arrow[r] & S^1
\end{tikzcd}$$

4. In particular, $\pi_1(S^1) \cong \mathbb{Z}$.

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R\colon S^1 \to \mathsf{Type}$.

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R\colon S^1 \to \mathsf{Type}$.
- By the eliminator for S^1, a function $R\colon S^1 \to \mathsf{Type}$ is determined by
  - A point $B\colon \mathsf{Type}$ and
  - A path $\ell\colon (B = B)$.

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R \colon S^1 \to \mathsf{Type}$.
- By the eliminator for S^1, a function $R \colon S^1 \to \mathsf{Type}$ is determined by
  - A point $B \colon \mathsf{Type}$ and
  - A path $\ell \colon (B = B)$.
- By univalence, ℓ is an equivalence $B \simeq B$.

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R \colon S^1 \to \mathsf{Type}$.
- By the eliminator for S^1, a function $R \colon S^1 \to \mathsf{Type}$ is determined by
  - A point $B \colon \mathsf{Type}$ and
  - A path $\ell \colon (B = B)$.
- By univalence, ℓ is an equivalence $B \simeq B$.

Thus we can take $B = \mathbb{Z}$ and ℓ to be "$+1$".

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R \colon S^1 \to \mathsf{Type}$.
- By the eliminator for S^1, a function $R \colon S^1 \to \mathsf{Type}$ is determined by
  - A point $B \colon \mathsf{Type}$ and
  - A path $\ell \colon (B = B)$.
- By univalence, ℓ is an equivalence $B \simeq B$.

Thus we can take $B = \mathbb{Z}$ and ℓ to be "$+1$".

- All that's left to do is prove that $\sum_{x \colon S^1} R(x)$ is contractible. We can do this by "induction" on S^1.

$\pi_1(S^1) \cong \mathbb{Z}$, type-theoretically

How can we build the fibration $\mathbb{R} \twoheadrightarrow S^1$ in type theory?

- A fibration over S^1 is a dependent type $R \colon S^1 \to \mathsf{Type}$.
- By the eliminator for S^1, a function $R \colon S^1 \to \mathsf{Type}$ is determined by
  - A point $B \colon \mathsf{Type}$ and
  - A path $\ell \colon (B = B)$.
- By univalence, ℓ is an equivalence $B \simeq B$.

Thus we can take $B = \mathbb{Z}$ and ℓ to be "$+1$".

- All that's left to do is prove that $\sum_{x \colon S^1} R(x)$ is contractible. We can do this by "induction" on S^1.
- What we get is $\Omega S^1 \cong \mathbb{Z}$, which is classically stronger than $\pi_1(S^1) \cong \mathbb{Z}$. Here, we don't yet have a definition of π_1.

Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

Supports

Recall: A is (-1)-truncated, or an h-prop, if

$$\prod_{x,y\,:\,A} (x = y).$$

The support of A, denoted $\mathrm{supp}(A)$, is supposed to be:

- an h-prop that contains a point precisely when A does.
- a reflection of A into h-props.

Support as an HIT

Definition (Lumsdaine)

The support of A is inductively defined by two constructors:

$$\begin{aligned} (a\colon A) &\vdash (\mathsf{inhab}(a) : \mathsf{supp}(A)) \\ (x\colon \mathsf{supp}(A)), (y\colon \mathsf{supp}(A)) &\vdash (\mathsf{inpath}(x,y) : (x = y)) \end{aligned}$$

Support as an HIT

Definition (Lumsdaine)

The support of A is inductively defined by two constructors:

$$\begin{aligned} (a\colon A) &\vdash (\mathsf{inhab}(a) : \mathsf{supp}(A)) \\ (x\colon \mathsf{supp}(A)), (y\colon \mathsf{supp}(A)) &\vdash (\mathsf{inpath}(x,y) : (x = y)) \end{aligned}$$

The type of inpath is precisely $\mathsf{isProp}(\mathsf{supp}(A))$!

Support as an HIT

Definition (Lumsdaine)

The support of A is inductively defined by two constructors:

$$\begin{aligned} (a\colon A) &\vdash (\mathrm{inhab}(a) : \mathrm{supp}(A)) \\ (x\colon \mathrm{supp}(A)), (y\colon \mathrm{supp}(A)) &\vdash (\mathrm{inpath}(x,y) : (x = y)) \end{aligned}$$

The type of inpath is precisely $\mathrm{isProp}(\mathrm{supp}(A))$!

3. if $(x\colon A) \vdash (c_A : C)$ and $(z, w\colon C) \vdash (c_= : (z = w))$, for any $p\colon \mathrm{supp}(A)$ we have $\mathrm{match}(p, c_A, c_=) : C$.

Support as an HIT

Definition (Lumsdaine)

The support of A is inductively defined by two constructors:

$$\begin{aligned} (a\colon A) &\vdash (\mathrm{inhab}(a) : \mathrm{supp}(A)) \\ (x\colon \mathrm{supp}(A)), (y\colon \mathrm{supp}(A)) &\vdash (\mathrm{inpath}(x, y) : (x = y)) \end{aligned}$$

The type of inpath is precisely isProp(supp(A))!

③ if $(x\colon A) \vdash (c_A : C)$ and $(z, w\colon C) \vdash (c_= : (z = w))$, for any $p\colon \mathrm{supp}(A)$ we have $\mathrm{match}(p, c_A, c_=) : C$.

The hypotheses of the eliminator say exactly that C is an h-prop and we have a map $A \to C$.

$$\begin{tikzcd} A \arrow[d, "\mathrm{inhab}"'] \arrow[dr, "c_A"] & \\ \mathrm{supp}(A) \arrow[r, dotted] & C \end{tikzcd}$$

The rest of logic

$$\begin{array}{rcl} P \text{ and } Q & \longleftrightarrow & P \times Q \\ P \text{ implies } Q & \longleftrightarrow & Q^P \\ \top \text{ (true)} & \longleftrightarrow & \mathbf{1} \\ \bot \text{ (false)} & \longleftrightarrow & \emptyset \\ (\forall x \colon A) P(x) & \longleftrightarrow & \prod_{x \colon A} B(x) \end{array}$$

$$\begin{array}{rcl} P \text{ or } Q & \longleftrightarrow & \mathrm{supp}(P + Q) \\ (\exists x \colon A) P(x) & \longleftrightarrow & \mathrm{supp}(\sum_{x \colon A} B(x)) \end{array}$$

The magic of supp

Note: our ability to define “isProp” without using “supp” was crucial to our ability to define “supp” itself!

- Because we defined isProp using only paths, path-constructors can “universally force” a type to be an h-prop.
- Because isProp is an h-prop, these path-constructors have no other effect (give no extra data).

0-truncation

Example

The 0-truncation $\pi_0(A)$ has two constructors:

$$\begin{aligned} (a \colon A) &\vdash (\mathrm{cpnt}(a) : \pi_0(A)) \\ (x, y : \pi_0(A)), (p, q : (x = y)) &\vdash (\mathrm{pp}(x, y, p, q) : (p = q)) \end{aligned}$$

0-truncation

Example

The 0-truncation $\pi_0(A)$ has two constructors:

$$\begin{aligned}(a\colon A) &\vdash (\mathsf{cpnt}(a) : \pi_0(A)) \\ (x, y : \pi_0(A)), (p, q : (x = y)) &\vdash (\mathsf{pp}(x, y, p, q) : (p = q))\end{aligned}$$

- The type of pp is precisely $\mathsf{isHlevel}(2, A)$.
- The eliminator says that $\pi_0(A)$ is a reflection of A into h-sets.

0-truncation

Example

The 0-truncation $\pi_0(A)$ has two constructors:

$$\begin{aligned} (a\colon A) &\vdash (\mathrm{cpnt}(a) : \pi_0(A)) \\ (x, y : \pi_0(A)), (p, q : (x = y)) &\vdash (\mathrm{pp}(x, y, p, q) : (p = q)) \end{aligned}$$

- The type of pp is precisely $\mathrm{isHlevel}(2, A)$.
- The eliminator says that $\pi_0(A)$ is a reflection of A into h-sets.

Now we can define

$$\pi_1(A) \coloneqq \pi_0(\Omega A)$$

etc. . . .

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f\colon A \to B$ which induces $\pi_n(A) \xrightarrow{\simeq} \pi_n(B)$ for all $n\colon \mathbb{N}$ is not necessarily an equivalence!

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f\colon A \to B$ which induces $\pi_n(A) \xrightarrow{\simeq} \pi_n(B)$ for all $n\colon \mathbb{N}$ is not necessarily an equivalence!
   - Not closely related to non-CW-complex spaces.
   - It has to do with non-hypercomplete $(\infty, 1)$-toposes.
   - A reason not to call “equivalences” “weak equivalences”.

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f : A \to B$ which induces $\pi_n(A) \xrightarrow{\sim} \pi_n(B)$ for all $n : \mathbb{N}$ is not necessarily an equivalence!
    - Not closely related to non-CW-complex spaces.
    - It has to do with non-hypercomplete $(\infty, 1)$-toposes.
    - A reason not to call "equivalences" "weak equivalences".
2. There may be types which do not admit a connected map from an h-set!
    - This happens in $\infty\mathrm{Gpd}/X$ if X is not discrete.

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f\colon A \to B$ which induces $\pi_n(A) \xrightarrow{\sim} \pi_n(B)$ for all $n\colon \mathbb{N}$ is not necessarily an equivalence!
    - Not closely related to non-CW-complex spaces.
    - It has to do with non-hypercomplete $(\infty, 1)$-toposes.
    - A reason not to call "equivalences" "weak equivalences".
2. There may be types which do not admit a connected map from an h-set!
    - This happens in $\infty\mathrm{Gpd}/X$ if X is not discrete.
    - As a foundation, not every ∞-groupoid has an "underlying set" of objects (though it does have a π_0).

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f\colon A \to B$ which induces $\pi_n(A) \xrightarrow{\sim} \pi_n(B)$ for all $n\colon \mathbb{N}$ is not necessarily an equivalence!
    - Not closely related to non-CW-complex spaces.
    - It has to do with non-hypercomplete $(\infty, 1)$-toposes.
    - A reason not to call "equivalences" "weak equivalences".
2. There may be types which do not admit a connected map from an h-set!
    - This happens in $\infty\mathrm{Gpd}/X$ if X is not discrete.
    - As a foundation, not every ∞-groupoid has an "underlying set" of objects (though it does have a π_0).
    - In particular, not every type has a cell decomposition.

Nonclassicality

Remark

h-sets and homotopy groups are a bit surprising.

1. A map $f\colon A \to B$ which induces $\pi_n(A) \xrightarrow{\sim} \pi_n(B)$ for all $n\colon \mathbb{N}$ is not necessarily an equivalence!
   - Not closely related to non-CW-complex spaces.
   - It has to do with non-hypercomplete $(\infty, 1)$-toposes.
   - A reason not to call "equivalences" "weak equivalences".
2. There may be types which do not admit a connected map from an h-set!
   - This happens in $\infty\mathrm{Gpd}/X$ if X is not discrete.
   - As a foundation, not every ∞-groupoid has an "underlying set" of objects (though it does have a π_0).
   - In particular, not every type has a cell decomposition.

These are "classicality properties" of $\infty\mathrm{Gpd}$, like excluded middle and the axiom of choice in Set.

Localization

Given $f\colon A \to B$.

Definition

- Z is f-local if $Z^B \xrightarrow{-\circ f} Z^A$ is an equivalence.
- An f-localization of X is a reflection of X into f-local spaces.

Localization

Given $f\colon A \to B$.

Definition

- Z is f-local if $Z^B \xrightarrow{-\circ f} Z^A$ is an equivalence.
- An f-localization of X is a reflection of X into f-local spaces.

Examples

- If f is $S^n \to D^{n+1}$, then f-local means $(n-1)$-truncated.
- Localization and completion at primes.
- Construction of $(\infty, 1)$-toposes from $(\infty, 1)$-presheaves.
- $\ldots$

h-isomorphisms

Recall: $f\colon A \to B$ is an h-isomorphism if we have

- A map $g\colon B \to A$
- A homotopy $r\colon \prod_{a\colon A}(g(f(a)) = a)$
- A map $h\colon B \to A$
- A homotopy $s\colon \prod_{b\colon B}(f(g(b)) = b)$

The type isHiso(f) is an h-prop, equivalent to isEquiv(f).

Localization as a HIT

Definition

Given $f\colon A \to B$ and X, the localization $L_f X$ has constructors:

$$\begin{aligned}
(x\colon X) &\vdash (\mathsf{tolocal}(x) : L_f X)\\
(g\colon A \to L_f X), (b\colon B) &\vdash (\mathsf{lsec}(g, b) : L_f X)\\
(g\colon A \to L_f X), (a\colon A) &\vdash (\mathsf{lsech}(g, a) : (\mathsf{lsec}(g, f(a)) = g(a)))\\
(g\colon A \to L_f X), (b\colon B) &\vdash (\mathsf{lret}(g, b) : L_f X)\\
(h\colon B \to L_f X), (b\colon B) &\vdash (\mathsf{lreth}(h, b) : (\mathsf{lret}(h \circ f, b) = h(b))
\end{aligned}$$

The meaning of localization

- Of course, tolocal is a map $X \to L_f X$.
- lsec is a map $(L_f X)^A \to (L_f X)^B$.
- lsech is a homotopy from $(L_f X)^A \xrightarrow{\text{lsec}} (L_f X)^B \xrightarrow{- \circ f} (L_f X)^A$ to the identity.

The meaning of localization

- Of course, tolocal is a map $X \to L_f X$.
- lsec is a map $(L_f X)^A \to (L_f X)^B$.
- lsech is a homotopy from $(L_f X)^A \xrightarrow{\text{lsec}} (L_f X)^B \xrightarrow{-\circ f} (L_f X)^A$ to the identity.
- lret is a map $(L_f X)^A \to (L_f X)^B$.
- lreth is a homotopy from $(L_f X)^B \xrightarrow{-\circ f} (L_f X)^A \xrightarrow{\text{lret}} (L_f X)^B$ to the identity.

Together, (lsec, lsech, lret, lreth) exactly inhabit "isHiso($- \circ f$)", i.e. "isLocal(f, X)".

Thus, $L_f X$ is an f-localization of X.

Outline

1. Inductive types
2. Inductive types and initial algebras
3. Higher inductive types
4. Computing with HITs
5. Properly recursive HITs
6. Cofibrations and model structures

The other factorization

Recall:

- A model category has two weak factorization systems:

  (acyclic cofibrations, fibrations)
  (cofibrations, acyclic fibrations)

- Identity types correspond to the first WFS, using the mapping path space:

$$A \to \big[y\colon B,\, x\colon A,\, p\colon (g(x)=y)\big] \twoheadrightarrow B$$

- In topology, the second WFS is likewise related to the mapping cylinder.

$$A \to Mf \twoheadrightarrow B$$

  Can we use HITs to construct this?

Acyclic fibrations

What is an acyclic fibration in type theory?

1. A fibration that is also an equivalence.
2. A fibration $p\colon B \twoheadrightarrow A$ which admits a section $s\colon A \to B$ (hence $ps = 1_A$) such that $sp \sim 1_B$.
3. A dependent type $B\colon A \to \mathrm{Type}$ such that each $B(a)$ is contractible.

Cofibrations

What is a cofibration in type theory?

Cofibrations

What is a cofibration in type theory?

Actually, what is an acyclic cofibration in type theory?
I.e. when does $i\colon A \to B$ satisfy $i \boxslash p$ for any fibration p?

Acyclic cofibrations

Theorem (Gambino-Garner)

If B is an inductive type and i is its only constructor, then $i \boxslash p$ for any fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Acyclic cofibrations

Theorem (Gambino-Garner)

If B is an inductive type and i is its only constructor, then $i \boxslash p$ for any fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Proof.

- p is a dependent type $Y \colon X \to \text{Type}$; we want to define

$$h \colon \prod_{b \colon B} Y(g(b))$$

Acyclic cofibrations

Theorem (Gambino-Garner)

If B is an inductive type and i is its only constructor, then $i \boxslash p$ for any fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Proof.

- p is a dependent type $Y : X \to \text{Type}$; we want to define

$$h : \prod_{b : B} Y(g(b))$$

- By the eliminator, it suffices to specify $h(b)$ when $b = i(a)$.
- But then we can take $h(i(a)) :\equiv f(a)$. □

Path object factorizations

Example

refl: $A \to \mathsf{Id}_A$ is the only constructor of the identity type. Thus,

$$A \xrightarrow{\mathsf{refl}} \mathsf{Id}_A \twoheadrightarrow A \times A$$

is an (acyclic cofibration, fibration) factorization.

Some cofibrations

Theorem

If B is an inductive type and $i \colon A \to B$ is one of its constructors, then $i \boxslash p$ for any acyclic fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Some cofibrations

Theorem

If B is an inductive type and $i\colon A \to B$ is one of its constructors, then $i \boxslash p$ for any acyclic fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Proof.

- Now we have a section $s\colon \prod_{x\colon X} Y(x)$.
- We define $h\colon \prod_{b\colon B} Y(g(b))$ with the eliminator of B:
    - If $b = i(a)$, take $h(b) :\equiv f(a)$.
    - If b is some other constructor, take $h(b) :\equiv s(g(b))$. ☐

Some cofibrations

Theorem

If B is an inductive type and $i\colon A \to B$ is one of its constructors, then $i \boxslash p$ for any fibration p with a section.

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\ B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X \end{tikzcd}$$

Proof.

- Now we have a section $s\colon \prod_{x\colon X} Y(x)$.
- We define $h\colon \prod_{b\colon B} Y(g(b))$ with the eliminator of B:
  - If $b = i(a)$, take $h(b) :\equiv f(a)$.
  - If b is some other constructor, take $h(b) :\equiv s(g(b))$. $\square$

More cofibrations

Theorem

If B is a higher inductive type and $i \colon A \to B$ is one of its point-constructors, then $i \boxslash p$ for any acyclic fibration p.

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "i"'] & Y \arrow[d, two heads, "p"] \\
B \arrow[r, "g"'] \arrow[ur, dotted, "?"] & X
\end{tikzcd}$$

Proof.

- Now we have a section $s \colon \prod_{x : x} Y(x)$.
- We define $h \colon \prod_{b : B} Y(g(b))$ with the eliminator of B:
  - If $b = i(a)$, take $h(b) \coloneqq f(a)$.
  - If b is some other point-constructor, take $h(b) \coloneqq s(g(b))$.
  - In the case of path-constructors, use the contractibility of the fibers of p. □

The other factorization

Need a mapping cylinder for $f\colon A \to B$ that is dependent over B.

Definition

The mapping cylinder $Mf\colon B \to \mathrm{Type}$ has three constructors:

$$\begin{aligned}(b\colon B) &\vdash (\mathrm{right}(b) : Mf(b))\\ (a\colon A) &\vdash (\mathrm{left}(a) : Mf(f(a)))\\ (a\colon A) &\vdash (\mathrm{glue}(a) : (\mathrm{left}(a) = \mathrm{right}(f(a))))\end{aligned}$$

The other factorization

Need a mapping cylinder for $f\colon A \to B$ that is dependent over B.

Definition

The mapping cylinder $Mf\colon B \to \mathrm{Type}$ has three constructors:

$$\begin{aligned}(b\colon B) &\vdash (\mathrm{right}(b) : Mf(b))\\(a\colon A) &\vdash (\mathrm{left}(a) : Mf(f(a)))\\(a\colon A) &\vdash (\mathrm{glue}(a) : (\mathrm{left}(a) = \mathrm{right}(f(a))))\end{aligned}$$

Theorem (Lumsdaine)

- *This defines a WFS (cofibrations, acyclic fibrations).*
- *With the other WFS, and the type-theoretic equivalences, we have a model category (except for strict limits and colimits).*

Categorical models

Conversely:

Theorem (Lumsdaine–Shulman)

A well-behaved combinatorial model category which models type theory as before (lccc etc.) also models all higher inductive types.

(In particular, simplicial sets.)

Very rough sketch of proof.

Combine the transfinite construction of initial algebras with the homotopy-theoretic small object argument. ☐

Elementary $(\infty, 1)$-toposes

Proposal

An elementary $(\infty, 1)$-topos is an $(\infty, 1)$-category $\mathcal{C}$ such that:

1. $\mathcal{C}$ has finite limits.
2. $\mathcal{C}$ is locally cartesian closed.
3. $\mathcal{C}$ has sufficiently many object classifiers.
4. $\mathcal{C}$ has sufficently many "higher initial algebras" ($\Rightarrow \mathcal{C}$ has finite colimits).

Conjecture

Any elementary $(\infty, 1)$-topos has an internal homotopy type theory modeling the univalence axiom and higher inductive types.