Cubical Homotopy Type Theory

Steve Awodey
Carnegie Mellon University

Logic Colloquium and CLMPS
August 2015

Outline

I. Basic ideas of Homotopy Type Theory.
II. The univalence axiom.
III. Higher inductive types.
IV. Synthetic reasoning and formalization.
V. Recent work in progress: Cubical HoTT.

Overview of I. – III.

- Homotopy Type Theory is a recently discovered connection between Logic and Topology.
- It is based on an interpretation of intensional Martin-Löf type theory into homotopy theory.
- The homotopy interpretation suggests some new logical principles: Univalence and Higher Inductive Types.
- The univalence axiom implies that "isomorphic structures are equal". This makes the system more powerful logically, and also more interesting philosophically.
- Higher inductive types are used to introduce some basic spaces like the spheres S^n and some important constructions such as quotients $X/\sim$.

Type theory

Martin-Löf constructive type theory consists of:

- **Types**: $X, Y, \ldots, A \times B,\ A \to B, \ldots$
- **Terms**: $x : A,\ b : B,\ \langle a, b \rangle,\ \lambda x.b(x), \ldots$
- **Dependent Types**: $x : A \vdash B(x)$
  - $\sum_{x:A} B(x)$
  - $\prod_{x:A} B(x)$
- **Equations** $s = t : A$

Formal calculus of typed terms and equations, presented as a deductive system by rules of inference.

Intended as a foundation for constructive mathematics, but now used also in programming languages and computerized proof assistants.

Propositions as Types

The system has a dual interpretation:

- once as **mathematical** objects: types are "sets" and their terms are "elements", which are being constructed,
- once as **logical** objects: types are "propositions" and their terms are "proofs", which are being derived.

This is known as the **Curry-Howard correspondence**:

0	1	$A + B$	$A \times B$	$A \to B$	$\sum_{x:A} B(x)$	$\prod_{x:A} B(x)$
$\bot$	T	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$\exists_{x:A} B(x)$	$\forall_{x:A} B(x)$

Gives the system its **constructive character**.

Identity types

It's natural to add a primitive **identity type** between terms:

$$\mathtt{Id}_A(x, y)\,.$$

Logically this is just the proposition "x equals y".

0	1	$A + B$	$A \times B$	$A \to B$	$\sum_{x:A} B(x)$	$\prod_{x:A} B(x)$	$\mathtt{Id}_A(x, y)$
$\bot$	T	$A \vee B$	$A \wedge B$	$A \Rightarrow B$	$\exists_{x:A} B(x)$	$\forall_{x:A} B(x)$	$\mathrm{x} = \mathrm{y}$

But what is it **mathematically**? What are the terms $p : \mathtt{Id}_A(x, y)$ of these new types ? Can **they** differ?

This type introduces a rich new structure into the type theory and gives rise to the new interpretation.

The homotopy interpretation (Awodey-Warren)

Suppose we have terms of ascending identity types:

$$\begin{aligned} a,\ b &: A \\ p,\ q &: \mathtt{Id}_A(a,b) \\ \alpha,\ \beta &: \mathtt{Id}_{\mathtt{Id}_A(a,b)}(p,q) \\ \ldots &: \mathtt{Id}_{\mathtt{Id}_{\mathtt{Id}_{\ldots}}}(\ldots) \end{aligned}$$

Consider the following interpretation:

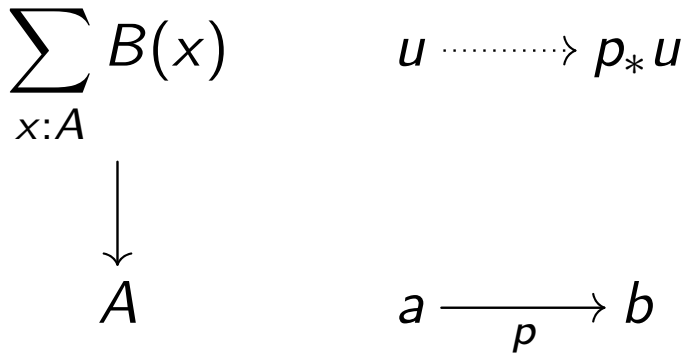
$$\begin{aligned} \text{Types} &\rightsquigarrow \text{Spaces} \\ \text{Terms} &\rightsquigarrow \text{Maps} \\ a : A &\rightsquigarrow \text{Points } a : 1 \to A \\ p : \mathtt{Id}_A(a,b) &\rightsquigarrow \text{Paths } p : a \Rightarrow b \\ \alpha : \mathtt{Id}_{\mathtt{Id}_A(a,b)}(p,q) &\rightsquigarrow \text{Homotopies } \alpha : p \Rrightarrow q \\ &\vdots \end{aligned}$$

The homotopy interpretation: Type dependency

What about dependent types $x : A \vdash B(x)$?
The identity rules imply the following:

$$\frac{p : \mathtt{Id}_A(a,b) \qquad u : B(a)}{\vdots \qquad\qquad \vdots}$$
$$p_* u : B(b)$$

Logically, this just says "$a = b$ & $B(a) \Rightarrow B(b)$".
But topologically, it is the familiar **lifting property**:



This is the notion of a **fibration** of spaces.

The homotopy interpretation: Summary

We take the **topological interpretation** of the *simply-typed* λ-calculus:

$$\text{types} \rightsquigarrow \text{spaces}$$
$$\text{terms} \rightsquigarrow \text{continuous functions}$$

and extend it to *dependently typed* λ-calculus with $\mathtt{Id}$-types, via the **basic idea**:

$$p : \mathtt{Id}_X(a, b) \;\rightsquigarrow\; p \text{ is a path from point } a \text{ to point } b \text{ in } X$$

This forces:

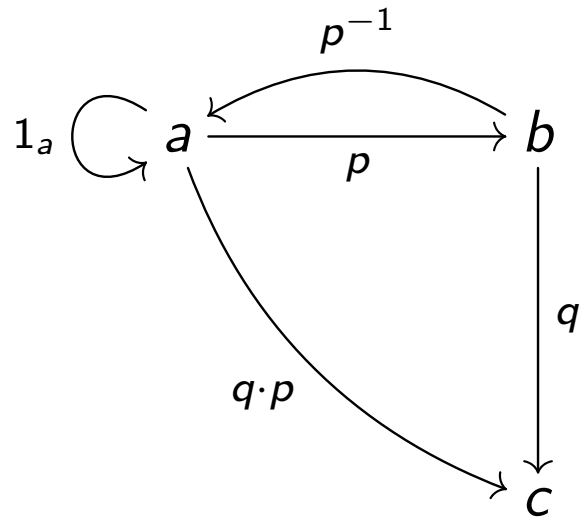
- *dependent types to be fibrations*,
- $\mathtt{Id}$*-types to be path spaces*,
- *homotopic maps to be identical*.

The fundamental groupoid of a type (Hofmann-Streicher)

In topology, the points and paths in any space bear the structure of a **groupoid**.

Definition

A *groupoid* is a category in which every arrow has an inverse.



In the same way the **terms** $a, b, c : X$ and **identity terms** $p : \mathtt{Id}_X(a, b)$ and $q : \mathtt{Id}_X(b, c)$ of any type X also form a groupoid.

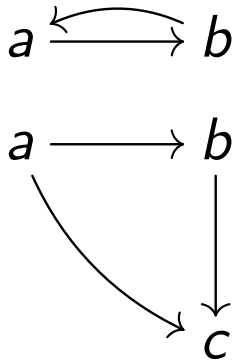
The fundamental groupoid of a type

The usual laws of identity provide the **groupoid operations**:

$$r : \mathtt{Id}(a, a) \qquad \text{reflexivity}$$

$$s : \mathtt{Id}(a, b) \to \mathtt{Id}(b, a) \qquad \text{symmetry}$$

$$t : \mathtt{Id}(a, b) \times \mathtt{Id}(b, c) \to \mathtt{Id}(a, c) \qquad \text{transitivity}$$



But as in topology, the **groupoid equations**:

$$p \cdot (q \cdot r) = (p \cdot q) \cdot r \qquad \text{associativity}$$

$$p^{-1} \cdot p = 1 = p \cdot p^{-1} \qquad \text{inverse}$$

$$1 \cdot p = p = p \cdot 1 \qquad \text{unit}$$

do not hold **strictly**, but only "**up to homotopy**".

The fundamental groupoid of a type

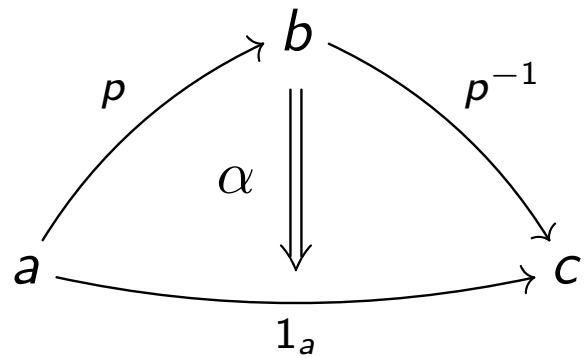
This means that they are **witnessed by identities of the next higher order**. That is, instead of

$$p^{-1} \cdot p = 1_a \,,$$

we have a term,

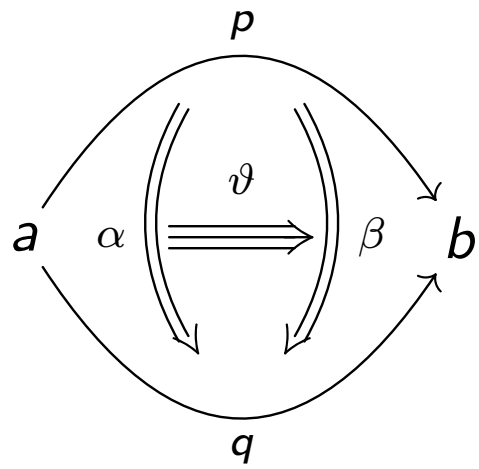
$$\alpha : \mathtt{Id}_{\mathtt{Id}} \left(p^{-1} \cdot p,\, 1_a \right) \,,$$

which can be represented:



The fundamental groupoid of a type

In this way, each type in the system is endowed with the structure of an ∞-**groupoid**, with terms, identities between terms, identities between identities, ...



Such structures have already occurred elsewhere in Mathematics, e.g. in Grothendieck's famous **homotopy hypothesis**.

Homotopy n-types (Voevodsky)

The universe of all types is naturally **stratified** by "homotopical dimension": the level at which the fundamental groupoid becomes trivial. This leads to a modification of **Propositions-as-Types**.

Formally, a type X is called:

contractible iff $\sum_{x:X} \prod_{y:X} \mathtt{Id}_X(x,y)$,
X *has essentially one term.*

proposition iff $\prod_{x,y:X} \mathrm{Contr}(\mathtt{Id}_X(x,y))$,
X *has at most one term.*

set iff $\prod_{x,y:X} \mathrm{Prop}(\mathtt{Id}_X(x,y))$,
Identity of terms is always a proposition.

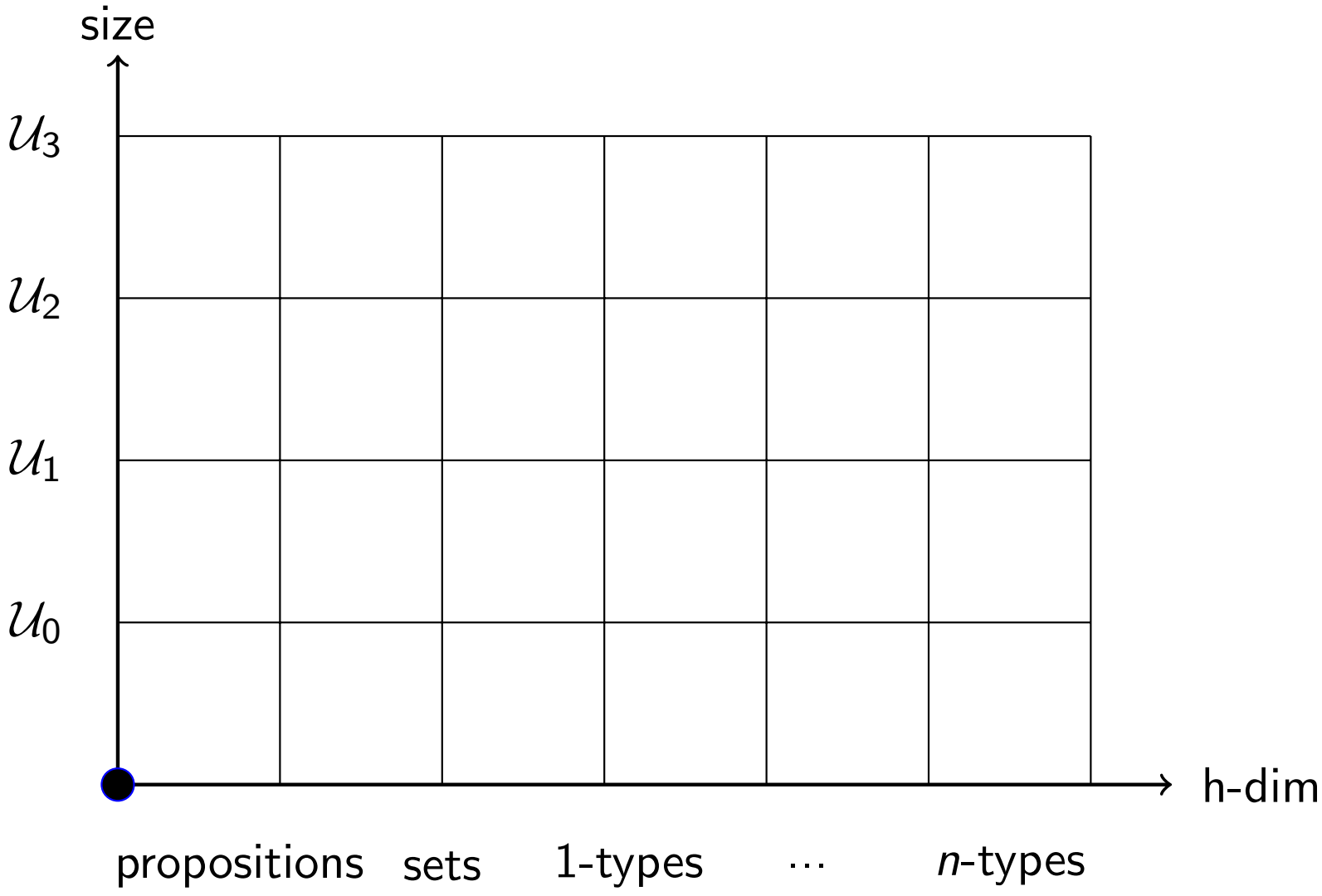
1-type iff $\prod_{x,y:X} \mathrm{Set}(\mathtt{Id}_X(x,y))$,
Identity of identity terms is always a proposition.

$(n+1)$-type iff $\prod_{x,y:X} \mathrm{nType}(\mathtt{Id}_X(x,y))$.

Higher types are **structures**, rather than mere **propositions**.

The Hierarchy of n-Types

This gives a **new view of the mathematical universe**, in which types can have intrinsic **higher-dimensional structure**.



II. The Univalence Axiom (Voevodsky)

Voevodsky has proposed a new axiom to be added to HoTT: the **Univalence Axiom**.

- It captures the informal mathematical practice of **identifying isomorphic objects**.
- It is quite **useful** from a practical point of view, especially when combined with HITs.
- It is **formally incompatible** with the assumption that all types are **sets**.
- It implies a version of **philosophical structuralism**.

Isomorphism

In type theory, the usual notion of *isomorphism* $A \cong B$ is definable:

$$A \cong B \;\Leftrightarrow\; \textit{there are } f : A \to B \textit{ and } g : B \to A \textit{ such that } gf(x) = x \textit{ and } fg(y) = y.$$

Formally, there is the type of isomorphisms:

$$\mathrm{Iso}(A, B) \;:=\; \sum_{f:A\to B} \sum_{g:B\to A} \Big(\prod_{x:A} \mathtt{Id}_A(gf(x), x) \times \prod_{y:B} \mathtt{Id}_B(fg(y), y) \Big)$$

Thus $A \cong B$ iff this type is inhabited by a closed term, which is then just an isomorphism between A and B.

Equivalence

There is also a more refined notion of **equivalence of types**,

$$A \simeq B$$

which adds a further "coherence" condition relating the *proofs* of $gf(x) = x$ and $fg(y) = y$ via f and g.

- Since every isomorphism can be promoted to an equivalence, $A \simeq B$ if and only if $A \cong B$.
- But as a condition on a map $f : A \to B$, being an equivalence is a **proposition**, and therefore is more precise.
- Under the homotopy interpretation, $A \simeq B$ is the type of all *homotopy equivalences*.
- Equivalence also subsumes *categorical equivalence* (for 1-types), *isomorphism of sets* (for 0-types), and *logical equivalence* (for propositions).

Invariance

One can show that all definable properties $P(X)$ of types X **respect type equivalence**:

$$A \simeq B \text{ and } P(A) \text{ implies } P(B)$$

This is a generalization of the familiar *isomorphism invariance of first-order logic*.

Equivalent types $A \simeq B$ are therefore formally **indiscernable**,

$$P(A) \Rightarrow P(B), \text{for all definable } P$$

How then is equivalence related to the **identity** of types A and B?

Univalence

To reason about **identity of types**, we need a *type universe* $\mathcal{U}$, with an identity type,

$$\mathtt{Id}_{\mathcal{U}}(A, B).$$

Since identity implies equivalence there is a comparison map,

$$\mathtt{Id}_{\mathcal{U}}(A, B) \to (A \simeq B).$$

The **Univalence Axiom** asserts that this map is an equivalence:

$$\mathtt{Id}_{\mathcal{U}}(A, B) \;\simeq\; (A \simeq B) \qquad \text{(UA)}$$

So UA can be stated: *"Identity is equivalent to equivalence."*

The Univalence Axiom: Remarks

- Since UA is an equivalence, there is a map coming back:

$$\mathtt{Id}_{\mathcal{U}}(A, B) \longleftarrow (A \simeq B)$$

  In this sense, **equivalent objects are identical**.
- So logically equivalent propositions are identical, as are isomorphic sets, categorically equivalent groups, etc..
- UA implies that $\mathcal{U}$, in particular, is not a set (0-type).
- UA is formally consistent with ZFC$^+$, via a model construction in simplicial sets due to Voevodsky.
- The computational character of UA is an open question (see Part V below).

III. Higher inductive types (Lumsdaine-Shulman)

The natural numbers $\mathbb{N}$ are given as an (ordinary) inductive type:

$$\mathbb{N} := \begin{cases} 0 : \mathbb{N} \\ s : \mathbb{N} \to \mathbb{N} \end{cases}$$

The **recursion property** is captured by an elimination rule:

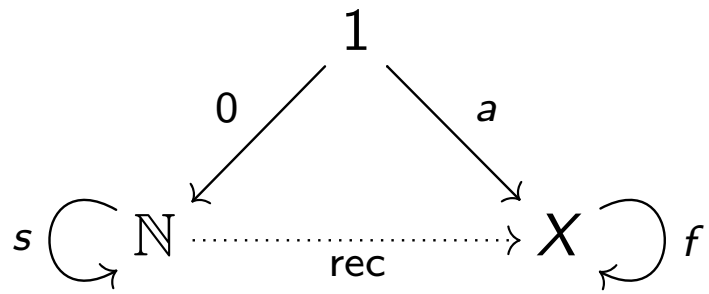
$$\frac{a : X \qquad f : X \to X}{\mathrm{rec}(a, f) : \mathbb{N} \to X}$$

with computation rules:

$$\begin{aligned} \mathrm{rec}(a, f)(0) &= a \\ \mathrm{rec}(a, f)(sn) &= f(\mathrm{rec}(a, f)(n)) \end{aligned}$$

Inductive types: The natural numbers $\mathbb{N}$

In other words, $(\mathbb{N}, 0, s)$ is the **free** structure of this kind:



The map $\mathrm{rec}(a, f) : \mathbb{N} \to X$ is unique.

Theorem

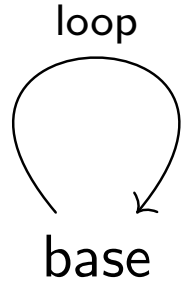
$\mathbb{N}$ *is a set (i.e. a* 0*-type).*

Higher inductive types: The circle $\mathbb{S}$

The homotopical circle $\mathbb{S}$ can be given as an inductive type involving a "higher-dimensional" generator:

$$\mathbb{S} := \begin{cases} \mathsf{base} : \mathbb{S} \\ \mathsf{loop} : \mathtt{Id}_{\mathbb{S}}(\mathsf{base}, \mathsf{base}) \end{cases}$$

Here we think of $\mathsf{loop} : \mathtt{Id}_{\mathbb{S}}(\mathsf{base}, \mathsf{base})$ as a non-trivial path,



Higher inductive types: The circle $\mathbb{S}$

$$\mathbb{S} := \begin{cases} \mathsf{base} : \mathbb{S} \\ \mathsf{loop} : \mathtt{Id}_{\mathbb{S}}(\mathsf{base}, \mathsf{base}) \end{cases}$$

The **recursion principle** of $\mathbb{S}$ is given by its elimination rule:

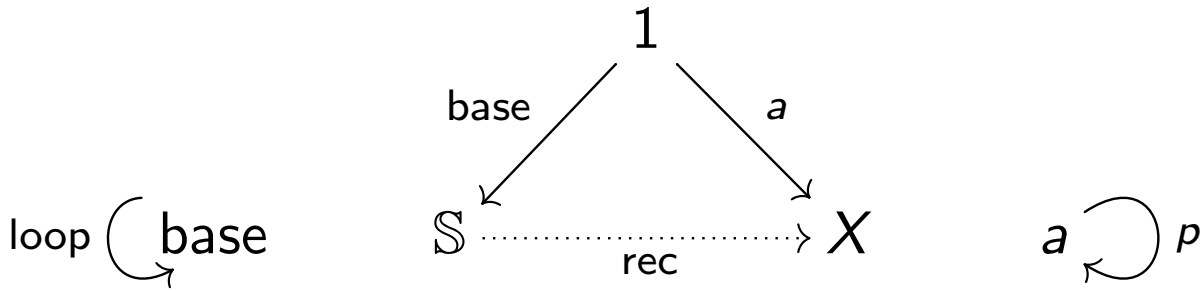
$$\frac{a : X \qquad p : \mathtt{Id}_{\mathbb{X}}(a, a)}{\mathsf{rec}(a, p) : \mathbb{S} \to X}$$

with computation rules:

$$\mathsf{rec}(a, p)(\mathsf{base}) = a$$
$$\mathsf{rec}(a, p)(\mathsf{loop}) = p$$

Higher inductive types: The circle $\mathbb{S}$

In other words, $(\mathbb{S}, \mathsf{base}, \mathsf{loop})$ is the **free** structure of this kind:



The map $\mathrm{rec}(a, p) : \mathbb{S} \to X$ is then unique **up to homotopy**.

Higher inductive types: The circle $\mathbb{S}$

This confirms that things are as they should be:

Theorem (Shulman 2011)

The type-theoretic circle $\mathbb{S}$ *has the correct homotopy groups:*

$$\pi_n(\mathbb{S}) = \begin{cases} \mathbb{Z}, & \text{if } n = 1, \\ 0, & \text{if } n \neq 1. \end{cases}$$

Corollary

$\mathbb{S}$ *is a* 1*-type.*

We sketch the proof of the theorem below — it combines classical homotopy theory with methods from constructive type theory, and uses the univalence axiom.

Higher inductive types: The interval $\mathbb{I}$

The unit interval $\mathbb{I} = [0, 1]$ is also an inductive type, on the data:

$$\mathbb{I} := \begin{cases} 0, 1 : \mathbb{I} \\ p : \mathtt{Id}_{\mathbb{I}}(0, 1) \end{cases}$$

We are now thinking of $p : \mathtt{Id}_{\mathbb{I}}(0, 1)$ as a "free path",

$$0 \xrightarrow{p} 1 \,.$$

Slogan:

In topology, we start with the **interval** and use it to define the notion of a **path**.

In HoTT, we start with the notion of a **path**, and use it to define the **interval**.

Higher inductive types: Summary

Many basic spaces and constructions can be introduced as HITs:

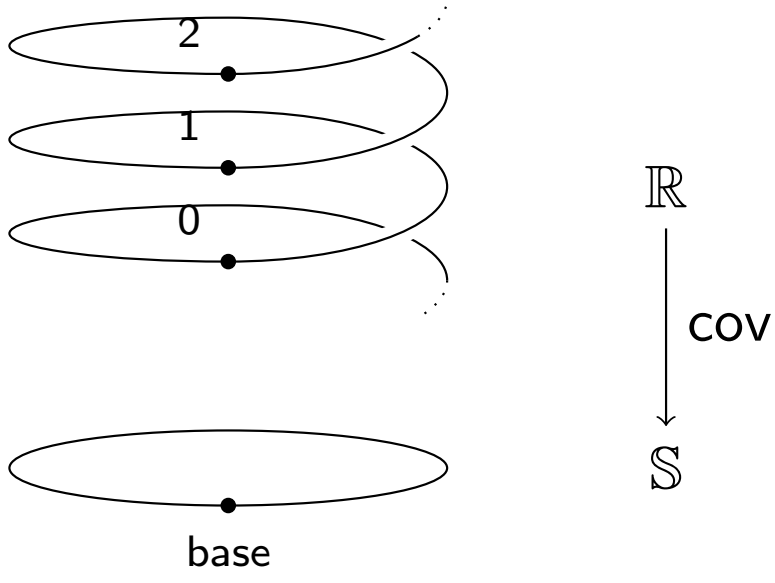
- higher spheres $\mathbb{S}^n$, cylinders, tori, cell complexes, ...,
- suspensions ΣA, mapping cylinders, homotopy pushouts, etc.,
- truncations, such as connected components $\pi_0(A)$ and "bracket" types $[A]$,
- (higher) homotopy groups π_n, Eilenberg-MacLane spaces $K(G, n)$, Postnikov systems,
- Quillen model structure,
- quotients by equivalence relations and more general quotients,
- free algebras, algebras presented by generators and relations,
- real numbers,
- cumulative hierarchy of sets.

IV. Synthetic reasoning and formalization

- In addition to doing "logical homotopy theory", HoTT can be used as a general **foundation for mathematics**.
- A new "**synthetic**" style of axiomatics based on the use of HITs simplifies and shortens many proofs.
- Much **classical mathematics** has already been developed: basic homotopy theory, category theory, analysis, algebra, set theory, ...
- Proofs can be fully formalized and verified in **computerized proof assistants** like Coq, Agda, and now Lean.

Synthetic reasoning: An example

To compute the fundamental group of the circle $\mathbb{S}$, as in classical algebraic topology we shall use the universal cover:



In HoTT, this will be a dependent type over $\mathbb{S}$, i.e. a type family,

$$\mathsf{cov} : \mathbb{S} \longrightarrow \mathcal{U}.$$

Synthetic reasoning: An example

To define such a type family

$$\mathsf{cov} : \mathbb{S} \longrightarrow \mathcal{U},$$

by the recursion property of the circle, we need the following data:

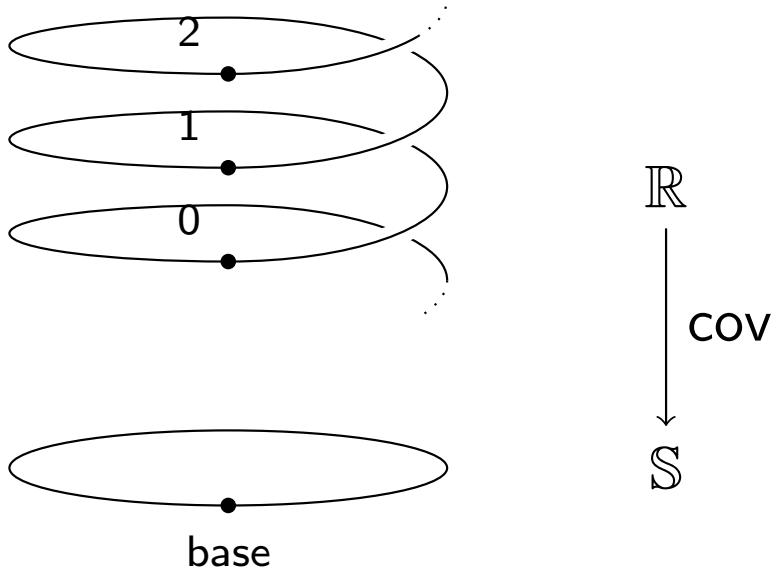
- a point $A : \mathcal{U}$
- a loop $p : \mathtt{Id}_{\mathcal{U}}(A, A)$

For the point A we take the integers $\mathbb{Z}$.

By the univalence axiom, to give a loop $p : \mathtt{Id}_{\mathcal{U}}(A, A)$ in $\mathcal{U}$,
it suffices to give an equivalence $\mathbb{Z} \simeq \mathbb{Z}$.

Since $\mathbb{Z}$ is a set, equivalences are just isomorphisms;
so we can take the successor function $\mathsf{succ} : \mathbb{Z} \cong \mathbb{Z}$.

Synthetic reasoning: An example

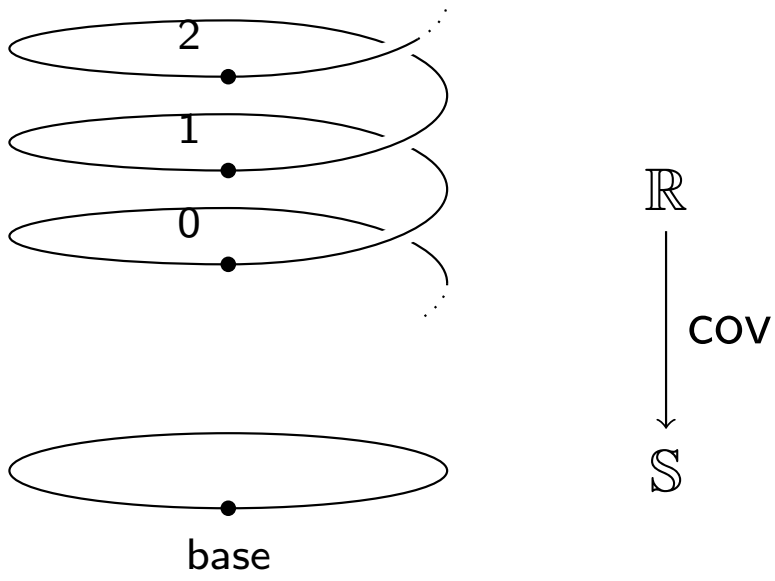


Definition (Universal Cover of $\mathbb{S}$)

The dependent type $\mathsf{cov} : \mathbb{S} \longrightarrow \mathcal{U}$ is given by circle recursion, with:

$$\mathsf{cov}(\mathsf{base}) = \mathbb{Z}\,,$$
$$\mathsf{cov}(\mathsf{loop}) = \mathsf{ua}(\mathsf{succ})\,.$$

Synthetic reasoning: An example



As in classical homotopy theory, we now use the universal cover to define the "winding number" of any path $p : \mathtt{Id}_{\mathbb{S}}(\mathsf{base}, \mathsf{base})$ by $\mathsf{wind}(p) = p_*(0)$. This gives a map

$$\mathsf{wind} : \Omega(\mathbb{S}) \longrightarrow \mathbb{Z},$$

inverse to the map $\mathbb{Z} \longrightarrow \Omega(\mathbb{S})$ given by iterated composition,

$$i \mapsto \mathsf{loop}^i.$$

The formal proof

```
(* *** Definition of the circle. *)

Module Export Circle.

Local Inductive S1 : Type :=
| base : S1.

Axiom loop : base = base.

Definition S1_rect (P : S1 -> Type) (b : P base) (l : loop # b = b)
  : forall (x:S1), P x
  := fun x => match x with base => b end.

Axiom S1_rect_beta_loop
  : forall (P : S1 -> Type) (b : P base) (l : loop # b = b),
  apD (S1_rect P b l) loop = l.

End Circle.

(* *** The non-dependent eliminator *)

Definition S1_rectnd (P : Type) (b : P) (l : b = b)
  : S1 -> P
  := S1_rect (fun _ => P) b (transport_const _ _ @ l).

Definition S1_rectnd_beta_loop (P : Type) (b : P) (l : b = b)
  : ap (S1_rectnd P b l) loop = l.
Proof.
  unfold S1_rectnd.
  refine (cancelL (transport_const loop b) _ _ _).
  refine ((apD_const (S1_rect (fun _ => P) b _) loop)^ @ _).
  refine (S1_rect_beta_loop (fun _ => P) _ _).
Defined.
```

```
(* First we define the appropriate integers. *)

Inductive Pos : Type :=
| one : Pos
| succ_pos : Pos -> Pos.

Definition one_neq_succ_pos (z : Pos) : ~ (one = succ_pos z)
  := fun p => transport (fun s => match s with one => Unit | succ_pos t => Empty end) p tt.

Definition succ_pos_injective {z w : Pos} (p : succ_pos z = succ_pos w) : z = w
  := transport (fun s => z = (match s with one => w | succ_pos a => a end)) p (idpath z).

Inductive Int : Type :=
| neg : Pos -> Int
| zero : Int
| pos : Pos -> Int.

Definition neg_injective {z w : Pos} (p : neg z = neg w) : z = w
  := transport (fun s => z = (match s with neg a => a | zero => w | pos a => w end)) p (idpath z).

Definition pos_injective {z w : Pos} (p : pos z = pos w) : z = w
  := transport (fun s => z = (match s with neg a => w | zero => w | pos a => a end)) p (idpath z).

Definition neg_neq_zero {z : Pos} : ~ (neg z = zero)
  := fun p => transport (fun s => match s with neg a => z = a | zero => Empty
  | pos _ => Empty end) p (idpath z).

Definition pos_neq_zero {z : Pos} : ~ (pos z = zero)
  := fun p => transport (fun s => match s with pos a => z = a
  | zero => Empty | neg _ => Empty end) p (idpath z).

Definition neg_neq_pos {z w : Pos} : ~ (neg z = pos w)
  := fun p => transport (fun s => match s with neg a => z = a
  | zero => Empty | pos _ => Empty end) p (idpath z).
```

```
(* And prove that they are a set. *)

Instance hset_int : IsHSet Int.
Proof.
  apply hset_decidable.
  intros [n | | n] [m | | m].
  revert m; induction n as [|n IHn]; intros m; induction m as [|m IHm].
  exact (inl 1).
  exact (inr (fun p => one_neq_succ_pos _ (neg_injective p))).
  exact (inr (fun p => one_neq_succ_pos _ (symmetry _ _ (neg_injective p)))).
  destruct (IHn m) as [p | np].
  exact (inl (ap neg (ap succ_pos (neg_injective p)))).
  exact (inr (fun p => np (ap neg (succ_pos_injective (neg_injective p))))).
  exact (inr neg_neq_zero).
  exact (inr neg_neq_pos).
  exact (inr (neg_neq_zero o symmetry _ _)).
  exact (inl 1).
  exact (inr (pos_neq_zero o symmetry _ _)).
  exact (inr (neg_neq_pos o symmetry _ _)).
  exact (inr pos_neq_zero).
  revert m; induction n as [|n IHn]; intros m; induction m as [|m IHm].
  exact (inl 1).
  exact (inr (fun p => one_neq_succ_pos _ (pos_injective p))).
  exact (inr (fun p => one_neq_succ_pos _ (symmetry _ _ (pos_injective p)))).
  destruct (IHn m) as [p | np].
  exact (inl (ap pos (ap succ_pos (pos_injective p)))).
  exact (inr (fun p => np (ap pos (succ_pos_injective (pos_injective p))))).
Defined.
```

```
 (* Successor is an autoequivalence of [Int]. *)

Definition succ_int (z : Int) : Int
  := match z with
       | neg (succ_pos n) => neg n
       | neg one => zero
       | zero => pos one
       | pos n => pos (succ_pos n)
     end.

Definition pred_int (z : Int) : Int
  := match z with
       | neg n => neg (succ_pos n)
       | zero => neg one
       | pos one => zero
       | pos (succ_pos n) => pos n
     end.

Instance isequiv_succ_int : IsEquiv succ_int
  := isequiv_adjointify succ_int pred_int _ _.
Proof.
  intros [[|n] | | [|n]]; reflexivity.
  intros [[|n] | | [|n]]; reflexivity.
Defined.

(* Now we do the encode/decode. *)

Section AssumeUnivalence.
Context `{Univalence} `{Funext}.

Definition S1_code : S1 -> Type
  := S1_rectnd Type Int (path_universe succ_int).
```

```
(* Transporting in the codes fibration is the successor autoequivalence. *)

Definition transport_S1_code_loop (z : Int)
  : transport S1_code loop z = succ_int z.
Proof.
  refine (transport_compose idmap S1_code loop z @ _).
  unfold S1_code; rewrite S1_rectnd_beta_loop.
  apply transport_path_universe.
Defined.

Definition transport_S1_code_loopV (z : Int)
  : transport S1_code loop^ z = pred_int z.
Proof.
  refine (transport_compose idmap S1_code loop^ z @ _).
  rewrite ap_V.
  unfold S1_code; rewrite S1_rectnd_beta_loop.
  rewrite <- path_universe_V.
  apply transport_path_universe.
Defined.

(* Encode by transporting *)

Definition S1_encode (x:S1) : (base = x) -> S1_code x
  := fun p => p # zero.

(* Decode by iterating loop. *)

Fixpoint loopexp {A : Type} {x : A} (p : x = x) (n : Pos) : (x = x)
  := match n with
       | one => p
       | succ_pos n => loopexp p n @ p
     end.
```

```
Definition looptothe (z : Int) : (base = base)
  := match z with
       | neg n => loopexp (loop^) n
       | zero => 1
       | pos n => loopexp (loop) n
     end.

Definition S1_decode (x:S1) : S1_code x -> (base = x).
Proof.
  revert x; refine (S1_rect (fun x => S1_code x -> base = x) looptothe _).
  apply path_forall; intros z; simpl in z.
  refine (transport_arrow _ _ _ @ _).
  refine (transport_paths_r loop _ @ _).
  rewrite transport_S1_code_loopV.
  destruct z as [[|n] | | [|n]]; simpl.
  by apply concat_pV_p.
  by apply concat_pV_p.
  by apply concat_Vp.
  by apply concat_1p.
  reflexivity.
Defined.

(* The nontrivial part of the proof that decode and encode are equivalences is
showing that decoding followed by encoding is the identity on the fibers over [base]. *)

Definition S1_encode_looptothe (z:Int)
  : S1_encode base (looptothe z) = z.
Proof.
  destruct z as [n | | n]; unfold S1_encode.
  induction n; simpl in *.
  refine (moveR_transport_V _ loop _ _ _).
  by apply symmetry, transport_S1_code_loop.
  rewrite transport_pp.
  refine (moveR_transport_V _ loop _ _ _).
```

```
  refine (_ @ (transport_S1_code_loop _)^).
  assumption.
  reflexivity.
  induction n; simpl in *.
  by apply transport_S1_code_loop.
  rewrite transport_pp.
  refine (moveR_transport_p _ loop _ _ _).
  refine (_ @ (transport_S1_code_loopV _)^).
  assumption.
Defined.

(* Now we put it together. *)

Definition S1_encode_isequiv (x:S1) : IsEquiv (S1_encode x).
Proof.
  refine (isequiv_adjointify (S1_encode x) (S1_decode x) _ _).
  (* Here we induct on [x:S1].  We just did the case when [x] is [base]. *)
  refine (S1_rect (fun x => Sect (S1_decode x) (S1_encode x))
    S1_encode_looptothe _ _).
  (* What remains is easy since [Int] is known to be a set. *)
  by apply path_forall; intros z; apply set_path2.
  (* The other side is trivial by path induction. *)
  intros []; reflexivity.
Defined.

Definition equiv_loopS1_int : (base = base) <~> Int
  := BuildEquiv _ _ (S1_encode base) (S1_encode_isequiv base).

End AssumeUnivalence.
```

Formalization of mathematics

- HoTT uses a **synthetic** methodology involving high-level axiomatics and structural descriptions, allowing for shorter, more abstract proofs that are closer to mathematical practice than the **analytic** method of ZFC.
- The idea of logical foundations of math has always had great philosophical and conceptual appeal, **but** in the past it was too lengthy and cumbersome to be of much **practical** use.
- Explicit formalization is now **feasible**, because computers can take over what was once too tedious or complicated to be done by hand **and** because a synthetic approach like HoTT is more efficient than an analytic one.
- Formalization can provide a **practical tool** for working mathematicians, through increased certainty and precision, support for remote collaboration, cumulativity of results, searchable libraries of code, etc.

V. Cubical Homotopy Type Theory

What's still missing from HoTT (if anything)?

- Basic MLTT has good proof-theoretic properties that also make it well-suited for use in computational proof assistants: strong normalization of terms, decidability of type-checking, decidability of judgemental equality, canonicity, etc. This is part of the **constructive character** of the system.
- But when we add new **axioms** like Univalence and HITs, this constructive character may be spoiled. Instances of UA cannot always be eliminated, and new primitive terms of higher Id-type need not reduce to normal forms.
- Voevodsky has conjectured that there is an algorithm for "**normalization up to homotopy**", which would partially restore the constructive character of the system.
- This would be important philosophically, but also **practically** for a new generation of proof assistants based on HoTT.

Cubical HoTT: Recent work

Why cubical?

- Some success was had by Licata-Harper (2011) and Shulman (2013) in verifying the conjecture at **low dimensions**, using methods based on groupoids.
- Coquand has recently (2014) devised a promising approach based on a **constructive** model of HoTT in **cubical sets**, which are a version of ∞-groupoids.
- Cubical sets are a **combinatorial** model of homotopy theory used in algebraic topology. Like the more familiar simplicial sets, they provide a more "algebraic" setting to study the homotopy theory of topological spaces.
- Voevodsky's original model of UA used classical simplicial sets and is **not constructive**. Known models of HITs are also based on **classical methods** from the theory of ∞-toposes.

Cubical HoTT: Recent success

Cubes rule!

- The cubical model also suggests enriching the type theory with additional **cubical operations** and **equations** which are present in the model and which allow calculations that are otherwise available only "up-to-homotopy". This makes the system **more computational**.
- Coquand et al. have programmed a **proof checker** for such a **cubical type theory**, in which all terms — even those involving UA and HITs — compute to normal forms.
- Brunerie and Licata (LICS 2015) have a variant system of **cubical HoTT** in which e.g. the proof that $\mathbf{T}^2 \simeq \mathbf{S}^1 \times \mathbf{S}^1$ is short and sweet (in contrast to the original "heroic" proof in plain HoTT given by Sojakova in 2013).

The category of cubes

Definition

The **cube category** $\mathbb{C}$ can be defined as the dual of the category $\mathbb{B}$ of finite, strictly **bipointed sets**,

$$\mathbb{C} \;=_{\mathsf{def}}\; \mathbb{B}^{\mathsf{op}}\,.$$

By Lawvere duality $\mathbb{C}$ is the free finite-product category on the *bipointed object* $[0] \to [1] \leftarrow [0]$, where $[n]$ is the **n-cube**.
The objects of $\mathbb{C}$ are all the finite powers of $[1]$,

$$[0], [1], [2] = [1] \times [1], \ldots, [n], \ldots$$

The arrows are the maps that can be composed from the fp-structure and the basic points $0, 1 : [0] \rightrightarrows [1]$.
They can also be represented as the **terms** of an **algebraic theory**.

Cubical Sets

Definition

The category **cSet** of **cubical sets** is the **presheaves** on $\mathbb{C}$.
It is thus equivalent to the *covariant* functors on $\mathbb{B}$,

$$\mathbf{Set}^{\mathbb{C}^{\mathrm{op}}} \cong \mathbf{Set}^{\mathbb{B}}\,.$$

The **cubes** in **cSet** are the *representable functors*:

$$\mathtt{I}^n = \hom_{\mathbb{C}}(-, [n])\,.$$

The **interval** object $\mathtt{I} = \hom_{\mathbb{C}}(-, [1])$ generates all the other cubes, which are closed under finite products and satisfy:

$$\mathtt{I}^n \times \mathtt{I}^m \cong \mathtt{I}^{n+m}\,.$$

Cubical Sets

The interval $1 + 1 \to \mathtt{I}$ in **cSet** is **universal**, in the following sense.

Theorem (A. 2015)

The category **cSet** *of cubical sets is the* **classifying topos** *for strictly bipointed objects.*

- This allows us to relate **cSet** to other logical and homotopical models in toposes.
- Other models of type theory, such as **sSet**, have a **canonical comparision** with **cSet**.
- Since $\mathbb{C}$ is a **strict test category** in the sense of Grothendieck, **cSet** has "the same" homotopy theory as classical spaces.

Path spaces in cubical sets

The interval $1 + 1 \to \mathtt{I}$ endows each cubical set A with a **canonical path object**,

$$A^{\mathtt{I}} \to A^{1+1} \cong A \times A\,.$$

The object $A^{\mathtt{I}}$ has the special property,

$$A^{\mathtt{I}}(n) \;\cong\; A(n+1).$$

So an n-cube of **paths** in A is just an $n+1$-cube in A.

This **combinatorial specification** makes this choice of a path object very well-behaved. Using it as the $\mathtt{Id}$-**type** of A as in $\mathtt{Id}_A = A^{\mathtt{I}}$ then implies many new **equations** and special type-theoretic conditions, such as:

$$\begin{aligned} \mathtt{Id}_{\mathtt{Id}_A} &= (A^{\mathtt{I}})^{\mathtt{I}} \cong A^{\mathtt{I}\times\mathtt{I}}\,, \\ \mathtt{Id}_{A+B} &= (A+B)^{\mathtt{I}} \;\cong\; A^{\mathtt{I}} + B^{\mathtt{I}} = \mathtt{Id}_A + \mathtt{Id}_B\,. \end{aligned}$$

Path spaces in cubical sets

Thus we are led to ask:

When does $A^{\mathtt{I}} \to A \times A$ satisfy the rules for `Id`*-types?*

Theorem (A. 2015)

The path space $A^{\mathtt{I}} \to A \times A$ satisfies all the rules for `Id`*-types if*

1. *The obect A is a Kan complex.*
2. *The dependent types $B \to A$ are Kan fibrations.*

The notions of **Kan complex** and **Kan fibration** are from algebraic topology, and are determined by certain **filling conditions**.

There is a partial **converse** to the theorem, which derives these filling conditions from the `Id`-rules.

Univalence in cubical sets

The **rough idea** is this:

- A path $c : \mathtt{Id}_{\mathcal{U}}(A, B)$ in the universe $\mathcal{U}$ of types corresponds to a map $c : \mathtt{I} \to \mathcal{U}$, since $\mathtt{Id}_{\mathcal{U}} = \mathcal{U}^{\mathtt{I}}$.
- Such a map determines a **fibration** $C \to \mathtt{I}$ over the 1-cube, with $C_0 = A$ and $C_1 = B$.
- Since there is a (distinguished) path $p : \mathtt{Id}_{\mathtt{I}}(0, 1)$ in $\mathtt{I}$, and $C \to \mathtt{I}$ is a fibration, we have the **transport** map

$$p_* : A = C_0 \to C_1 = B\,,$$

  which is an **equivalence** $A \simeq B$.
- This is the map $\mathtt{Id}_{\mathcal{U}}(A, B) \to A \simeq B$ which by UA is supposed to have an **inverse**.
- Given an equivalence $e : A \simeq B$, we can build a suitable fibration $A +_e B \to \mathtt{I}$ using the **mapping cylinder** construction from homotopy theory.

Summary

- Under the new homotopical interpretation, constructive type theory provides a "logic of homotopy".
- Some new logical ideas are suggested by the homotopy interpretation: the Univalence Axiom, Higher Inductive Types.
- Using a new synthetic approach, many basic results have already been formalized in computational proof assistants.
- Cubical methods promise to make the theory more combinatorial, and so more computational.
- This should support its use as the basis of a new generation of computerized proof assistants.
- The formalization of mathematics in a rigorous system of foundations with a strongly structural character presents a new challenge for philosophers of mathematics.

More Information

The Website:

`www.HomotopyTypeTheory.org`

The Book:

Homotopy Type Theory:
Univalent Foundations of Mathematics
The Univalent Foundations Program
Institute for Advanced Study, 2013