Homotopy type theory and synthetic homotopy theory

Jonathan Weinberger

MπIM Bonn
Topology Seminar

Mar 14, 2022

Materialism vs. structuralism

- Early 20th century: set theory as mathematical foundations
- Everything is a *set*.
- *Elements* aren't an actual notion.
- Different ways to encode natural numbers:
- Von Neumann: $0 := \emptyset$, $1 := 0 \cup \{0\} = \{\emptyset\}$, $2 := 1 \cup \{1\} = \{\emptyset, \{\{\emptyset\}\}\}, \ldots$
- Zermelo: $0' := \emptyset$, $1' := \{0'\} = \{\emptyset\}$, $2' := \{1'\} := \{\{\emptyset\}\}, \ldots$
- Some statements about single natural numbers are now *dependent on the coding*:
- $0 \in 2$, but $0' \notin 2'$
- What does a statement like "$n \in m$" even mean?
- Syntactically, we can even make statements such as:
- $\mathbb{Q} \in \mathbb{C}$, $0 \in \pi$, $\frac{1}{2} \in \frac{2}{4}$, $\mathbb{R} \in \exp(2)$
- Not how we think about the $\in$-relation!

Equality vs. isomorphism

- In modern-day mathematics, we often treat isomorphic structures as equal.
- Problem: If $A \cong B$ and $x \in A$, then in general $x \notin B$!
- Examples: $2\mathbb{N} \cong 2\mathbb{N} + 1$ as sets, $\mathbb{Z}/2\mathbb{Z} \cong \{\mathrm{id}, (01)\}$ as groups, $(-1, 1) \cong (-\pi, \pi)$ as topological spaces, ...
- Solution: Make the isomorphism explicit. Given $\varphi : A \cong B$, if $x \in A$ we do have $\varphi(x) \in B$.
- But this is a rather manual process: syntactically, $x \in B$ makes sense (*well-formed*), even if $A \not\subseteq B$.

More structuralist foundations?

- Question: Alternative foundations that have more structuralist flavor?
- Some desiderata:
  - abstract away from *encodings* ($\mathbb{N}$ vs. $\mathbb{N}'$)
  - disallow *"nonsense expressions"* like $0 \in 1$, $\mathbb{N} \in \pi, \ldots$
  - capture *isomorphism* more natively
- Answer: Yes, for instance use *type theory* instead of **set theory.**
- In particular, the variant due to Per Martin-Löf, **Martin-Löf type theory (MLTT).**
- Extension of MLTT: "Isomorphism as equality" made precise by Vladimir Voevodsky's **Univalence Axiom** $\rightsquigarrow$ **Homotopy Type Theory/Univalent Foundations (HoTT/UF)**
- **In this talk:** Basic vocabulary of MLTT & taste of *synthetic* homotopy theory in HoTT

Homotopical foundations? Sets vs. homotopy types

Let's follow Yuri Manin[1] in imagining what homotopical foundations could look like:

*Instead of **sets**, clouds of discrete elements, we envisage some sorts of vague **spaces**, which can be very severely deformed, mapped one to another, and all the while the specific space is not impor- tant, but only the space **up to deformation**.*

Replace **sets** by **homotopy types**!

All **maps** are **continuous**.

[1] *M. Gelfand*, Notices Am. Math. Soc. 56, Non.10, 1268–1274 (2009; Zbl 1178.01044). *Emphases and interpretations due to the speaker.*

Homotopical foundations? Everything is continuous

Earlier, all these spaces were thought of as Cantor ***sets with topology****, their maps were Cantor maps, some of them were* ***homotopies*** *that should have been factored out, and so on.*

Set theory is fundamentally discrete (*e.g.* def. of a topological space).

Homotopical foundations should be fundamentally continuous.

Then, have to *break* continuity this to achieve discreteness:

I am pretty strongly convinced that there is an ongoing reversal in the collective consciousness of mathematicians: the right hemispherical and homotopical picture of the world becomes the basic intuition, and if you want to get a discrete set, then you pass to the set of connected components of a space defined only up to homotopy.

Dependent type theory (DTT)

- DTT consists of derivations of **judgments** involving:

$$\textbf{types } A, \quad \textbf{terms } a : A, \quad \textbf{contexts } \Gamma \equiv [x_1 : A_1, \ldots, x_n : A_n]$$

- *Intuitions:* **types:** objects, **terms:** elements, **contexts:** lists of variables
- Valid derivations are produced from a bunch of given **rules** (next slide).

- **Dependent types/type families $\Gamma \vdash A$:**

  "A is a (dep.) type in context Γ"
- This means: for any $\vec{x}$ in Γ, we have that $A(\vec{x})$ is a type.
- Examples:
  - $\cdot \vdash \mathrm{Bool}$
  - $\cdot \vdash \mathbb{N}$ and $\cdot \vdash \mathbb{R}$
  - $n : \mathbb{N} \vdash \mathbb{R}^n$
  - $p : \mathrm{Prime}, n : \mathbb{N} \vdash \mathbb{F}_{p^n}$
  - $x : M \vdash T_x M$

- **Dependent terms $\Gamma \vdash f : A$:**

  "f is a (dep.) term in A (over ctxt. Γ)"
- This means: for any $\vec{x}$ in Γ, we have a term $f(\vec{x}) : A(\vec{x})$.
- Examples:
  - $\cdot \vdash \bot : \mathrm{Bool}$ and $\cdot \vdash \top : \mathrm{Bool}$
  - $n : \mathbb{N} \vdash \mathrm{succ}(n) : \mathbb{N}$
  - $n : \mathbb{N} \vdash \vec{0}_n : \mathbb{R}^n$
  - $p : \mathrm{Prime}, n : \mathbb{N} \vdash 1_{p^n} : \mathbb{F}_{p^n}$
  - $x : M \vdash \vec{0}_x : T_x M$

Type formers: new types from old ones

- **Product types**
- Given types A and B are types, there ex. a type $A \times B$ (*formation*).

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \times B}\ (\times\text{-Form})$$

- Given terms $a : A$ and $b : B$, there ex. a term $\langle a, b\rangle : A \times B$ (*introduction*).

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b\rangle : A \times B}\ (\times\text{-Intro})$$

- Given a term $p : A \times B$, there ex. terms $\mathrm{pr}_1(p) : A$ and $\mathrm{pr}_2(p) : B$ (*elimination*).

$$\frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{pr}_1(p) : A}\ (\times\text{-Elim}_1) \quad \frac{\Gamma \vdash p : A \times B}{\Gamma \vdash \mathrm{pr}_2(p) : B}\ (\times\text{-Elim}_2)$$

- *Computation:* $\mathrm{pr}_1(\langle a, b\rangle) \equiv a$, $\mathrm{pr}_2(\langle a, b\rangle) \equiv b$

- **Function types**
- Given types A and B, there ex. a type $A \to B$.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \to B}\ (\to\text{-Form})$$

- Given for all $a : A$ a term $f(a) : B$, there ex. a term $\lambda a.f(a) : A \to B$ (*i.e.* "$a \mapsto f(a)$").

$$\frac{\Gamma, a : A \vdash f(a) : B}{\Gamma \vdash \lambda a.f(a) : A \to B}\ (\to\text{-Intro})$$

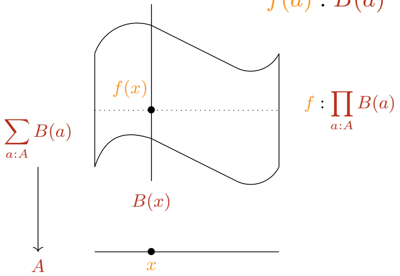
- Given terms $f : A \to B$ and $a : A$, there ex. a term $f(a) : B$.

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B}\ (\to\text{-Elim})$$

- $(\lambda a.f(a))(x) \equiv f(x)$ and $\lambda a.f(a) \equiv f$

Dependent type formers

- Terms of **product type**: pairs $\langle a, b\rangle : A \times B$ with $a : A$ and $b : B$.
- **Dependent generalization**: A type and $a : A \vdash B(a)$ dep. type $\rightsquigarrow$ **dep. pair** or **dep. *sum* type** $\sum_{a:A} B(a)$ whose elements are pairs $\langle a, b\rangle$ with $a : A$ and $b : B(a)$.
- Terms of **function type**: functions $f : A \to B$ taking $a : A$ to $f(a) : B$.
- **Dependent generalization**: A type and $a : A \vdash B(a)$ dep. type $\rightsquigarrow$ **dep. function** or **dep. *product* type** $\prod_{a:A} B(a)$ whose elements are functions (**sections**) f taking $a : A$ to $f(a) : B(a)$



Natural numbers

- The structure of natural numbers is freely generated 0 and succ.
- Proof by induction: Want to prove a property for all k. Then it suffices to assume that $k = 0$ or $k = n + 1$ (given that it holds for n).
- More generally: Given any set A, to define a function $f : \mathbb{N} \to A$, it suffices to define $f(0)$ and $f(n + 1)$ (given that $f(n)$ has been already defined).
- To define $\mathbb{N}$ in type theory, formulate the induction principle type-theoretically:

$$\frac{}{\cdot \vdash \mathbb{N}}\ \text{(Nat-Form)} \qquad \frac{}{\cdot \vdash 0 : \mathbb{N}}\ \text{(Nat-Intro}_0\text{)} \qquad \frac{}{n : \mathbb{N} \vdash \mathrm{succ}(n) : \mathbb{N}}\ \text{(Nat-Intro}_{\mathrm{succ}}\text{)}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : \prod_{n:\mathbb{N}} P(n) \to P(\mathrm{succ}(n))}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(p_0, p_s) : \prod_{n:\mathbb{N}} P(n)}\ \text{(Nat-Elim)}$$

& computation rules: $\mathrm{ind}_{\mathbb{N}}(p_0, p_s)(0) \equiv p_0 : P(0)$ and $\mathrm{ind}_{\mathbb{N}}(p_0, p_s)(\mathrm{succ}(n)) \equiv p_s(n, \mathrm{ind}_{\mathbb{N}}(p_0, p_s, n)) : P(\mathrm{succ}(n))$

The Curry–Howard interpretation

Type theory	Logic	Set theory
A	proposition A	set A
$x : A$	evidence/witness for A	element $x \in A$
$\mathbf{0}, \mathbf{1}$	$\bot, \top$	$\emptyset, \{\emptyset\}$
$A + B$	$A \vee B$	disjoint union $A + B$
$A \times B$	$A \wedge B$	set $A \times B$ of ordered pairs
$A \to B$	$A \Rightarrow B$	set $A \to B$ of functions
$x : A \vdash B(x)$	property/predicate $B(x)$	family of sets $(B_x)_{x \in A}$
$x : A \vdash b : B(x)$	conditional proof	choice of elements/section $x \mapsto \langle x, b(x) \rangle$
$\sum_{x:A} B(x)$	$\exists\, x. B(x)$	disjoint sum $\coprod_{x:A} B(x)$
$\prod_{x:A} B(x)$	$\forall\, x. B(x)$	product $\prod_{x:A} B(x)$

Equality vs. equality

- Recall: We wanted a nice (meaning intrinsic) way to treat isomorphism.
- So far, our type theory comes with two basic judgments of equality: term equality $x \equiv y : A$ and type equality $A \equiv B$.
- This is called **definitional** or **judgmental equality**.
- These are produced by *rewrite rules*, *i.e.* syntactic conversions, given by the postulated computation rules (*e.g.* $\mathrm{pr}_1(\langle a, b\rangle) \equiv a$).
- **Problem:** We cannot expect this to adequately model an interesting notion of isomorphism.
- In particular, having too many definitional equalities destroys the computational behavior of the theory. $\rightsquigarrow$ Can't use as programming language!
- **Solution:** Introduce another notion of equality!

Identity type: Formation and introduction

- Per Martin–Löf's **identity types** from the 1970s. $\rightsquigarrow$ **propositional equality**
- **Idea:** Let $x, y : A$. Then there is a type $(x =_A y)$ of *identifications* or *proofs* that x is equal to y (*formation*).
- In a topological picture, we could imagine $p : (x =_A y)$ to be a *path* from x to y (more on this later).
- For any $x : A$ there should be a term $\mathrm{refl}_x : (x =_A x)$ (*introduction*).

$$\frac{\Gamma \vdash A}{\Gamma \vdash x, y : A \vdash (x =_A y)}\ \text{Id-Form} \qquad \frac{\Gamma \vdash A}{\Gamma, x : A \vdash \mathrm{refl}_x : (x =_A x)}\ \text{Id-Intro}$$

Identity type: Elimination and computation

- **Q:** How do we *eliminate* out of $(x =_A y)$?
- **A:** Another induction principle!
- **Idea:** Identity types are freely generated by the reflexivity terms.
- **Identity** or **path induction:** Given a type B depending on $x, y : A$ and $p : x =_A y$, to give a term in $B(x, y, p)$ we can assume $y \equiv x$ and $p \equiv \mathrm{refl}_x$:

$$\frac{\Gamma \vdash A \qquad \Gamma, x : A, y : A, p : x =_A y \vdash B(x, y, p)}{\Gamma \vdash \mathrm{ind}_{=_A} : \prod_{a:A} B(a, a, \mathrm{refl}_a) \to \prod_{x,y:A} \prod_{p:(x =_A y)} B(x, y, p)} \text{ Id-Elim}$$

with *computation* rule: $\mathrm{ind}_{=_A}(q, a, a, \mathrm{refl}_a) \equiv q$

Identity type: Equality as structure

- Using path induction, we can show that the identity really behaves like equality should.
- We have postulated a *reflexivity* function $\mathrm{refl} : \prod_{x:A} (x =_A x)$.
- **Goal:** Want to define *symmetry/inversion* $\mathrm{inv} : \prod_{x,y:A} (x =_A y) \to (y =_A x)$ and *transitivity/composition* $\mathrm{comp} : \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$.

Identity type: Inversion

Theorem (Inversion for the identity type)

Let A be a type. There is a function $\mathrm{inv} : \prod_{x,y:A} (x =_A y) \to (y =_A x)$.

Proof.

We want to produce a function depending on $x, y : A$ and $p : (x =_A y)$, landing in $B(x, y, p) :\equiv (y =_A x)$. By path induction, it suffices to assume $x \equiv y$ and $p \equiv \mathrm{refl}_x$. We have the function $f :\equiv \lambda x.\mathrm{refl}_x : \prod_{x:A}(x =_A x)$. Thus, we take

$$\mathrm{inv} :\equiv \mathrm{ind}_{=_A}(f) : \prod_{x,y:A} \prod_{p:(x=_A y)} (y =_A x).$$

□

Types as groupoids

- Define composition $\mathrm{comp} : \prod_{x,y,z:A} (x =_A y) \to (y =_A z) \to (x =_A z)$ similarly by path induction.
- Let us abbreviate $p^{-1} :\equiv \mathrm{inv}_{x,y}(p)$ and $p * q :\equiv \mathrm{comp}_{x,y,z}(p,q)$.
- One can also show some expected laws, namely **associativity**, **neutrality**, and **inversion**, *e.g.*

  $$(p * q) * r =_{(x =_A z)} p * (q * r), \quad \mathrm{refl}_y * p =_{(x =_A y)} p, \quad p^{-1} * p =_{(x =_A x)} \mathrm{id}_x, \ldots$$
- These are known as the *groupoid laws*.
- They arise as *higher identities/homotopies* $\varphi : p =_{(x =_A y)} q$ for $p, q : (x =_A y)$:

$$\begin{tikzcd} x \arrow[r, bend left, equal, "p"] \arrow[r, bend right, equal, "q"'] \arrow[r, phantom, "\varphi \, \Vert"] & y \end{tikzcd}$$

Types as weak ∞-groupoids

- Thus, via the dependent identity type, any type A can be endowed with the structure of a **groupoid**. $\rightsquigarrow$ Martin Hofmann and Thomas Streicher's groupoid model with non-trivial identity types (1994).
- But any identity type $(x =_A y)$ is in particular a type and hence carries itself a groupoid structure.
- Moreover, all these ensuing groupoid laws hold only in the *propositional sense*, *i.e.* not definitionally but only up to higher paths, in arbitrarily high dimensions.

  $\rightsquigarrow$ types as *weak ∞-groupoids* (conj. Hofmann–Streicher, made more precise semantically in 2006 by Voevodsky and Streicher)
- More on this in Léonard's talk next Monday, March 21!

The Curry–Howard–Voevodsky interpretation[2]

Type theory	Logic	Set theory	Homotopy theory
A	proposition	set	space/homotopy type
$x : A$	witness/realizer	element	point
$\mathbf{0}, \mathbf{1}$	$\bot, \top$	$\emptyset, \{\emptyset\}$	$\emptyset, *$
$A + B$	$A \vee B$	disjoint union	coproduct space
$A \times B$	$A \wedge B$	set of ordered pairs	product space
$A \to B$	$A \Rightarrow B$	set of functions	function space
$x : A \vdash B(x)$	predicate $B(x)$	family of sets	fibration
$x : A \vdash b : B(x)$	conditional proof	choice of elements	section
$\Sigma_{x:A} B(x)$	$\exists\, x.B(x)$	disjoint sum	total space
$\Pi_{x:A} B(x)$	$\forall\, x.B(x)$	product	space of sections
$p : (x =_A y)$	$x = y$	$x = y$	path $x \rightsquigarrow y$ in A

[2] Table based on: Emily Riehl *The synthetic theory of ∞-categories vs the synthetic theory of ∞-categories*, Presentation at Vladimir Voevodsky Memorial Conference, IAS, Princeton, NJ, USA, 2018.

Dependent types as fibrations

- Identity proofs $p : (x =_A y)$ are a kind of “path”.
- Indeed, dependent types behave well w.r.t. paths in the base.
- Namely, every dependent type supports a notion of **path transport**.
- For $a : A \vdash B$, we can define a function $\mathrm{tr}_A : \prod_{x,y:A} \prod_{p:(x=_A y)} B(x) \to B(y)$.
- Again by path induction, with $\mathrm{tr}_B(x, x, \mathrm{refl}_x) :\equiv \mathrm{id}_{B(x)}$.
- Indeed, this is connected with a synthetic notion of **path lifting**:

$$
\begin{tikzcd}
\sum_{a:A} B(a) \arrow[d] & d \arrow[r, dashed, no head, "\mathrm{lift}(p,d)"] & \mathrm{tr}_B(d) \\
A & x \arrow[r, equal, "p"] & y
\end{tikzcd}
$$

Homotopical interpretations of MLTT?

- Types as weak ∞-groupoids (precursor: Hofmann–Streicher; established by Voevodsky, Kapulkin–Lumsdaine during 2006–2012)
- Can model MLTT in “abstract homotopy theories”, with identity types as path space fibrations (Awodey–Warren 2006)
- Syntactic structure groupoid and factorization structures from id types (van den Berg, Garner, Gambino, Lumsdaine mid-2000s)
- Voevodsky ’06: **Univalence Axiom (UA)** giving rise to **homotopy type theory (HoTT)** as an extension of MLTT
- Univalence identifies equality of types with equivalence
- What does this mean, more precisely?

Weak equivalences

- What is an appropriate notion for equivalence between types? It is a kind of notion of isomorphism.
- Let $f : A \to B$ be a function between types. We say that f is a *weak equivalence* (after Voevodsky) if the following type is inhabited:

$$\mathrm{isWeq}(f) :\equiv \Big(\sum_{g:B\to A} \prod_{x:A} (g \circ f)(x) = x \Big) \times \Big(\sum_{h:B\to A} \prod_{y:B} (f \circ h)(y) = y \Big)$$

- *A priori*, it looks as if being a weak equivalence is *structure* rather than a *property*. But one can show that it actually is just a property (more later).
- We can define the type of equivalences from A to B as

$$(A \simeq B) :\equiv \sum_{f:A\to B} \mathrm{isWeq}(f).$$

Universe and univalence

- How else could two types be equal?
- We postulate the existence of a ("large") type $\mathcal{U}$ of all ("small") types, *i.e.*: If A is a ("small") type, then $A : \mathcal{U}$.
- This will be convenient because it allows us to identify ($\mathcal{U}$-small) dependent types $a : A \vdash B(a)$ with families $B : A \to \mathcal{U}$.
- Since $\mathcal{U}$ is a (large) type, there exists the identity type $(A =_{\mathcal{U}} B)$ for $A, B : \mathcal{U}$.
- There is a map $\mathrm{idToWeq}_{A,B} : (A =_{\mathcal{U}} B) \to (A \simeq B)$ defined by path induction (mapping $\mathrm{refl}_A : (A =_{\mathcal{U}} A)$ to $\mathrm{id}_A : (A \simeq A)$).
- Voevodsky's Univalence Axiom states that $\mathrm{idToWeq}$ is an equivalence, thus "equivalence is equivalent to equality":

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B)$$

Consequences of univalence

- Why is univalence useful or desirable?
- **Function extensionality:** For $x : A \vdash B(x) : \mathcal{U}$ and $f, g : \prod_{x:A} B(x)$, we have
  $$(f = g) \simeq \prod_{x:A} (f(x) =_B g(x)).$$
- **Fibrations are families:** For a type $A : \mathcal{U}$, we have $\sum_{E:\mathcal{U}} (E \to A) \simeq (A \to \mathcal{U})$.
- **Univalent foundations:** Isomorphism-invariant foundations of mathematics (unlike set theory which is sensitive to encoding)
- **Structure identity principles:** Find out more in Paige's talk in two weeks, Monday, March 28!

Homotopy fibers and type families

- Let $f : A \to B$ be a map between types. Voevodsky defines the *homotopy fiber* at $y : B$ as the type
$$\mathrm{fib}(f, y) :\equiv \sum_{x:A} f(x) =_B y.$$
- Using univalence, one can show that every map $f : A \to B$ is equivalent to the 1st coordinate projection $\mathrm{pr}_1 : \left(\sum_{b:B} \mathrm{fib}(f, b)\right) \to A$ (*fibrant replacement*).
- Converting between a map into B and a family over B is given by considering the **family of fibers** or, resp., the *associated* projection:

$$\begin{tikzcd} \left(\sum_{E:\mathcal{U}} E \to B\right) \arrow[rr, bend left, "\lambda E,f.\lambda b.\mathrm{fib}(f,b)"] \arrow[rr, phantom, "\simeq"] & & (B \to \mathcal{U}) \arrow[ll, bend left, "\lambda P.\pi_P"] \end{tikzcd}$$

where $\pi_P :\equiv \mathrm{pr}_1 : \left(\sum_{b:B} P(b)\right) \to B$. This, again, uses univalence.

Homotopy levels

Voevodsky defines the following hierarchy of homotopy levels:

- A type A is *contractible* or a (-2)*-type* if we have an inhabitant

$$\mathrm{isContr}(A) :\equiv \sum_{x:A} \prod_{y:A} (x =_A y).$$

- A type A is a *proposition* or a (-1)*-type* if we have an inhabitant

$$\mathrm{isProp}(A) :\equiv \prod_{x,y:A} \mathrm{isContr}(x =_A y).$$

- A type A is a *set* or a 0*-type* if we have an inhabitant

$$\mathrm{isSet}(A) :\equiv \prod_{x,y:A} \mathrm{isProp}(x =_A y).$$

- In general: For $n \geq 1$, a type A is an n*-type* if we have an inhabitant

$$\text{is-}n\text{-type}(A) :\equiv \prod_{x,y:A} \text{is-}(n-1)\text{-type}(x =_A y).$$

Propositions and weak equivalences

- Propositions are important because their inhabitants are determined uniquely up to homotopy.

  $\rightsquigarrow$ mere properties (up to homotopy) rather than structure.
- Examples: $\text{is-}n\text{-type}(A)$, $\text{isWeq}(f)$, ...
- Voevodsky initially defined a weak equivalence $f : A \to B$ such that all its fibers are contractible:

$$\prod_{b:B} \text{isContr}(\text{fib}(b, f))$$

- This type is again a proposition, and it is equivalent to our previous definition (bi-invertibility).

The circle $\mathbf{S}^1$ in HoTT: Idea

- The circle $\mathbf{S}^1$ will be defined as the “free type” equipped with a base point $b : \mathbf{S}^1$ and a loop $\ell : b = b$, satisfying an appropriate *induction principle*.
- *Intuition:* Let $P : \mathbf{S}^1 \to \mathcal{U}$ be a family. Then, given an element $y : P(b)$ and a path $p : y =_\ell y$ over ℓ, this induces a *section* $f : \prod_{x:\mathbf{S}^1} P(x)$ (plus computational properties).

Synthetic homotopy groups

- Proposed alternative, more structuralist foundations...
- ...that even have homotopical meaning.
- Basic entities are homotopy types rather than bare sets.
- Define objects by “universal properties” via typing rules $\rightsquigarrow$ independence from coding.
- More conceptual proofs, *e.g.* in homotopy theory.
- More general statements: models are “abstract homotopy theories” (Awodey–Warren ’06, Shulman ’19, . . .)
- Plus: can be verified on a computer.

References

- S. Awodey (2010): *Type theory and homotopy*. arXiv:1010.1810
- D. R. Grayson (2018): *An introduction to univalent foundations for mathematicians*. Bull. Am. Math. Soc., New Ser. 55, No. 4, 427–450. arXiv:1711.01477
- The HoTT-UF Project (2022): *Symmetry*. CAS Oslo. Forthcoming book (github)
- E. Riehl (2021): *Math 721: Homotopy type theory* Lecture notes (PDF)
- E. Rijke (2022): *Introduction to Homotopy Type Theory* Forthcoming book for CUP (github)
- M. Shulman (2021): *Homotopy type theory: the logic of space*. In . CUP. 322-403. arXiv:1703.03007
- The Univalent Foundations Program (2013): *Homotopy Type Theory: Univalent Foundations of Mathematics*. IAS Princeton `https://homotopytypetheory.org/book`

Thank you!