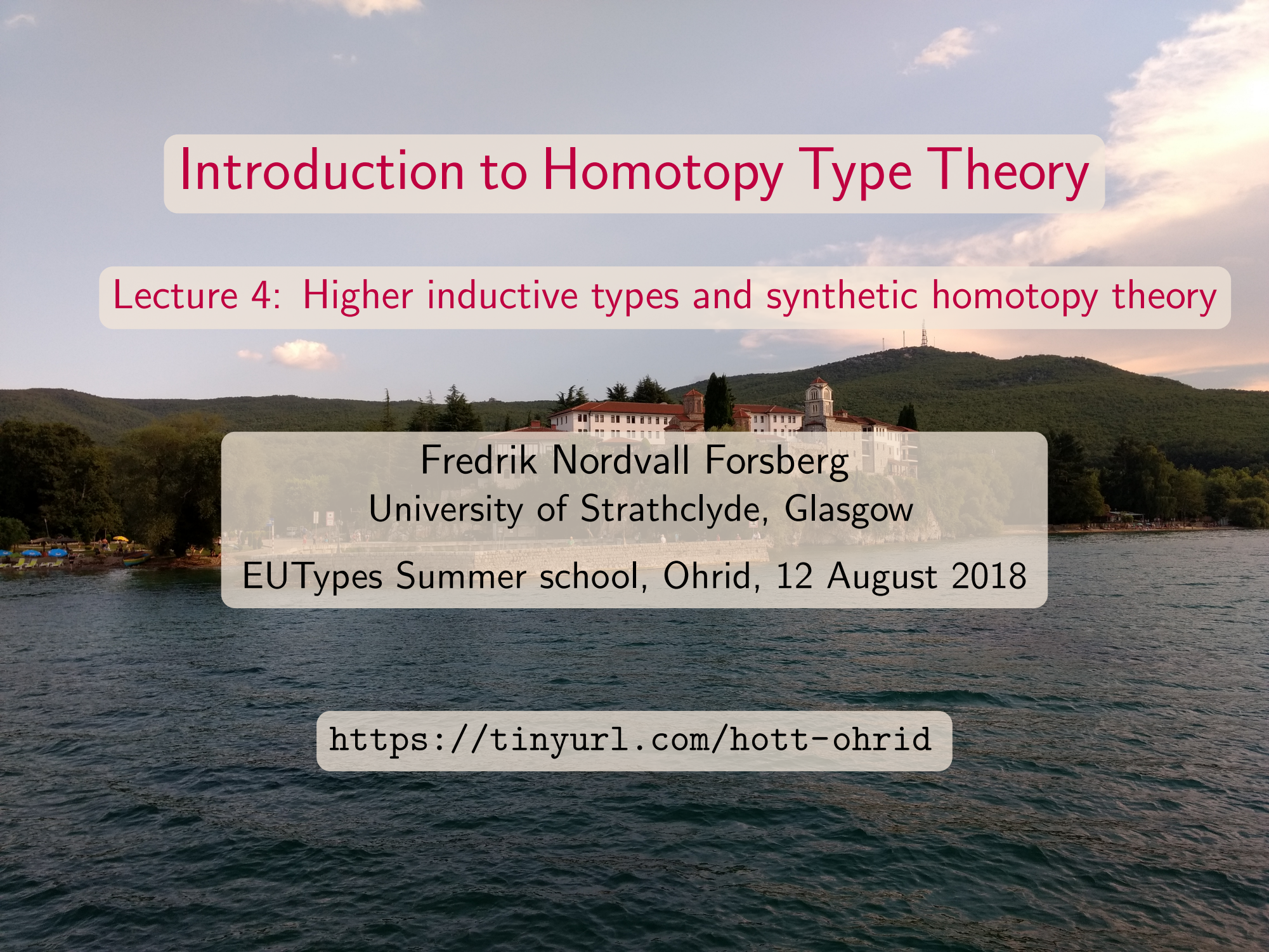

Introduction to Homotopy Type Theory

Lecture 4: Higher inductive types and synthetic homotopy theory

Fredrik Nordvall Forsberg
University of Strathclyde, Glasgow

EUTypes Summer school, Ohrid, 12 August 2018

`https://tinyurl.com/hott-ohrid`

Outline

1. Higher inductive types

2. Synthetic homotopy theory

Types with higher-dimensional structure

The idea with HoTT is to be able to manipulate equalities between equalities, and higher dimensional object.

Sometimes we have to work quite hard to do so, e.g. proving that a construction is coherent.

Types with higher-dimensional structure

The idea with HoTT is to be able to manipulate equalities between equalities, and higher dimensional object.

Sometimes we have to work quite hard to do so, e.g. proving that a construction is coherent.

However, do you actually know of a higher-dimensional type?

Types with higher-dimensional structure

The idea with HoTT is to be able to manipulate equalities between equalities, and higher dimensional object.

Sometimes we have to work quite hard to do so, e.g. proving that a construction is coherent.

However, do you actually know of a higher-dimensional type?

Okay, the universe has multiple parallel paths because of the Univalence Axiom, but what about a small type?

All standard types are sets

Theorem ([Coquand, Danielsson 2013])

All basic small types such as $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$ *and* $\mathbb{N}$ *are sets, and all constructions on small types preserve being a set, i.e. if* $A : \mathsf{Set}$ *and* $B(a) : \mathsf{Set}$ *for all* $a : A$*, then*

$$(\Sigma\ A\ B), (\Pi\ A\ B), (x =_A y), (\mathsf{W}\ A\ B) : \mathsf{Set}$$

(using function extensionality).

All standard types are sets

Theorem ([Coquand, Danielsson 2013])

All basic small types such as $\mathbf{0}$, $\mathbf{1}$, $\mathbf{2}$ *and* $\mathbb{N}$ *are sets, and all constructions on small types preserve being a set, i.e. if* $A : \mathrm{Set}$ *and* $B(a) : \mathrm{Set}$ *for all* $a : A$*, then*

$$(\Sigma\ A\ B), (\Pi\ A\ B), (x =_A y), (\mathrm{W}\ A\ B) : \mathrm{Set}$$

(using function extensionality).

Mismatch with homotopical models: there are plenty of small spaces (e.g. the circle, the torus, . . .) with non-trivial path spaces!

Higher inductive types: inductively generating paths

This, together with viewing types as coming with their path spaces, suggests that we should be allowed to inductively generate not only points, but also paths, and paths between paths, and ...

Higher inductive types: inductively generating paths

This, together with viewing types as coming with their path spaces, suggests that we should be allowed to inductively generate not only points, but also paths, and paths between paths, and ...

Together with the Univalence Axiom, Higher Inductive Types is the main addition of HoTT.

Higher inductive types: inductively generating paths

This, together with viewing types as coming with their path spaces, suggests that we should be allowed to inductively generate not only points, but also paths, and paths between paths, and . . .

Together with the Univalence Axiom, Higher Inductive Types is the main addition of HoTT.

Not used in UniMath — for ease of semantics, and philosophical reasons.

An Inductive Type: binary trees

The type BTree_A of binary trees is given by the constructors

$$\begin{aligned} \mathsf{leaf} &: \mathsf{BTree}_A \\ \mathsf{node} &: A \to \mathsf{BTree}_A \to \mathsf{BTree}_A \to \mathsf{BTree}_A \end{aligned}$$

An Inductive Type: binary trees

The type BTree_A of binary trees is given by the constructors

$$\begin{aligned} \mathsf{leaf} &: \mathsf{BTree}_A \\ \mathsf{node} &: A \to \mathsf{BTree}_A \to \mathsf{BTree}_A \to \mathsf{BTree}_A \end{aligned}$$

An inductive definition, meaning that we intend BTree_A to be the smallest type closed under the constructors.

Functions out of binary trees

Functions out of binary trees

To define $f : (\Pi x : \mathrm{BTree}_A) P(x)$ for $P : \mathrm{BTree}_A \to \mathcal{U}$, we need to make $P(x)$ into a "BTree_A-algebra" by giving

$$\begin{aligned} \mathsf{leaf}' &: P(\mathsf{leaf}) \\ \mathsf{node}' &: (\Pi a : A)(\Pi l' : P(l))(\Pi r' : P(r))\big(P(\mathsf{node}\, a\, l\, r)\big) \end{aligned}$$

Functions out of binary trees

To define $f : (\Pi x : \mathrm{BTree}_A) P(x)$ for $P : \mathrm{BTree}_A \to \mathcal{U}$, we need to make $P(x)$ into a "BTree_A-algebra" by giving

$$\begin{aligned} \mathsf{leaf}' &: P(\mathsf{leaf}) \\ \mathsf{node}' &: (\Pi a : A)(\Pi l' : P(l))(\Pi r' : P(r))\big(P(\mathsf{node}\, a\, l\, r)\big) \end{aligned}$$

The function will satisfy the equations

$$\begin{aligned} f\,\mathsf{leaf} &\equiv \mathsf{leaf}' : P(\mathsf{leaf}) \\ f\,(\mathsf{node}\, a\, l\, r) &\equiv \mathsf{node}'\, a\, f(l)\, f(r) \end{aligned}$$

Functions out of binary trees

To define $f : (\Pi x : \mathrm{BTree}_A) P(x)$ for $P : \mathrm{BTree}_A \to \mathcal{U}$, we need to make $P(x)$ into a "BTree_A-algebra" by giving

$$\begin{aligned} \mathsf{leaf}' &: P(\mathsf{leaf}) \\ \mathsf{node}' &: (\Pi a : A)(\Pi l' : P(l))(\Pi r' : P(r))\big(P(\mathsf{node}\, a\, l\, r)\big) \end{aligned}$$

The function will satisfy the equations

$$\begin{aligned} f\,\mathsf{leaf} &\equiv \mathsf{leaf}' : P(\mathsf{leaf}) \\ f\,(\mathsf{node}\, a\, l\, r) &\equiv \mathsf{node}'\, a\, f(l)\, f(r) \end{aligned}$$

We will often write the function in such pattern matching style directly.

The depth of a binary tree

Choose $P(x) :\equiv \mathbb{N}$. Define $\mathsf{depth} : \mathsf{BTree}_A \to \mathbb{N}$ by

$$\mathsf{depth}\,\mathsf{leaf} :\equiv \{?_0 : \mathbb{N}\}$$

$$\mathsf{depth}\,(\mathsf{node}\,a\,l\,r) :\equiv \{?_1 : \mathbb{N}\}$$

The depth of a binary tree

Choose $P(x) :\equiv \mathbb{N}$. Define $\mathsf{depth} : \mathsf{BTree}_A \to \mathbb{N}$ by

$$\mathsf{depth}\,\mathsf{leaf} :\equiv 0$$

$$\mathsf{depth}\,(\mathsf{node}\,a\,l\,r) :\equiv \{?_1 : \mathbb{N}\}$$

The depth of a binary tree

Choose $P(x) :\equiv \mathbb{N}$. Define $\mathsf{depth} : \mathsf{BTree}_A \to \mathbb{N}$ by

$$\begin{aligned} \mathsf{depth}\,\mathsf{leaf} &:\equiv 0 \\ \mathsf{depth}\,(\mathsf{node}\,a\,l\,r) &:\equiv 1 + \max(\mathsf{depth}\,l)\,(\mathsf{depth}\,r) \end{aligned}$$

Important aspects of inductive types

- Freely generated by its constructors
- Functions out of the type defined by specifying its values on all constructors
- Formally: formation, introduction, elimination and computation rules.

Important aspects of inductive types

- Freely generated by its constructors
- Functions out of the type defined by specifying its values on all constructors
- Formally: formation, introduction, elimination and computation rules.

Generalising to Higher Inductive Types: Allow also paths to be freely generated by constructors.

Propositional truncation

For a type A, the propositional truncation $\|A\|$ is given by the constructors

$$|-| : A \to \|A\|$$
$$\mathsf{irr} : \Pi(x, y : \|A\|)(x =_{\|A\|} y)$$

Propositional truncation

For a type A, the propositional truncation $\|A\|$ is given by the constructors

$$\begin{aligned} |-| &: A \to \|A\| \\ \mathsf{irr} &: \Pi(x, y : \|A\|)(x =_{\|A\|} y) \end{aligned}$$

To define a function $f : \|A\| \to B$, need to give

$$\begin{aligned} |-|' &: A \to B \\ \mathsf{irr}' &: (\Pi x', y' : B)(x' =_B y') \end{aligned}$$

Propositional truncation

For a type A, the propositional truncation $\|A\|$ is given by the constructors

$$\begin{aligned} |-| &: A \to \|A\| \\ \mathsf{irr} &: \Pi(x, y : \|A\|)(x =_{\|A\|} y) \end{aligned}$$

To define a function $f : \|A\| \to B$, need to give

$$\begin{aligned} |-|' &: A \to B \\ \mathsf{irr}' &: (\Pi x', y' : B)(x' =_B y') \end{aligned}$$

The function will satisfy the equations

$$\begin{aligned} f\,(|a|) &\equiv |a|' : B \\ \mathsf{ap}_f\,(\mathsf{irr}\,x\,y) &\equiv \mathsf{irr}'\,(f\,x)\,(f\,y) : f\,x =_B f\,y \end{aligned}$$

Set quotients

For $A : \mathrm{Set}$ and $R : A \to A \to \mathrm{Prop}$, we define $A/R : \mathrm{Set}$ by the constructors

$$
\begin{aligned}
[-] &: A \to A/R \\
\mathrm{squash} &: (\Pi x, y : A)(R\,x\,y \to [x] =_{A/R} [y]) \\
\mathrm{set} &: (\Pi x, y : A/R)(\Pi p, q : x =_{A/R} y)(p =_{x =_{A/R} y} q)
\end{aligned}
$$

Set quotients

For $A : \mathrm{Set}$ and $R : A \to A \to \mathrm{Prop}$, we define $A/R : \mathrm{Set}$ by the constructors

$$
\begin{aligned}
[-] &: A \to A/R \\
\mathrm{squash} &: (\Pi x, y : A)(R\,x\,y \to [x] =_{A/R} [y]) \\
\mathrm{set} &: (\Pi x, y : A/R)(\Pi p, q : x =_{A/R} y)(p =_{x =_{A/R} y} q)
\end{aligned}
$$

We need to set constructor to force the type to again be a set.

Set quotients

For $A : \mathrm{Set}$ and $R : A \to A \to \mathrm{Prop}$, we define $A/R : \mathrm{Set}$ by the constructors

$$
\begin{aligned}
[-] &: A \to A/R \\
\mathrm{squash} &: (\Pi x, y : A)(R\,x\,y \to [x] =_{A/R} [y]) \\
\mathrm{set} &: (\Pi x, y : A/R)(\Pi p, q : x =_{A/R} y)(p =_{x =_{A/R} y} q)
\end{aligned}
$$

We need to set constructor to force the type to again be a set.

Note that R might not be an equivalence relation, but equality on A/R will be — we really quotient by the equivalence closure of R.

Set quotients

For $A : \mathrm{Set}$ and $R : A \to A \to \mathrm{Prop}$, we define $A/R : \mathrm{Set}$ by the constructors

$$\begin{aligned} [-] &: A \to A/R \\ \mathrm{squash} &: (\Pi x, y : A)(R\,x\,y \to [x] =_{A/R} [y]) \\ \mathrm{set} &: (\Pi x, y : A/R)(\Pi p, q : x =_{A/R} y)(p =_{x=_{A/R}y} q) \end{aligned}$$

We need to set constructor to force the type to again be a set.

Note that R might not be an equivalence relation, but equality on A/R will be — we really quotient by the equivalence closure of R.

Example: Define $R : (\mathbb{N} \times \mathbb{N}) \to (\mathbb{N} \times \mathbb{N}) \to \mathrm{Prop}$ by

$$R\,(x, y)\,(x', y') :\equiv x + y' =_{\mathbb{N}} x' + y$$

and $\mathbb{Z} \equiv (\mathbb{N} \times \mathbb{N})/R$.

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

$$\begin{aligned}
\mathsf{d}\,\mathsf{leaf} &:\equiv \{?_0 : \mathbb{N}\} \\
\mathsf{d}\,(\mathsf{node}\, a\, l\, r) &:\equiv \{?_1 : \mathbb{N}\} \\
\mathsf{ap}_{\mathsf{d}}\,(\mathsf{perm}\, a\, l\, r) &:\equiv \{?_2 : \mathsf{d}\,(\mathsf{node}\, a\, l\, r) = \mathsf{d}\,(\mathsf{node}\, a\, r\, l)\}
\end{aligned}$$

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

$$\begin{aligned}
\mathsf{d}\,\mathsf{leaf} &:\equiv 0 \\
\mathsf{d}\,(\mathsf{node}\, a\, l\, r) &:\equiv \{?_1 : \mathbb{N}\} \\
\mathsf{ap}_{\mathsf{d}}\,(\mathsf{perm}\, a\, l\, r) &:\equiv \{?_2 : \mathsf{d}\,(\mathsf{node}\, a\, l\, r) = \mathsf{d}\,(\mathsf{node}\, a\, r\, l)\}
\end{aligned}$$

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

$$\begin{aligned}
\mathsf{d}\,\mathsf{leaf} &:\equiv 0 \\
\mathsf{d}\,(\mathsf{node}\, a\, l\, r) &:\equiv 1 + \max(\mathsf{d}\, l)(\mathsf{d}\, r) \\
\mathsf{ap}_{\mathsf{d}}\,(\mathsf{perm}\, a\, l\, r) &:\equiv \{?_2 : 1 + \max(\mathsf{d}\, l)(\mathsf{d}\, r) = 1 + \max(\mathsf{d}\, r)(\mathsf{d}\, l)\}
\end{aligned}$$

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

$$\begin{aligned}
\mathsf{d}\,\mathsf{leaf} &:\equiv 0 \\
\mathsf{d}\,(\mathsf{node}\, a\, l\, r) &:\equiv 1 + \max(\mathsf{d}\, l)(\mathsf{d}\, r) \\
\mathsf{ap}_{\mathsf{d}}\,(\mathsf{perm}\, a\, l\, r) &:\equiv \mathsf{ap}_{1+_}\ \{?_3 : \max(\mathsf{d}\, l)(\mathsf{d}\, r) = \max(\mathsf{d}\, r)(\mathsf{d}\, l)\}
\end{aligned}$$

Binary trees without sibling order

The type PBTree_A is given by the constructors

$$\begin{aligned}
\mathsf{leaf} &: \mathsf{PBTree}_A \\
\mathsf{node} &: A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \to \mathsf{PBTree}_A \\
\mathsf{perm} &: (\Pi a : A)(\Pi l, r : \mathsf{PBTree}_A)(\mathsf{node}\, a\, l\, r =_{\mathsf{PBTree}_A} \mathsf{node}\, a\, r\, l)
\end{aligned}$$

$$\begin{aligned}
\mathsf{d}\,\mathsf{leaf} &:\equiv 0 \\
\mathsf{d}\,(\mathsf{node}\, a\, l\, r) &:\equiv 1 + \mathsf{max}\,(\mathsf{d}\, l)\,(\mathsf{d}\, r) \\
\mathsf{ap}_{\mathsf{d}}\,(\mathsf{perm}\, a\, l\, r) &:\equiv \mathsf{ap}_{1+_}\,(\mathsf{max} - \mathsf{commutative}\,(\mathsf{d}\, l)\,(\mathsf{d}\, r))
\end{aligned}$$

The interval

The type I is given by the constructors

$$0 : \mathsf{I}$$

$$1 : \mathsf{I}$$

$$\mathsf{seg} : 0 =_{\mathsf{I}} 1$$

The interval

The type I is given by the constructors

$$0 : \mathsf{I}$$
$$1 : \mathsf{I}$$
$$\mathsf{seg} : 0 =_{\mathsf{I}} 1$$

A function $f : (\Pi x : \mathsf{I})P(x)$ is defined by giving

$$0' : P(0)$$
$$1' : P(1)$$
$$\mathsf{seg}' : 0' \underset{\mathsf{seg}}{=} 1'$$

The interval is contractible

Theorem

The interval $\mathbf{I}$ *is contractible.*

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0.

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\mathsf{refl} : P(0) \equiv 0 = 0$$

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\mathsf{refl} : P(0) \equiv 0 = 0$$
$$\mathsf{seg} : P(1) \equiv 0 = 1$$

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\mathsf{refl} : P(0) \equiv 0 = 0$$

$$\mathsf{seg} : P(1) \equiv 0 = 1$$

$$\{?\} : \mathsf{refl} \underset{\mathsf{seg}}{=} \mathsf{seg}$$

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\mathsf{refl} : P(0) \equiv 0 = 0$$

$$\mathsf{seg} : P(1) \equiv 0 = 1$$

$$\{?\} : \mathsf{seg} \cdot \mathsf{refl} = \mathsf{seg}$$

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\mathsf{refl} : P(0) \equiv 0 = 0$$

$$\mathsf{seg} : P(1) \equiv 0 = 1$$

$$\mathsf{runit}\,\mathsf{seg} : \mathsf{seg} \cdot \mathsf{refl} = \mathsf{seg}$$

□

The interval is contractible

Theorem

The interval I *is contractible.*

Proof.

As centre of contraction we choose 0. We show $(\Pi y : \mathsf{I})(0 =_{\mathsf{I}} y)$ by induction on y, i.e. we choose $P(y) \equiv 0 =_{\mathsf{I}} y$:

$$\begin{aligned} \mathsf{refl} &: P(0) \equiv 0 = 0 \\ \mathsf{seg} &: P(1) \equiv 0 = 1 \\ \mathsf{runit}\,\mathsf{seg} &: \mathsf{seg} \cdot \mathsf{refl} = \mathsf{seg} \end{aligned}$$

□

Hence $\mathsf{I} \simeq \mathbf{1}$. But the interval has surprising consequences:

Theorem ([Shulman, 2011; Hofmann, 1995])

Assuming the computation rules for the interval implies function extensionality.

The circle

The type $\mathbb{S}^1$ is given by the constructors

$$\mathsf{base} : \mathbb{S}^1$$

$$\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$$

The circle

The type $\mathbb{S}^1$ is given by the constructors

$$\mathsf{base} : \mathbb{S}^1$$
$$\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$$

A function $f : (\Pi x : \mathbb{S}^1) P(x)$ is defined by giving

$$\mathsf{base}' : P(\mathsf{base})$$
$$\mathsf{loop}' : \mathsf{base}' \underset{\mathsf{loop}}{=} \mathsf{base}'$$

Note: not just $\mathsf{loop}' : \mathsf{base}' =_{P(\mathsf{base})} \mathsf{base}'$!

The circle

The type $\mathbb{S}^1$ is given by the constructors

$$\mathsf{base} : \mathbb{S}^1$$
$$\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$$

A function $f : (\Pi x : \mathbb{S}^1) P(x)$ is defined by giving

$$\mathsf{base}' : P(\mathsf{base})$$
$$\mathsf{loop}' : \mathsf{base}' \underset{\mathsf{loop}}{=} \mathsf{base}'$$

Note: not just ~~$\mathsf{loop}' : \mathsf{base}' =_{P(\mathsf{base})} \mathsf{base}'$~~!

The circle

The type $\mathbb{S}^1$ is given by the constructors

$$\mathsf{base} : \mathbb{S}^1$$
$$\mathsf{loop} : \mathsf{base} =_{\mathbb{S}^1} \mathsf{base}$$

A function $f : (\Pi x : \mathbb{S}^1) P(x)$ is defined by giving

$$\mathsf{base}' : P(\mathsf{base})$$
$$\mathsf{loop}' : \mathsf{base}' \underset{\mathsf{loop}}{=} \mathsf{base}'$$

Note: not just ~~$\mathsf{loop}' : \mathsf{base}' =_{P(\mathsf{base})} \mathsf{base}'$~~!

Even though there is only one point in sight, this type is not contractible.

The sphere

The type $\mathbb{S}^2$ is given by the constructors

$$\mathsf{base} : \mathbb{S}^2$$

$$\mathsf{surface} : \mathsf{refl}_{\mathsf{base}} =_{\mathsf{base}=\mathsf{base}} \mathsf{refl}_{\mathsf{base}}$$

The sphere

The type $\mathbb{S}^2$ is given by the constructors

$$\begin{aligned} \mathsf{base} &: \mathbb{S}^2 \\ \mathsf{surface} &: \mathsf{refl}_{\mathsf{base}} =_{\mathsf{base}=\mathsf{base}} \mathsf{refl}_{\mathsf{base}} \end{aligned}$$

Alternatively by

$$\begin{aligned} \mathsf{north} &: \mathbb{S}^2 \\ \mathsf{south} &: \mathbb{S}^2 \\ \mathsf{greenwich} &: \mathsf{north} = \mathsf{south} \\ \mathsf{dateline} &: \mathsf{north} = \mathsf{south} \\ \mathsf{east} &: \mathsf{greenwich} = \mathsf{dateline} \\ \mathsf{west} &: \mathsf{greenwich} = \mathsf{dateline} \end{aligned}$$

The sphere

The type $\mathbb{S}^2$ is given by the constructors

$$\begin{aligned} \mathsf{base} &: \mathbb{S}^2 \\ \mathsf{surface} &: \mathsf{refl}_{\mathsf{base}} =_{\mathsf{base}=\mathsf{base}} \mathsf{refl}_{\mathsf{base}} \end{aligned}$$

Alternatively by

$$\begin{aligned} \mathsf{north} &: \mathbb{S}^2 \\ \mathsf{south} &: \mathbb{S}^2 \\ \mathsf{greenwich} &: \mathsf{north} = \mathsf{south} \\ \mathsf{dateline} &: \mathsf{north} = \mathsf{south} \\ \mathsf{east} &: \mathsf{greenwich} = \mathsf{dateline} \\ \mathsf{west} &: \mathsf{greenwich} = \mathsf{dateline} \end{aligned}$$

The type $\mathbb{S}^2$ is believed not to have a finite homotopy level.

The torus

The type $\mathbb{T}^2$ is given by the constructors

$$\begin{aligned} \mathsf{base} &: \mathbb{T}^2 \\ \mathsf{vertLoop} &: \mathsf{base} = \mathsf{base} \\ \mathsf{horLoop} &: \mathsf{base} = \mathsf{base} \\ \mathsf{surface} &: \mathsf{vertLoop} \cdot \mathsf{horLoop} = \mathsf{horLoop} \cdot \mathsf{vertLoop} \end{aligned}$$

The torus

The type $\mathbb{T}^2$ is given by the constructors

$$\begin{aligned} \mathsf{base} &: \mathbb{T}^2 \\ \mathsf{vertLoop} &: \mathsf{base} = \mathsf{base} \\ \mathsf{horLoop} &: \mathsf{base} = \mathsf{base} \\ \mathsf{surface} &: \mathsf{vertLoop} \cdot \mathsf{horLoop} = \mathsf{horLoop} \cdot \mathsf{vertLoop} \end{aligned}$$

Theorem ([Sojakova 2016])
$\mathbb{T}^2 \simeq \mathbb{S}^1 \times \mathbb{S}^1$.

Status in implementations

Currently no system fully implements HITs. There are some workarounds to use them anyway:

Status in implementations

Currently no system fully implements HITs. There are some workarounds to use them anyway:

Coq Postulate everything; necessarily no definitional computation rules.

Status in implementations

Currently no system fully implements HITs. There are some workarounds to use them anyway:

Coq Postulate everything; necessarily no definitional computation rules.

Agda Define the point constructors as an inductive type, then postulate the path constructors. Define eliminator on point constructors, then postulate computation rules for path constructors. Use `REWRITE` to make them definitional.

Status in implementations

Currently no system fully implements HITs. There are some workarounds to use them anyway:

Coq Postulate everything; necessarily no definitional computation rules.

Agda Define the point constructors as an inductive type, then postulate the path constructors. Define eliminator on point constructors, then postulate computation rules for path constructors. Use `REWRITE` to make them definitional.

Cubicaltt Has builtin support for basic HITs (but not e.g. recursive HITs with parameters).

Synthetic homotopy theory: what does it mean?

Derive results in the model by working in the syntax.

Synthetic homotopy theory: what does it mean?

Derive results in the model by working in the syntax.

The point: in the syntax, e.g. a path is a basic object, not a function from the interval $[0, 1]$ — less baggage.

Synthetic homotopy theory: what does it mean?

Derive results in the model by working in the syntax.

The point: in the syntax, e.g. a path is a basic object, not a function from the interval $[0, 1]$ — less baggage.

If we for example prove isContr(A), then $[\![A]\!]$ will be a contractible space in the model.

Synthetic homotopy theory: what does it mean?

Derive results in the model by working in the syntax.

The point: in the syntax, e.g. a path is a basic object, not a function from the interval $[0, 1]$ — less baggage.

If we for example prove isContr(A), then $[\![A]\!]$ will be a contractible space in the model.

Perhaps more interest from type theorists than homotopy theorists; Jacob Lurie: "No comment".

Homotopy groups

A typical goal: try to compute the homotopy groups of a space A.

Homotopy groups

A typical goal: try to compute the homotopy groups of a space A.

Loop spaces of a pointed space (A, a):

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega^n(a =_A a, \mathsf{refl})$$

Homotopy groups

A typical goal: try to compute the homotopy groups of a space A.

Loop spaces of a pointed space (A, a):

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega^n(a =_A a, \mathsf{refl})$$

Homotopy groups of a pointed space (A, a):

$$\pi_n(A, a) :\equiv \|\Omega^n(A, a)\|_0$$

Homotopy groups

A typical goal: try to compute the homotopy groups of a space A.

Loop spaces of a pointed space (A, a):

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega^n(a =_A a, \mathsf{refl})$$

Homotopy groups of a pointed space (A, a):

$$\pi_n(A, a) :\equiv \|\Omega^n(A, a)\|_0$$

Simplest example: $\pi_1(A, a)$ is the fundamental group of A — group with multiplication given by path concatenation.

Homotopy groups

A typical goal: try to compute the homotopy groups of a space A.

Loop spaces of a pointed space (A, a):

$$\Omega^0(A, a) :\equiv (A, a)$$
$$\Omega^{n+1}(A, a) :\equiv \Omega^n(a =_A a, \mathsf{refl})$$

Homotopy groups of a pointed space (A, a):

$$\pi_n(A, a) :\equiv \|\Omega^n(A, a)\|_0$$

Simplest example: $\pi_1(A, a)$ is the fundamental group of A — group with multiplication given by path concatenation.

Characterising $\pi_n(A, a)$ is a hard problem, e.g. it is not known even for $\mathbb{S}^k$ for many k.

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl
- loop

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base $=$ base. Can we give a more direct description of the path space base $=$ base?

The path space contains

- refl
- loop
- loop $\cdot$ loop

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl
- loop
- loop ▪ loop
- loop ▪ loop ▪ loop
- ...

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl
- loop
- loop ▪ loop
- loop ▪ loop ▪ loop
- ...
- loop^{-1}

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl
- loop
- loop ▪ loop
- loop ▪ loop ▪ loop
- ...
- loop^{-1}
- $\text{loop}^{-1} \blacksquare \text{loop}^{-1}$

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by base : $\mathbb{S}^1$ and loop : base = base. Can we give a more direct description of the path space base = base?

The path space contains

- refl
- loop
- loop ▪ loop
- loop ▪ loop ▪ loop
- ...
- loop^{-1}
- $\text{loop}^{-1} \blacksquare \text{loop}^{-1}$
- ...

The fundamental group of the circle

Recall $\mathbb{S}^1$ given by $\mathsf{base} : \mathbb{S}^1$ and $\mathsf{loop} : \mathsf{base} = \mathsf{base}$. Can we give a more direct description of the path space $\mathsf{base} = \mathsf{base}$?

The path space contains

- refl
- loop
- $\mathsf{loop} \cdot \mathsf{loop}$
- $\mathsf{loop} \cdot \mathsf{loop} \cdot \mathsf{loop}$
- $\ldots$
- loop^{-1}
- $\mathsf{loop}^{-1} \cdot \mathsf{loop}^{-1}$
- $\ldots$

Idea: Perhaps $(\mathsf{base} = \mathsf{base}) \simeq \mathbb{Z}$?

Encoding and decoding loops [Licata, Shulman 2013]

We define a function $\mathsf{encode} : \mathbb{Z} \to (\mathsf{base} = \mathsf{base})$ by

$$\mathsf{encode}\,(+n) :\equiv \mathsf{loop}^n$$
$$\mathsf{encode}\,0 :\equiv \mathsf{refl}$$
$$\mathsf{encode}\,(-n) :\equiv (\mathsf{loop}^{-1})^n$$

where p^n is $p \cdot p \cdot \ldots \cdot p$ n times.

Encoding and decoding loops [Licata, Shulman 2013]

We define a function $\mathsf{encode} : \mathbb{Z} \to (\mathsf{base} = \mathsf{base})$ by

$$\mathsf{encode}\,(+n) :\equiv \mathsf{loop}^n$$
$$\mathsf{encode}\,0 :\equiv \mathsf{refl}$$
$$\mathsf{encode}\,(-n) :\equiv (\mathsf{loop}^{-1})^n$$

where p^n is $p \cdot p \cdot \ldots \cdot p$ n times.

Encoding and decoding loops [Licata, Shulman 2013]

We define a function $\mathsf{encode} : \mathbb{Z} \to (\mathsf{base} = \mathsf{base})$ by

$$\mathsf{encode}\,(+n) :\equiv \mathsf{loop}^n$$
$$\mathsf{encode}\,0 :\equiv \mathsf{refl}$$
$$\mathsf{encode}\,(-n) :\equiv (\mathsf{loop}^{-1})^n$$

where p^n is $p \cdot p \cdot \ldots \cdot p$ n times.

In the other direction, we first need to generalise from $\mathsf{base} = \mathsf{base}$ to $\mathsf{base} = x$ — otherwise we cannot use path induction.

Encoding and decoding loops [Licata, Shulman 2013]

We define a function $\mathsf{encode} : \mathbb{Z} \to (\mathsf{base} = \mathsf{base})$ by

$$\mathsf{encode}\,(+n) :\equiv \mathsf{loop}^n$$
$$\mathsf{encode}\,0 :\equiv \mathsf{refl}$$
$$\mathsf{encode}\,(-n) :\equiv (\mathsf{loop}^{-1})^n$$

where p^n is $p \cdot p \cdot \ldots \cdot p$ n times.

In the other direction, we first need to generalise from $\mathsf{base} = \mathsf{base}$ to $\mathsf{base} = x$ — otherwise we cannot use path induction.

Hence our goal is to define a function

$$\mathsf{decode} : (\Pi x : \mathbb{S}^1)(\mathsf{base} = x \to P(x))$$

for some $P : \mathbb{S}^1 \to \mathcal{U}$ such that $P(\mathsf{base}) = \mathbb{Z}$.

Encoding and decoding loops [Licata, Shulman 2013]

We define a function $\mathsf{encode} : \mathbb{Z} \to (\mathsf{base} = \mathsf{base})$ by

$$\mathsf{encode}\,(+n) :\equiv \mathsf{loop}^n$$
$$\mathsf{encode}\,0 :\equiv \mathsf{refl}$$
$$\mathsf{encode}\,(-n) :\equiv (\mathsf{loop}^{-1})^n$$

where p^n is $p \cdot p \cdot \ldots \cdot p$ n times.

In the other direction, we first need to generalise from $\mathsf{base} = \mathsf{base}$ to $\mathsf{base} = x$ — otherwise we cannot use path induction.

Hence our goal is to define a function

$$\mathsf{decode} : (\Pi x : \mathbb{S}^1)(\mathsf{base} = x \to P(x))$$

for some $P : \mathbb{S}^1 \to \mathcal{U}$ such that $P(\mathsf{base}) = \mathbb{Z}$.

Why so complicated? Otherwise we would get stuck proving the roundtrips.

Generalising the decoding

We define $P : \mathbb{S}^1 \to \mathcal{U}$ by

$$P\,\mathsf{base} :\equiv \mathbb{Z}$$

$$ap_P\,\mathsf{loop} :\equiv \{\mathbb{Z} = \mathbb{Z}\}$$

Generalising the decoding

We define $P : \mathbb{S}^1 \to \mathcal{U}$ by

$$P\,\mathsf{base} :\equiv \mathbb{Z}$$
$$ap_P\,\mathsf{loop} :\equiv \mathsf{ua}(+1)$$

Generalising the decoding

We define $P : \mathbb{S}^1 \to \mathcal{U}$ by

$$P\,\mathsf{base} :\equiv \mathbb{Z}$$
$$ap_P\,\mathsf{loop} :\equiv \mathsf{ua}(+1)$$

By path induction, we can now define

$$\mathsf{decode} : (\Pi x : \mathbb{S}^1)(\mathsf{base} = x \to P(x))$$

by $\mathsf{decode}\,\mathsf{base}\,\mathsf{refl} :\equiv 0$, and we get

$$\mathsf{decode}\,\mathsf{base} : (\mathsf{base} = \mathsf{base}) \to \mathbb{Z}$$

Generalising the decoding

We define $P : \mathbb{S}^1 \to \mathcal{U}$ by

$$P\,\mathsf{base} :\equiv \mathbb{Z}$$
$$ap_P\,\mathsf{loop} :\equiv \mathsf{ua}(+1)$$

By path induction, we can now define

$$\mathsf{decode} : (\Pi x : \mathbb{S}^1)(\mathsf{base} = x \to P(x))$$

by $\mathsf{decode}\,\mathsf{base}\,\mathsf{refl} :\equiv 0$, and we get

$$\mathsf{decode}\,\mathsf{base} : (\mathsf{base} = \mathsf{base}) \to \mathbb{Z}$$

Exercise: Prove the roundtrip equations to finish the argument that $\pi_1(\mathbb{S}^1) \simeq \mathbb{Z}$ (actually $\Omega^1(\mathbb{S}^1) \simeq \mathbb{Z}$!).

Other results

- $\pi_{k<n}(\mathbb{S}^n) \simeq \mathbf{0}$ [Brunerie]
- $\pi_n(\mathbb{S}^n) \simeq \mathbb{Z}$ [Brunerie-Licata 2013]
- $\pi_4(\mathbb{S}^3) \simeq \mathbb{Z}/n\mathbb{Z}$ where $n = 2$ (Brunerie's constant) [Brunerie 2016]
- Blakers-Massey theorem [Favonia, Finster, Licata, Lumsdaine 2016]
- van Kampen theorem [Favonia, Shulman 2016]
- ...

Exercises

1. Define the integers $\mathbb{Z}$ as a HIT with constructors

$$\begin{aligned} 0 &: \mathbb{Z} \\ \mathsf{succ} &: \mathbb{Z} \to \mathbb{Z} \\ \mathsf{pred} &: \mathbb{Z} \to \mathbb{Z} \\ \mathsf{succpred} &: (\Pi x : \mathbb{Z})(\mathsf{succ}\,(\mathsf{pred}\,x) = x) \\ \mathsf{predsucc} &: (\Pi x : \mathbb{Z})(\mathsf{pred}\,(\mathsf{succ}\,x) = x) \\ \mathsf{set} &: (\Pi x, y : \mathbb{Z})(\Pi p, q : x = y)(p = q) \end{aligned}$$

   Formulate how to define functions $(\Pi x : \mathbb{Z})P(x)$.
2. Define addition $+ : \mathbb{Z} \to \mathbb{Z} \to \mathbb{Z}$, and prove it associative.
3. Prove that $\mathbb{Z}$ has decidable equality by defining a normalisation function $\mathbb{Z} \simeq \mathbb{N} + \mathbf{1} + \mathbb{N}$ (each summand representing positive, zero, negative numbers, respectively).
4. Show that the set truncation constructor can be replaced by a "coherence" constructor

$$\mathsf{coh} : (\Pi x : \mathbb{Z})(\mathsf{succpred}(\mathsf{succ}\,x) = \mathsf{ap}_{\mathsf{succ}}\,(\mathsf{predsucc}\,x))$$

Summary

Higher inductive types allow also paths to be inductively generated.

Can be used to construct down-to-earth types such as quotients, and also truly higher-dimensional types for synthetic homotopy theory.

Summary

Higher inductive types allow also paths to be inductively generated.

Can be used to construct down-to-earth types such as quotients, and also truly higher-dimensional types for synthetic homotopy theory.

Thank you!

References

T. Coquand, N. Danielsson

Isomorphism is equality

Indagationes Mathematicae 24(4), 2013

K. Sojakova

The Equivalence of the Torus and the Product of Two Circles in Homotopy Type Theory

ACM Transactions on Computational Logic 17(4), 2016

D. Licata, M. Shulman

Calculating the Fundamental Group of the Circle in Homotopy Type Theory

LICS 2013.

D. Licata, G. Brunerie

$\pi_n(\mathbb{S}^n)$ in Homotopy Type Theory

CPP 2013

G. Brunerie

On the homotopy groups of spheres in homotopy type theory

PhD thesis, 2016

Favonia, E. Finster, D. Licata, P. Lumsdaine

A Mechanization of the Blakers-Massey Connectivity Theorem in Homotopy Type Theory

LICS 2016

Favonia, M. Shulman

The Seifert–van Kampen Theorem in Homotopy Type Theory

CSL 2016