Homotopy Type Theory
Synthetic reasoning about spaces through dependent type theory
Homotopy Type Theory

Synthetic reasoning about spaces through dependent type theory

\[ A : U \]

Types are spaces
Homotopy Type Theory

Synthetic reasoning about spaces through dependent type theory

- $A : \mathcal{U}$
- $a, b : A$

Terms are points
Homotopy Type Theory

Synthetic reasoning about spaces through dependent type theory

- $A : \mathcal{U}$
- $a, b : A$
- $e, f : a =_A b$

(Propositional) equality proofs are paths
Synthetic reasoning about spaces through dependent type theory

- $A : \mathcal{U}$
- $a, b : A$
- $e, f : a =_A b$
- $h : e =_{a=b} f$

Equalities between equalities are homotopies
Synthetic reasoning about spaces through dependent type theory

- $A : U$
- $a, b : A$
- $e, f : a =_A b$
- $h : e =_{a=b} f$

And so on...
HoTT is based on intensional MLTT plus
HoTT is based on intensional MLTT plus Univalence Axiom

\[(A =_U B) \simeq (A \simeq B)\]

“Equivalences are equalities between types”
Homotopy Type Theory

HoTT is based on intensional MLTT plus

Univalence Axiom

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

“Equivalences are equalities between types”

Higher Inductive Types

Inductive \(S^1 : \mathcal{U} :=\)

\[\begin{align*}
  & base : S^1 \\
  & loop : base =_{S^1} base
\end{align*}\]
Homotopy Type Theory

HoTT is based on intensional MLTT plus

Univalence Axiom

\[(A =_U B) \simeq (A \simeq B)\]

“Equivalences are equalities between types”

Higher Inductive Types

\[
\begin{align*}
\text{Inductive } S^1 : U :={}& \\
| \text{base} & : S^1 \\
| \text{loop} & : \text{base} =_{S^1} \text{base}
\end{align*}
\]

Which do not have computational content as is.
The Cubical Model
The Cubical Model\textsuperscript{1} is an interpretation of HoTT in constructive set theory. Types are interpreted as \textit{cubical presheaves}.

\textsuperscript{1}Coquand et al., 2013
The Cubical Model is an interpretation of HoTT in constructive set theory. Types are interpreted as *cubical presheaves*.

A cubical presheaf is a functor

\[ F : \square \to \text{Set} \]

from the category of cubes to Set.

Basically what you get by gluing sets of points, lines, squares, cubes, ... along their faces.

---

1 Coquand et al., 2013
The Cubical Model\textsuperscript{1} is an interpretation of HoTT in constructive set theory. Types are interpreted as \textit{cubical presheaves}.

A cubical presheaf is a functor

\[ F : \square \rightarrow \text{Set} \]

from the category of cubes to Set.

Basically what you get by gluing sets of points, lines, squares, cubes, ... along their faces.

\textsuperscript{1}Coquand et al., 2013
But not every cubical presheaf can be a type! Types are **fibrant** cubical presheaves.
The Cubical Model

But not every cubical presheaf can be a type! Types are **fibrant** cubical presheaves.

Fibrancy structure (roughly) : maps any open n-dimensional box to a n-dimensional cube that extends it
The Cubical Model

But not every cubical presheaf can be a type! Types are **fibrant** cubical presheaves.

Fibrancy structure (roughly) : maps any open n-dimensional box to a n-dimensional cube that extends it
The Cubical Model

But not every cubical presheaf can be a type! Types are fibrant cubical presheaves.

Fibrancy structure (roughly) : maps any open n-dimensional box to a n-dimensional cube that extends it

The box filling operation implements the J eliminator for the equalities (higher dimensional faces) of the cubical structure.
Fibrant cubical presheaves are stable under all of the constructs of MLTT, so one can build a model with them.
Fibrant cubical presheaves are stable under all of the constructs of MLTT, so one can build a model with them.

This models also happens to realize function extensionality, univalence, and HITs \(\Rightarrow\) full model of HoTT.
Fibrant cubical presheaves are stable under all of the constructs of MLTT, so one can build a model with them.

This models also happens to realize function extensionality, univalence, and HITs $\Rightarrow$ full model of HoTT.

It has been reified into **Cubical Type Theory**, which satisfies canonicity and univalence. However, normalization and decidability of type-checking are still open.
Models as Syntactic Translations
Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$: 
Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$\Gamma \vdash_S t : A$$
Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$
\Gamma \vdash_S t : A \sim \leadsto [\Gamma] \vdash_T [t] : [A]
$$
Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$\Gamma \vdash_S t : A \quad \rightsquigarrow \quad [\Gamma] \vdash_T [t] : [A]$$

Translations do this inductively by translating every typing rule:
**Translations** are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$
\Gamma \vdash_S t : A \implies [\Gamma] \vdash_T [t] : [A]
$$

Translations do this inductively by translating every typing rule:

$$
\Gamma, x : A \vdash_S t : B \\
\Gamma \vdash_S \lambda x . t : \Pi (x : A) . B
$$
Models as Syntactic Translations

Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

\[
\Gamma \vdash_S t : A \quad \leadsto \quad [\Gamma] \vdash_T [t] : [A]
\]

Translations do this inductively by translating every typing rule:

\[
\frac{
\begin{align*}
\Gamma, x : A & \vdash_S t : B \\
\Gamma & \vdash_S \lambda x \cdot t : \Pi (x : A) \cdot B
\end{align*}
}{
\frac{
\begin{align*}
[\Gamma], x : [A] & \vdash_T [t] : [B] \\
[\Gamma] & \vdash_T [\lambda x \cdot t] : \Pi (x : [A]) \cdot [B]
\end{align*}
}\]
**Translations** are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$
\Gamma \vdash_S t : A \quad \leadsto \quad [\Gamma] \vdash_T [t] : [A]
$$

Translations do this inductively by translating every typing rule:

$$
\begin{align*}
\Gamma, x : A &\vdash_S t : B \\
\Gamma &\vdash_S \lambda x \cdot t : \Pi (x : A) \cdot B \\
\end{align*}
\quad \leadsto \quad
\begin{align*}
[\Gamma], x : [A] &\vdash_T [t] : [B] \\
[\Gamma] &\vdash_T [\lambda x \cdot t] : \Pi (x : [A]) \cdot [B] \\
\end{align*}
$$

And they have to preserve definitional equality between terms:
Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$
\Gamma \vdash_S t : A \quad \sim \rightarrow \quad [\Gamma] \vdash_T [t] : [A]
$$

Translations do this inductively by translating every typing rule:

$$
\begin{align*}
\Gamma, x : A &\vdash_S t : B \\
\Gamma &\vdash_S \lambda x . t : \Pi (x : A) . B
\end{align*}
\quad \sim \rightarrow \quad
\begin{align*}
[\Gamma], x : [A] &\vdash_T [t] : [B] \\
[\Gamma] &\vdash_T [\lambda x . t] : \Pi (x : [A]) . [B]
\end{align*}
$$

And they have to preserve definitional equality between terms:

$$
t \equiv_S u
$$
Models as Syntactic Translations

Translations are a class of models for dependent type theory. Judgements in the source theory $S$ are mapped to the target theory $T$:

$$\Gamma \vdash_S t : A \leadsto [\Gamma] \vdash_T [t] : [A]$$

Translations do this inductively by translating every typing rule:

$$\begin{align*}
\Gamma, x : A \vdash_S t : B & \leadsto \quad [\Gamma], x : [A] \vdash_T [t] : [B] \\
\Gamma \vdash_S \lambda x . t : \Pi (x : A) . B & \leadsto \quad [\Gamma] \vdash_T [\lambda x . t] : \Pi (x : [A]) . [B]
\end{align*}$$

And they have to preserve definitional equality between terms:

$$t \equiv_S u \leadsto [t] \equiv_T [u]$$
A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \cong u \iff [t] \equiv_T [u]$$
A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \equiv u \iff [t] \equiv_T [u]$$

The new theory $(S, \equiv)$ can inherit **good properties** from $T$: 
Models as Syntactic Translations

A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \equiv u \iff [t] \equiv_T [u]$$

The new theory $(S, \equiv)$ can inherit good properties from $T$:

$T \not\vdash [\bot] \Rightarrow S$ and $(S, \equiv)$ are consistent
A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \equiv u \iff \llbracket t \rrbracket \equiv_T \llbracket u \rrbracket$$

The new theory $(S, \equiv)$ can inherit **good properties** from $T$:

- $T \not \vdash \bot$ \quad $\Rightarrow$ \quad $S$ and $(S, \equiv)$ are consistent
- $T$ satisfies canonicity for $\mathbb{N}$ \quad $\Rightarrow$ \quad $(S, \equiv)$ satisfies canonicity
Models as Syntactic Translations

A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \equiv u \iff [t] \equiv_T [u]$$

The new theory $(S, \equiv)$ can inherit good properties from $T$:

- $T \not\models [\bot]$ $\Rightarrow$ $S$ and $(S, \equiv)$ are consistent
- $T$ satisfies canonicity for $[\mathbb{N}]$ $\Rightarrow$ $(S, \equiv)$ satisfies canonicity
- $T$ has decidable type-checking $\Rightarrow$ $(S, \equiv)$ has decidable type-checking
A translation induces a new definitional equality on $S$ that extends $\equiv_S$

$$t \equiv u \iff [t] \equiv_T [u]$$

The new theory $(S, \equiv)$ can inherit good properties from $T$:

- $T \not\models [\bot] \implies S$ and $(S, \equiv)$ are consistent
- $T$ satisfies canonicity for $[\mathbb{N}] \implies (S, \equiv)$ satisfies canonicity
- $T$ has decidable type-checking $\implies (S, \equiv)$ has decidable type-checking

“(S, \equiv) computes if T computes”
A translation induces a new definitional equality on $S$ that extends $\equiv_S$

\[ t \equiv u \iff [t] \equiv_T [u] \]

The new theory $(S, \equiv)$ can inherit good properties from $T$:

$T \not\models \bot$ \implies $S$ and $(S, \equiv)$ are consistent

$T$ satisfies canonicity for $[\mathbb{N}]$ \implies $(S, \equiv)$ satisfies canonicity

$T$ has decidable type-checking \implies $(S, \equiv)$ has decidable type-checking

“$(S, \equiv)$ computes if $T$ computes”

Note that $(S, \equiv)$ does not have a directed reduction, though.
This is enough motivation for writing the cubical model as a translation from HoTT to MLTT.
Models as Syntactic Translations

This is enough motivation for writing the cubical model as a translation from HoTT to MLTT.

The naive approach to write such a translation fails, however.
Models as Syntactic Translations

This is enough motivation for writing the cubical model as a translation from HoTT to MLTT.

The naive approach to write such a translation *fails*, however. Translation must preserve definitional equality, in particular reduction:

$$[A\{x \leftarrow t\}] \equiv [A]\{x \leftarrow [t]\}$$

But crucial properties such as functoriality of presheaves are stated with propositional equality, which does not imply conversion.
Models as Syntactic Translations

This is enough motivation for writing the cubical model as a translation from HoTT to MLTT.

The naive approach to write such a translation fails, however. Translation must preserve definitional equality, in particular reduction:

\[ A\{x \leftarrow t \} \equiv [A]\{x \leftarrow [t] \} \]

But crucial properties such as functoriality of presheaves are stated with propositional equality, which does not imply conversion. (unless we are dealing with an extensional target, but then computation breaks)
Definitional Presheaves
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf.
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is \textit{not the right definition} of a presheaf. Assuming a definitional category $C$, one can define
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf. Assuming a definitional category $C$, one can define

\[ \text{Presheaf} : \mathcal{U} \]
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf.
Assuming a definitional category $C$, one can define

$$\text{Presheaf} : \mathcal{U} \quad \text{Elem} : \text{Presheaf} \to C_0 \to \mathcal{U}$$
This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf. Assuming a definitional category $C$, one can define

\[
\text{Presheaf} : \mathcal{U} \quad \text{Elem} : \text{Presheaf} \rightarrow C_0 \rightarrow \mathcal{U}
\]

\[
e : \text{Elem} F c \\
\alpha : C_1(d, c) \quad \vdash \quad e|_\alpha : \text{Elem} F d
\]
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is **not the right definition** of a presheaf.

Assuming a definitional category $\mathcal{C}$, one can define

\[
\begin{align*}
\text{Presheaf} &: \mathcal{U} & \text{Elem} &: \text{Presheaf} \to \mathcal{C}_0 \to \mathcal{U} \\
& e &: \text{Elem} \ F \ c & \vdash e|_\alpha &: \text{Elem} \ F \ d \\
& \alpha &: \mathcal{C}_1(d, c) & \equiv e|_{\alpha \circ \beta} 
\end{align*}
\]
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf. Assuming a definitional category $C$, one can define

$$
\text{Presheaf} : \mathcal{U} \quad \text{Elem} : \text{Presheaf} \to C_0 \to \mathcal{U}
$$

$$
e : \text{Elem } F c
\alpha : C_1(d, c) \vdash e|_\alpha : \text{Elem } F d
\quad e|_\alpha|_\beta \equiv e|_{\alpha \circ \beta}
$$

$$
\text{NatTransfo} : \text{Presheaf} \to \text{Presheaf} \to \mathcal{U}
$$
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is *not the right definition* of a presheaf.

Assuming a definitional category \( C \), one can define

\[
\begin{align*}
\text{Presheaf} : & \mathcal{U} & \text{Elem} : & \text{Presheaf} \to C_0 \to \mathcal{U} \\
& e : \text{Elem } F c & \alpha : & C_1(d, c) \\
& \vdash e|\alpha : \text{Elem } F d \\
& e|\alpha|\beta \equiv e|\alpha \circ \beta \\
\text{NatTransfo} : & \text{Presheaf} \to \text{Presheaf} \to \mathcal{U} \\
\text{Eval} : & \text{NatTransfo } F G \to \text{Elem } F c \to \text{Elem } G c
\end{align*}
\]
Definitional Presheaves

This failure happens when we try to write the usual definition of a presheaf in intensional type theory.

Claim: in ITT, this is not the right definition of a presheaf.

Assuming a definitional category $C$, one can define

\begin{align*}
\text{Presheaf} & : \mathcal{U} \\
\text{Elem} & : \text{Presheaf} \to C_0 \to \mathcal{U} \\
\end{align*}

\begin{align*}
e & : \text{Elem } F c \\
\alpha & : C_1(d, c) \quad \vdash \quad e|_\alpha : \text{Elem } F d \\
\end{align*}

\[ e|_\alpha|\beta \equiv e|_{\alpha \circ \beta} \]

\begin{align*}
\text{NatTransfo} & : \text{Presheaf} \to \text{Presheaf} \to \mathcal{U} \\
\text{Eval} & : \text{NatTransfo } F G \to \text{Elem } F c \to \text{Elem } G c \\
\text{(Eval } \theta \; e)|_\alpha & \equiv \text{Eval } \theta \; e|_\alpha \\
\end{align*}
Definitional Presheaves

Using this construction, Pédrot managed to write presheaf models of MLTT as translations to \texttt{MLTT+SProp+UIP} (all types are definitional hsets).
Using this construction, Pédrot managed to write presheaf models of MLTT as translations to $\text{MLTT} + \text{SProp} + \text{UIP}$ (all types are definitional hsets).

This theory is not as well understood as MLTT, but it does have a definite computational behaviour.
Using this construction, Pédrot managed to write presheaf models of MLTT as translations to **MLTT+SProp+UIP** (all types are definitional hsets).

This theory is not as well understood as MLTT, but it does have a definite computational behaviour.

Our goal is to do the same with **fibrant** cubical presheaves, to get a translation from HoTT to MLTT+SProp+UIP.
What has been done, what remains to be done

- Dependent Products ✓
- Dependent Sums ✓
- Booleans ✓
- Cubical equality ✓
- Function extensionality ✓
- J eliminator ✓
- Weak univalence ✓
- Full univalence ?
- Fibrancy of the universe ?
- Higher Inductive Types ?
Meta-theoretical Consequences
Meta-theoretical Consequences

What we (would) get:

- a formal, \textit{computational} proof of consistency for HoTT
Meta-theoretical Consequences

What we (would) get:

- a formal, *computational* proof of consistency for HoTT
- a new way to compute with univalence
Meta-theoretical Consequences

What we (would) get:

- a formal, \textit{computational} proof of consistency for HoTT
- a new way to compute with univalence
- \textit{MLTT} + \textit{UIP} + \textit{SProp} has good properties \(\Rightarrow\) (HoTT, \(\simeq\)) has good properties.
Meta-theoretical Consequences

What we (would) get:

- a formal, *computational* proof of consistency for HoTT
- a new way to compute with univalence
- MLTT + UIP + SProp has good properties $\Rightarrow$ (HoTT, $\simeq$) has good properties.

This is nice, but we already knew that “Cubical type theory has good properties $\Rightarrow$ (HoTT, $\simeq'$) has good properties.”
Meta-theoretical Consequences

What we (would) get:

- a formal, *computational* proof of consistency for HoTT
- a new way to compute with univalence
- MLTT + UIP + SProp has good properties $\Rightarrow$ (HoTT, $\simeq$) has good properties.

This is nice, but we already knew that “Cubical type theory has good properties $\Rightarrow$ (HoTT, $\simeq'$) has good properties.”

This method really shines when it comes to extensions of HoTT:

- Good properties for 2-level type theory with little extra work
- You want computational simplicial types? Just define them in the model using strict equality, prove their fibrancy, and voilà!
- etc
Thank you!