Cubical Type Theory: a constructive interpretation of the univalence axiom

Anders Mörberg
(jww C. Cohen, T. Coquand, and S. Huber)

Institute for Advanced Study, Princeton

June 26, 2016
Goal: provide a computational justification for notions from Homotopy Type Theory and Univalent Foundations, in particular the univalence axiom and higher inductive types

Specifically, design a type theory with good properties (normalization, decidability of type checking, etc.) where the univalence axiom computes and which has support for higher inductive types

\(^1\)Slogan: “Making equality great again!”
An extension of dependent type theory which allows the user to directly argue about n-dimensional cubes (points, lines, squares, cubes etc.) representing equality proofs.

Based on a model in cubical sets formulated in a constructive metatheory.

Each type has a “cubical” structure – *presheaf extension* of type theory.
Cubical Type Theory

Extends dependent type theory with:

1. Path types
2. Kan composition operations
3. Glue types (univalence)
4. Identity types
5. Higher inductive types
Basic dependent type theory

\[
\begin{align*}
\Gamma & ::= \; () \mid \Gamma, x : A \\
t, u, A, B & ::= \; x \mid \lambda x : A. \; t \mid tu \mid (x : A) \to B \\
 & \mid (t, u) \mid t.1 \mid t.2 \mid (x : A) \times B \\
 & \mid 0 \mid s \; u \mid \text{natrec} \; t \; u \mid \text{nat}
\end{align*}
\]

with \(\eta\) for functions and pairs
Path types

Path types provides a convenient syntax for reasoning about higher equality proofs.

Contexts can contain variables in the interval:

\[ \Gamma \vdash \Gamma, i : \mathbb{I} \vdash \]

Formal representation of the interval, \( \mathbb{I} \):

\[ r, s ::= 0 \mid 1 \mid i \mid 1 - r \mid r \land s \mid r \lor s \]

\( i, j, k \ldots \) formal symbols/names representing directions/dimensions
Path types

\[ i : \mathbb{I} \vdash A \] corresponds to a line:

\[ A(i_0) \xrightarrow{A} A(i_1) \]

\[ i : \mathbb{I}, j : \mathbb{I} \vdash A \] corresponds to a square:

\[ A(i_0)(j_1) \xrightarrow{A(j_1)} A(i_1)(j_1) \]

\[ \begin{array}{c}
A(i_0) \\
A(i_1)
\end{array} \]

\[ A(i_0)(j_0) \xrightarrow{A(j_0)} A(i_1)(j_0) \]

and so on…
Path types

\[
\Gamma \vdash A \quad \Gamma, i : \Pi \vdash t : A
\]

\[
\Gamma \vdash \langle i \rangle t : \text{Path } A \ t(i0) \ t(i1)
\]

\[
\Gamma \vdash t : \text{Path } A \ u_0 u_1
\]

\[
\Gamma \vdash t 0 = u_0 : A
\]

\[
\Gamma \vdash t 1 = u_1 : A
\]

\[
\Gamma \vdash A \quad \Gamma, i : \Pi \vdash t : A
\]

\[
\Gamma \vdash (\langle i \rangle t) r = t(i/r) : A
\]
Path types

Path abstraction, $\langle i \rangle t$, binds the name $i$ in $t$

$$t(i0) \xrightarrow{t} t(i1) \quad \quad \quad t(i0) \xrightarrow{\langle i \rangle t} t(i1)$$

Path application, $t r$, applies a term $t$ to an element $r : \mathbb{I}$

$$a \xrightarrow{t} b \quad \quad \quad b \xrightarrow{t(1-i)} a$$
Given $p : \text{Path } A \ a \ b$ we can build

$$
\begin{align*}
\text{Path types: connections} & \\
\text{Given } p : \text{Path } A \ a \ b \text{ we can build} & \\
\begin{array}{c}
a \xrightarrow{p_i} b \\
p_0 \\
a \xrightarrow{p_0} a \\
p (i \land j) \\
p_i & \text{Path types}\n\end{array} & \\
\begin{array}{c}
b \xrightarrow{p_1} b \\
p_j \\
a \xrightarrow{p_i} b \\
p (i \lor j) \\
p_1 & \text{Path types}\n\end{array} & \\
\begin{array}{c}
j \xrightarrow{i} \\
p_j \\
a \xrightarrow{p_i} b \\
p (i \lor j) \\
p_1 & \text{Path types}\n\end{array}
\end{align*}
$$
Path types are great!

Function extensionality for path types can be proved as:

\[
\Gamma \vdash f, g : (x : A) \to B \quad \Gamma \vdash p : (x : A) \to \text{Path } B \ (f \ x) \ (g \ x)
\]

\[
\Gamma \vdash \langle i \rangle \ \lambda x : A. \ p \ x \ i : \text{Path } ((x : A) \to B) \ f \ g
\]
Path types are great!

Given \( f : A \rightarrow B \) and \( p : \text{Path} \ A \ a \ b \) we can define:

\[
ap f p = \langle i \rangle f (p i) : \text{Path} \ B \ (f a) \ (f b)
\]

satisfying definitionally:

\[
ap \text{id} \ p = p
\]
\[
ap (f \circ g) \ p = ap f (ap g p)
\]

This way we get new ways for reasoning about equality: inline \( \text{ap} \), \( \text{funext} \), symmetry... with new definitional equalities
Path types are great!

We can also prove contractibility of singletons\(^2\):

\[
\Gamma \vdash p : \text{Path } A \ a \ b
\]
\[
\Gamma \vdash \langle i \rangle (p \ i, \langle j \rangle \ p \ (i \land j)) : \text{Path } ((x : A) \times (\text{Path } A \ a \ x)) \ (a, 1_a) \ (b, p)
\]

But we cannot yet compose paths...

\(^2\) or “Vacuum Cleaner Power Cord Principle”
Kan composition operations

We want to be able to compose paths:

\[
\begin{array}{cc}
a & \xrightarrow{p} & b \\
\end{array} \quad \quad \quad \quad \quad
\begin{array}{cc}
b & \xrightarrow{q} & c \\
\end{array}
\]

We do this by computing the dashed line in:

\[
\begin{array}{cc}
a & \xrightarrow{p} & b \\
\end{array}
\quad \quad \quad \quad \quad
\begin{array}{cc}
\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quarter
Kan composition operations

**Box principle:** any open box has a lid

Cubical version of the Kan condition for simplicial sets:

"Any horn can be filled"

First formulated by Daniel Kan in "Abstract Homotopy I" (1955) for cubical complexes
Partial elements

To formulate this we need syntax for representing partially specified n-dimensional cubes

We add context restrictions $\Gamma, \varphi$ where $\varphi$ is a “face” formula

If $\Gamma \vdash A$ and $\Gamma, \varphi \vdash a : A$ then $a$ is a partial element of $A$ of extent $\varphi$

If $\Gamma, \varphi \vdash A$ then $A$ is a partial type of extent $\varphi$
## Examples of partial types

<table>
<thead>
<tr>
<th>$i : \mathbb{I}, (i = 0) \lor (i = 1) \vdash A$</th>
<th>$A(i0) \bullet \quad \bullet A(i1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$i \cdot j : \mathbb{I}, (i = 0) \lor (i = 1) \lor (j = 0) \vdash A$</td>
<td>$A(i0)(j1) \quad A(i1)(j1)$</td>
</tr>
<tr>
<td>$A(i0)$</td>
<td>$A(i1)$</td>
</tr>
<tr>
<td>$A(i0)(j0) \quad \xrightarrow{A(j0)} \quad A(i1)(j0)$</td>
<td></td>
</tr>
</tbody>
</table>

The face lattice $\mathbb{F}$ is a bounded distributive lattice on formal generators $(i = 0)$ and $(i = 1)$ with relation $(i = 0) \land (i = 1) = 0_{\mathbb{F}}$
Partial elements

Any judgment valid in a context $\Gamma$ is also valid in a restriction $\Gamma, \varphi$

$$\Gamma \vdash A$$

$$\Gamma, \varphi \vdash A$$

Contexts $\Gamma$ are modeled by cubical sets

Restriction operation correspond to a **cofibration**:

$$\Gamma, \varphi \rightarrow \Gamma$$
An element $\Gamma, \varphi \vdash a : A$ is **connected** if we have $\Gamma \vdash b : A$ such that $\Gamma, \varphi \vdash a = b : A$.

We write $\Gamma \vdash b : A[\varphi \mapsto a]$ and say that $b$ **witnesses** that $a$ is connected.

This generalizes the notion of being path connected. Let $\varphi$ be $(i = 0) \lor (i = 1)$, an element $b : A[\varphi \mapsto a]$ is a line:

$$
\begin{array}{c}
\text{a(i0)} \quad \xrightarrow{b} \quad \text{a(i1)}
\end{array}
$$
Box principle

We can now formulate the box principle in type theory:

\[
\begin{align*}
\Gamma, i : \mathbb{I} &\vdash A \\
\Gamma, \varphi, i : \mathbb{I} &\vdash u : A \\
\Gamma &\vdash a_0 : A(i0)[\varphi \mapsto u(i0)] \\
\Gamma &\vdash \text{comp}^i A [\varphi \mapsto u] a_0 : A(i1)[\varphi \mapsto u(i1)]
\end{align*}
\]

\(u\) is a partial path connected at \(i = 0\) specifying the sides of the box
\(a_0\) is the bottom of the box
\(\text{comp}^i\) witnesses that \(u\) is connected at \(i = 1\)

The equality judgments for the composition operation are defined by induction on \(A\) – this is the main part of the system
Kan composition: example

With composition we can justify transitivity of path types:

\[
\Gamma \vdash p : \text{Path } A \ a \ b \quad \Gamma \vdash q : \text{Path } A \ b \ c
\]

\[
\Gamma \vdash \langle i \rangle \ \text{comp}^j A \ [(i = 0) \mapsto a, (i = 1) \mapsto q \ j] \ (p \ i) : \text{Path } A \ a \ c
\]
Kan composition: transport

Composition for $\varphi = 0_\mathbb{F}$ corresponds to transport:

$$
\frac{
\Gamma, i : I \vdash A \quad \Gamma \vdash a : A(i0)
}{
\Gamma \vdash \text{transport}^i A a = \text{comp}^i A [\cdot] a : A(i1)
}
$$

Together with contractibility of singletons we can prove path induction, that is, given $x : A$ and $p : \text{Path} A a x$ we get

$$
C(a, 1_a) \to C'(x, p)
$$
Glue types

We extend the system with Glue types, these allow us to:

- Define composition for the universe
- Prove univalence

Composition for these types is the most complicated part of the system
Univalence?

What is needed in order to prove univalence?
Univalence?

What is needed in order to prove univalence?

For all types $A$ and $B$ we need to define a term:

$$ua : \text{Equiv} \ (\text{Path} \ U \ A \ B) \ (\text{Equiv} \ A \ B)$$

showing that the canonical map

$$\text{pathToEquiv} : \text{Path} \ U \ A \ B \to \text{Equiv} \ A \ B$$

is an equivalence
Univalence?

The following is an alternative characterization of univalence:

**Univalence axiom**

\[
\text{For any type } A : U \text{ the type } (T : U) \times \text{Equiv } T \ A \text{ is contractible}
\]

This is a version of contractibility of singletons for equivalences. So if we can also transport along equivalences we get an induction principle for equivalences.
Univalence?

Lemma

The type $\text{isContr } A$ is inhabited iff we have an operation:

$$
\Gamma, \varphi \vdash u : A
$$

$$
\Gamma \vdash \text{ext } [\varphi \mapsto u] : A[\varphi \mapsto u]
$$
Lemma

The type isContr $A$ is inhabited iff we have an operation:

$$
\Gamma, \varphi \vdash u : A
\quad \Rightarrow
\Gamma \vdash \text{ext} [\varphi \mapsto u] : A[\varphi \mapsto u]
$$

So to prove univalence it suffices to show that any partial element

$$
\Gamma, \varphi \vdash (T, e) : (T : U) \times \text{Equiv } T A
$$

extends to a total element
Example: unary and binary numbers

Let \( \text{nat} \) be unary natural numbers (0 and successor) and \( \text{binnat} \) be binary natural numbers (lists of 0 and 1). We have an equivalence

\[
e : \text{binnat} \to \text{nat}
\]

and we want to construct a path \( P \) with \( P(i0) = \text{nat} \) and \( P(i1) = \text{binnat} \):

\[
\text{nat} \overset{P}{\longrightarrow} \text{binnat}
\]
Example: unary and binary numbers

$P$ should also store information about $e$, we achieve this by “glueing”:

$\begin{array}{c}
\text{nat} \xrightarrow{P} \text{binnat} \\
\downarrow \quad \downarrow \\
\text{id} \quad \$ \\
\text{nat} \xrightarrow{\text{nat}} \text{nat}
\end{array}$

We write

$$i : \Pi \vdash P = \text{Glue } [(i = 0) \mapsto (\text{nat}, \text{id}), (i = 1) \mapsto (\text{binnat}, e)] \text{ nat}$$
Glue: more generally

In the case when $\varphi$ is $(i = 0) \lor (i = 1)$ the glueing operation can be illustrated as the dashed line in:

\[
\begin{array}{ccc}
T_0 & \longrightarrow & T_1 \\
| & & |
\end{array}
\begin{array}{ccc}
e(i0) & \downarrow & e(i1) \\
| & & |
\end{array}
\begin{array}{ccc}
A(i0) & \longrightarrow & A(i1) \\
\downarrow & & \downarrow \\
A & & A
\end{array}
\]
Glue: even more generally

We assume that we are given

- $\Gamma \vdash A$
- A partial type $\Gamma, \varphi \vdash T$
- An equivalence $\Gamma, \varphi \vdash e : T \to A$

From this we define

A total type $\Gamma \vdash \text{Glue}[\varphi \mapsto (T, e)] A$

A map $\Gamma \vdash \text{unglue} : \text{Glue}[\varphi \mapsto (T, e)] A \to A$

such that $\text{Glue}[\varphi \mapsto (T, e)] A$ and $\text{unglue}$ are extensions of $T$ and $e$:

$\Gamma, \varphi \vdash T = \text{Glue}[\varphi \mapsto (T, e)] A$

$\Gamma, \varphi \vdash e = \text{unglue} : T \to A$
Glue: even more generally

We assume that we are given

- \( \Gamma \vdash A \)
- A partial type \( \Gamma, \varphi \vdash T \)
- An equivalence \( \Gamma, \varphi \vdash e : T \to A \)

From this we define

- A total type \( \Gamma \vdash \text{Glue} [\varphi \mapsto (T, e)] A \)
- A map \( \Gamma \vdash \text{unglue} : \text{Glue} [\varphi \mapsto (T, e)] A \to A \)
Glue: even more generally

We assume that we are given

- \( \Gamma \vdash A \)
- A partial type \( \Gamma, \varphi \vdash T \)
- An equivalence \( \Gamma, \varphi \vdash e : T \to A \)

From this we define

- A total type \( \Gamma \vdash \text{Glue} \ [\varphi \mapsto (T, e)] \ A \)
- A map \( \Gamma \vdash \text{unglue} : \text{Glue} \ [\varphi \mapsto (T, e)] \ A \to A \)

such that \( \text{Glue} \ [\varphi \mapsto (T, e)] \ A \) and \( \text{unglue} \) are extensions of \( T \) and \( e \):

\[
\Gamma, \varphi \vdash T = \text{Glue} \ [\varphi \mapsto (T, e)] \ A \quad \Gamma, \varphi \vdash e = \text{unglue} : T \to A
\]
Glue: diagrammatically

\[ T \xrightarrow{\varsigma} A \xrightarrow{e} A \]

\[ \Gamma, \varphi \xrightarrow{\varsigma} A \xrightarrow{\Gamma} \]
Glue: diagrammatically

\[
T \xrightarrow{e} \text{Glue} \quad \Downarrow\quad \sim \quad \Downarrow\quad \text{unglue}
\]

\[
\begin{array}{c}
\Gamma, \varphi \\
\Downarrow
\end{array} \quad \xrightarrow{\sim} \quad \begin{array}{c}
\Gamma
\end{array}
\]

\[
A \quad \xrightarrow{\sim} \quad A
\]

Anders Mörberg

Glue and univalence
Rules for Glue

\[
\begin{align*}
& \text{if } \Gamma \vdash A, \quad \Gamma, \varphi \vdash T, \quad \Gamma, \varphi \vdash e : \text{Equiv } T \ A, \\
& \quad \text{then } \Gamma \vdash \text{Glue } [\varphi \mapsto (T, e)] \ A \\
\end{align*}
\]

\[
\begin{align*}
& \text{if } \Gamma, \varphi \vdash e : \text{Equiv } T \ A, \quad \Gamma, \varphi \vdash t : T, \quad \Gamma \vdash a : A[\varphi \mapsto e \ t], \\
& \quad \text{then } \Gamma \vdash \text{glue } [\varphi \mapsto t] \ a : \text{Glue } [\varphi \mapsto (T, e)] \ A \\
\end{align*}
\]

\[
\begin{align*}
& \text{if } \Gamma \vdash b : \text{Glue } [\varphi \mapsto (T, e)] \ A, \\
& \quad \text{then } \Gamma \vdash \text{unglue } b : A
\end{align*}
\]

together with equality judgments
Composition for Glue

Let $\Gamma, i : \Pi \vdash B = \text{Glue} [\varphi \mapsto (T, e)] A$. Given

\[
\Gamma, \psi, i : \Pi \vdash b : B \quad \Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]
\]
Composition for Glue

Let $\Gamma, i : \Pi \vdash B = \text{Glue} \ [\varphi \mapsto (T, e)] \ A$. Given

$$\Gamma, \psi, i : \Pi \vdash b : B \quad \Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$$

The algorithm computes

$$b_1 = \text{comp}^i B \ [\psi \mapsto b] \ b_0$$

such that:

$$\Gamma \vdash b_1 : B(i1)[\psi \mapsto b(i1)] \quad \Gamma, \delta \vdash b_1 : T(i1)$$

where $\delta$ is the part of $\varphi$ that doesn’t mention $i$
Composition for Glue

Let $\Gamma, i : \Pi \vdash B = \text{Glue} \; [\varphi \mapsto (T, e)] \; A$. Given

$\Gamma, \psi, i : \Pi \vdash b : B$ \hspace{1cm} $\Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$

The algorithm computes

$$b_1 = \text{comp}^i B \; [\psi \mapsto b] \; b_0$$

such that:

$\Gamma \vdash b_1 : B(i1)[\psi \mapsto b(i1)]$ \hspace{1cm} $\Gamma, \delta \vdash b_1 : T(i1)$

where $\delta$ is the part of $\varphi$ that doesn’t mention $i$

Composition for Glue is the most complicated part of the system
Composition for Glue in Nuprl

\text{comp}(%\text{Glue} \ [\phi \mapsto T,f] \ A) = \\
\text{\backslash H,}\sigma,\psi, b, b0. \\
\text{let } a = \text{unglue}(b) \text{ in} \\
\text{let } a0 = \text{unglue}(b0) \text{ in} \\
\text{let } a'1 = \text{comp} (\text{cA})\sigma [\psi \mapsto a] a0 \text{ in} \\
\text{let } t'1 = \text{comp} (\text{cT})\sigma [\psi \mapsto b] b0 \text{ in} \\
\text{let } g = (f.1)\sigma \text{ in} \\
\text{let } w = \text{pres } g [\psi \mapsto b] b0 \text{ in} \\
\text{let } \phi' = \text{forall} (\phi)\sigma \text{ in} \\
\text{let } \phi1 = (\phi)\sigma[1] \text{ in} \\
\text{let } st = \text{if } \phi' \text{ then } t'1 \text{ else } b[1] \text{ in} \\
\text{let } sw = \text{if } \phi' \text{ then } w \text{ else } < > ((g b)[1]) p \text{ in} \\
\text{let } cF = \text{fiber-comp} (H, \phi1) (\text{cT})\sigma[1] (\text{cA})\sigma[1] g[1] a'1 \text{ in} \\
\text{let } z = \text{equiv } cF g[1] [\phi' \lor \psi \mapsto (st,sw)] a'1 \text{ in} \\
\text{let } t1 = z.1 \text{ in} \\
\text{let } \alpha = z.2 \text{ in} \\
\text{let } x = \text{if } (\phi1)p \text{ then } (\alpha)p \odot q \text{ else } a[1]p \text{ in} \\
\text{let } a1 = \text{comp} (\text{cA})\sigma[1]p [(\phi1 \lor \psi \mapsto x)] a'1 \text{ in} \\
\text{glue } [\phi1 \mapsto t1] a1
Composition for the universe from Glue

Given $\Gamma \vdash A$, $\Gamma \vdash B$, and $\Gamma, i : \Pi \vdash E$, such that

$$E(i0) = A \quad E(i1) = B$$

Using transport we can construct$^3$

$$\text{equiv}^i E : \text{Equiv} A B$$

$^3$Note that $\text{equiv}^i E$ binds $i$ in $E$
Composition for the universe from Glue

Given $\Gamma \vdash A$, $\Gamma \vdash B$, and $\Gamma, i : \mathbb{I} \vdash E$, such that

$$E(i0) = A \quad \quad E(i1) = B$$

Using transport we can construct$^3$

$$\text{equiv}^i E : \text{Equiv} \ A \ B$$

Using this we can define the composition for the universe:

$$\Gamma \vdash \text{comp}^i U [\varphi \mapsto E] A =$$

$$\text{Glue} [\varphi \mapsto (E(i1), \text{equiv}^i E(i/1 - i))] A : U[\varphi \mapsto E(i1)]$$

$^3$Note that $\text{equiv}^i E$ binds $i$ in $E$
Proof of univalence

Recall that in order to prove univalence it suffices to show that any partial element

$$\Gamma, \varphi \vdash (T, e) : (T : U) \times \text{Equiv } T A$$

extends to a total element

$$\Gamma \vdash (T', e') : ((T' : U) \times \text{Equiv } T' A)[\varphi \mapsto (T, e)]$$
Proof of univalence

Recall that in order to prove univalence it suffices to show that any partial element
\[ \Gamma, \varphi \vdash (T, e) : (T : U) \times \text{Equiv} \ T \ A \]
extends to a total element
\[ \Gamma \vdash (T', e') : ((T' : U) \times \text{Equiv} \ T' \ A)[\varphi \mapsto (T, e)] \]
This is exactly what Glue gives us!

\[ T' = \text{Glue} [\varphi \mapsto (T, e)] \ A \quad e' = (\text{unglue}, ?) \]
For ? we need to prove that unglue is an equivalence
Proof of univalence

\[
\begin{array}{ccc}
T & \xrightarrow{e} & \text{Glue} \\
\downarrow & & \downarrow \text{unglue} \\
\Gamma, \phi & \xrightarrow{\sim} & \Gamma \\
\end{array}
\]
Proof of univalence

\[
\begin{array}{c}
T \quad \xrightarrow{e} \quad \Gamma, \varphi \\
| \quad \downarrow \quad \downarrow \\
\Gamma \quad \xrightarrow{\varphi} \quad A
\end{array}
\]

\[
\begin{array}{c}
\text{Glue} \quad \xrightarrow{\text{unglue}} \quad A
\end{array}
\]
Proof of univalence

So we get:

Corollary

For any type \( A : U \) the type \((T : U) \times \text{Equiv } T A\) is contractible

From this we obtain this general statement of the univalence axiom:

Corollary

For any term

\[
t : (A B : U) \to \text{Path } U A B \to \text{Equiv } A B
\]

the map \( t A B : \text{Path } U A B \to \text{Equiv } A B \) is an equivalence
Identity types

Path types satisfy many new definitional equalities, but the computation rule for path elimination does **not** hold definitionally.
Identity types

Path types satisfy many new definitional equalities, but the computation rule for path elimination does \textbf{not} hold definitionally.

This is because the computation rule

$$\text{transport}^i \ A \ a = a$$

if $A$ is independent of $i$ doesn’t hold definitionally.
Identity types

Path types satisfy many new definitional equalities, but the computation rule for path elimination does **not** hold definitionally.

This is because the computation rule

$$\text{transport}^i \ A \ a = a$$

if $A$ is independent of $i$ doesn’t hold definitionally.

However we can define (based on ideas of Andrew Swan) a new type, equivalent to Path, which satisfies this.
We define a type $\text{Id } A a_0 a_1$ with the introduction rule

\[
\frac{
\Gamma \vdash \omega : \text{Path } A a_0 a_1[\varphi \mapsto \langle i \rangle a_0]
}{
\Gamma \vdash (\omega, \varphi) : \text{Id } A a_0 a_1
}
\]

and $r(a) = (\langle j \rangle a, 1_{\text{F}}) : \text{Id } A a a$

The intuition is that $\varphi$ specifies where $\omega$ is degenerate.
Identity types

Given $\Gamma \vdash \alpha = (\omega, \varphi) : \text{Id} A a x$ we define

$$\Gamma, i : \mathbb{I} \vdash \alpha^*(i) = (\langle j \rangle \omega (i \wedge j), \varphi \vee (i = 0)) : \text{Id} A a (\alpha i)$$

Using this we define

$$\Gamma, x : A, \alpha : \text{Id} A a x \vdash C \quad \Gamma \vdash \beta : \text{Id} A a b \quad \Gamma \vdash d : C(a, r(a))$$

$$\Gamma \vdash J C b \beta d = \text{comp}^i C(\omega i, \beta^*(i)) \ [\varphi \mapsto d] \ d : C(b, \beta)$$

so that $J C \ a \ r(a) \ d = d$ definitionally
Identity types: univalence

We can also define composition for Id-types and prove that $\text{Id} \ A \ a \ b$ is (Path)-equivalent to $\text{Path} \ A \ a \ b$, so we get

$$(\text{Id} \ U \ A \ B) \simeq (\text{Path} \ U \ A \ B) \simeq (A \simeq B)$$

**But** $\simeq$ is expressed using Path
Identity types: univalence

We can also define composition for Id-types and prove that $\text{Id } A \ a \ b$ is (Path)-equivalent to $\text{Path } A \ a \ b$, so we get

$$(\text{Id } U \ A \ B) \simeq (\text{Path } U \ A \ B) \simeq (A \simeq B)$$

But $\simeq$ is expressed using Path

But as Path and Id are equivalent we get

$$X \text{ Path-contractible } \iff X \text{ Id-contractible}$$
Identity types: univalence

We can also define composition for Id-types and prove that $\text{Id } A \ a \ b$ is (Path)-equivalent to $\text{Path } A \ a \ b$, so we get

$$(\text{Id } U \ A \ B) \simeq (\text{Path } U \ A \ B) \simeq (A \simeq B)$$

But $\simeq$ is expressed using Path

But as Path and Id are equivalent we get

$$X \text{ Path-contractible } \iff X \text{ Id-contractible}$$

So CTT+Id-types is an extension of MLTT+UA
We have a prototype implementation of a proof assistant based on cubical type theory written in Haskell.

We have formalized the proof of univalence in the system:

\[
\text{thmUniv} \quad (\lambda (t : (A X : U) \to \text{Id} U X A \to \text{equiv} X A) (A : U) : (X : U) \to \text{isEquiv} (\text{Id} U X A) (\text{equiv} X A) (t A X) = \text{equivFunFib} U (\lambda (X : U) \to \text{Id} U X A) (\lambda (X : U) \to \text{equiv} X A) (t A) (\text{lemSing1Contr'} U A) (\text{lem1} A)
\]

\[
\text{univalence} \quad (A X : U) : \text{isEquiv} (\text{Id} U X A) (\text{equiv} X A) (\text{transEquiv} A X) = \text{thmUniv} \text{transEquiv} A X
\]

\[
\text{corrUniv} \quad (A B : U) : \text{equiv} (\text{Id} U A B) (\text{equiv} A B) = (\text{transEquiv} B A, \text{univalence} B A)
\]
Normal form of univalence

We can compute and typecheck the normal form of \texttt{thmUniv}:

\begin{verbatim}
module nthmUniv where

import univalence

nthmUniv : (t : (A X : U) → Id U X A → equiv X A) (A : U)
         (X : U) → isEquiv (Id U X A) (equiv X A) (t A X) = \( (t : (A X : U) → (IdP (\langle!0\rangle U) X A) → (Sigma (X → A) (\lambda(f : X → A) → (y : A) → Sigma (Sigma X (\lambda(x : X) → IdP (\langle!0\rangle A) y (f x))) (\lambda(x : Sigma X (\lambda(x0 : X) → IdP (\langle!0\rangle A) y (f x0))) → (y0 : Sigma X (\lambda(x0 : X) → IdP (\langle!0\rangle A) y (f x0))) → IdP (\langle!0\rangle Sigma X (\lambda(x0 : X) → IdP (\langle!0\rangle A) y (f x0))) x y0)))) → \lambda(A x : U) → ...
\end{verbatim}

It takes 8min to compute the normal form, it is about 12MB and it takes 50 hours to typecheck it!
Normal form of univalence

We can compute and typecheck the normal form of \texttt{thmUniv}:

```haskell
module nthmUniv where

import univalence

nthmUniv : (t : (A X : U) → Id U X A → equiv X A) (A : U) → isEquiv (Id U X A) (equiv X A) (t A X) = \(t : (A X : U) → (IdP (\langle!0\rangle U) X A) → (Sigma (X → A) (\lambda f : X → A) → (y : A) → Sigma (Sigma X (\lambda (x : X) → IdP (\langle!0\rangle A) y (f x))) (\lambda (x : Sigma X (\lambda (x0 : X) → IdP (\langle!0\rangle A) y (f x))) y0 : Sigma X (\lambda (x0 : X) → IdP (\langle!0\rangle A) y (f x))) → (y0 : Sigma X (\lambda (x0 : X) → IdP (\langle!0\rangle A) y (f x))) → (y0 : Sigma X (\lambda (x0 : X) → IdP (\langle!0\rangle A) y (f x))) x y0)))) → \lambda (A x : U) → ...```

It takes 8min to compute the normal form, it is about 12MB and it takes 50 hours to typecheck it!
Computing with univalence

In practice this doesn’t seem to be too much of a problem. We have performed multiple experiments:

- Voevodsky’s impredicative set quotients and definition of $\mathbb{Z}$ as a quotient of $\text{nat} \times \text{nat}$
- Fundamental group of the circle (compute winding numbers)
- $\mathbb{T} \simeq S^1 \times S^1$ (by Dan Licata, 60 lines of code)
- ...

Higher inductive types

In the paper we consider two higher inductive types:

- Spheres
- Propositional truncation

In the implementation we have a general schema for defining HITs\textsuperscript{4}

\textsuperscript{4}Warning: composition for recursive HITs is currently incorrect in the implementation, but correct in paper
Integers as a higher inductive types

data int = pos (n : nat)
  | neg (n : nat)
  | zerop <i> [ (i = 0) -> pos zero
                        , (i = 1) -> neg zero ]

sucInt : int -> int = split
  pos n -> pos (suc n)
  neg n -> sucNat n
  where sucNat : nat -> int = split
       zero -> pos one
       suc n -> neg n
  zeroP @ i -> pos one
Torus as a higher inductive types (due to Dan Licata)

data Torus = ptT
  | pathOneT <i> [ (i=0) -> ptT, (i=1) -> ptT ]
  | pathTwoT <i> [ (i=0) -> ptT, (i=1) -> ptT ]
  | squareT <i j> [ (i=0) -> pathOneT @ j
                  , (i=1) -> pathOneT @ j
                  , (j=0) -> pathTwoT @ i
                  , (j=1) -> pathTwoT @ i ]

torus2circles : Torus -> and S1 S1 = split
  ptT -> (base;base)
  pathOneT @ j -> (loop @ j, base)
  pathTwoT @ i -> (base, loop @ i)
  squareT @ i j -> (loop @ j, loop @ i)
Current and future work

- Normalization: Any term of type \( \texttt{nat} \) reduces to a numeral (S. Huber is working on it now)
- Formalize correctness of the model (wip with Mark Bickford in Nuprl)
- General formulation and semantics of higher inductive types (we have an experimental implementation)

https://github.com/mortberg/cubicaltt/
Thank you for your attention!

Figure: Cat filling operation