# Cubical Type Theory: a constructive interpretation of the univalence axiom

**Anders Mörtberg**

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

Institute for Advanced Study, Princeton

June 26, 2016

# Introduction

**Goal:** provide a computational justification for notions from Homotopy Type Theory and Univalent Foundations, in particular the univalence axiom and higher inductive types[1]

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!"*

# Cubical Type Theory

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

$$\begin{array}{lcl}\Gamma & ::= & () \mid \Gamma, x : A \\[1em] t, u, A, B & ::= & x \mid \lambda x : A.\, t \mid t\ u \mid (x : A) \to B \\ & \mid & (t, u) \mid t.1 \mid t.2 \mid (x : A) \times B \\ & \mid & \mathsf{0} \mid \mathsf{s}\ u \mid \mathsf{natrec}\ t\ u \mid \mathtt{nat}\end{array}$$

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:

$$\frac{\Gamma \vdash}{\Gamma, i : \mathbb{I} \vdash}$$

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

$$r, s \quad ::= \quad 0 \mid 1 \mid i \mid 1 - r \mid r \wedge s \mid r \vee s$$

$i, j, k$... formal symbols/names representing directions/dimensions

# Path types

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

$$A(i0) \xrightarrow{\quad A \quad}_{i} A(i1)$$

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

$$\begin{array}{ccc}
A(i0)(j1) & \xrightarrow{A(j1)} & A(i1)(j1) \\
{\scriptstyle A(i0)}\uparrow & A & \uparrow{\scriptstyle A(i1)} \\
A(i0)(j0) & \xrightarrow[A(j0)]{} & A(i1)(j0)
\end{array}$$

and so on...

# Path types

$$\frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash \langle i \rangle\ t : \mathsf{Path}\ A\ t(i0)\ t(i1)} \qquad \frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1 \qquad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t\ r : A}$$

$$\frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1}{\Gamma \vdash t\ 0 = u_0 : A} \qquad \frac{\Gamma \vdash t : \mathsf{Path}\ A\ u_0\ u_1}{\Gamma \vdash t\ 1 = u_1 : A}$$

$$\frac{\Gamma \vdash A \qquad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash (\langle i \rangle\ t)\ r = t(i/r) : A}$$

# Path types

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

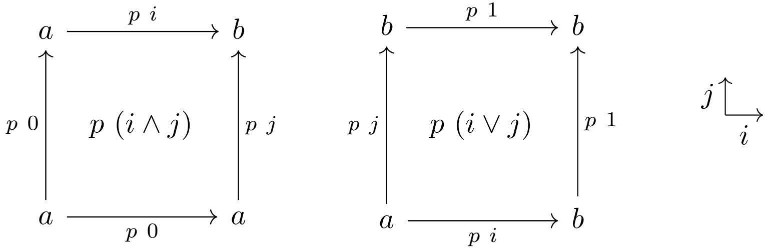
$$t(i0) \xrightarrow{\quad t \quad}_i t(i1) \qquad\qquad t(i0) \xrightarrow{\quad \langle i \rangle\ t \quad} t(i1)$$

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

$$a \xrightarrow{\quad t \quad}_i b \qquad\qquad b \xrightarrow{\quad t\ (1-i) \quad}_i a$$

# Path types: connections

Given $p$ : Path $A\ a\ b$ we can build



# Path types are great!

Function extensionality for path types can be proved as:

$$\frac{\Gamma \vdash f, g : (x : A) \to B \qquad \Gamma \vdash p : (x : A) \to \mathsf{Path}\ B\ (f\ x)\ (g\ x)}{\Gamma \vdash \langle i \rangle\ \lambda x : A.\, p\ x\ i : \mathsf{Path}\ ((x : A) \to B)\ f\ g}$$

# Path types are great!

Given $f : A \to B$ and $p$ : Path $A\ a\ b$ we can define:

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

satisfying definitionally:

$$\begin{array}{lllll} \texttt{ap} & \text{id} & p & = & p \\ \texttt{ap} & (f \circ g) & p & = & \texttt{ap}\ f\ (\texttt{ap}\ g\ p) \end{array}$$

This way we get new ways for reasoning about equality: inline ap, funext, symmetry... with new definitional equalities

# Path types are great!

We can also prove contractibility of singletons[2]:

$$\frac{\Gamma \vdash p : \mathsf{Path}\ A\ a\ b}{\Gamma \vdash \langle i \rangle\ (p\ i, \langle j \rangle\ p\ (i \wedge j)) : \mathsf{Path}\ ((x : A) \times (\mathsf{Path}\ A\ a\ x))\ (a, 1_a)\ (b, p)}$$

But we cannot yet compose paths...

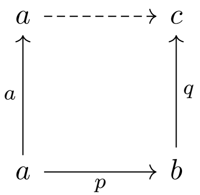
[2] or *"Vacuum Cleaner Power Cord Principle"*

# Kan composition operations

We want to be able to compose paths:

$$a \xrightarrow{p} b \qquad\qquad b \xrightarrow{q} c$$

We do this by computing the dashed line in:




In general this corresponds to computing the missing sides of n-dimensional cubes

# 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

| | |
|---|---|
| $i : \mathbb{I}, (i = 0) \vee (i = 1) \vdash A$ | $A(i0) \bullet \qquad \bullet A(i1)$ |
| $i\ j : \mathbb{I}, (i = 0) \vee (i = 1) \vee (j = 0) \vdash A$ | $\begin{array}{ccc} A(i0)(j1) & & A(i1)(j1) \\ {\scriptstyle A(i0)}\uparrow & & \uparrow{\scriptstyle A(i1)} \\ A(i0)(j0) & \xrightarrow[A(j0)]{} & A(i1)(j0) \end{array}$ |

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

# Partial elements

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

$$\frac{\Gamma \vdash A}{\Gamma, \varphi \vdash A}$$

Contexts $\Gamma$ are modeled by cubical sets

Restriction operation correspond to a **cofibration**:

$$\Gamma, \varphi \to \Gamma$$

# Face lattice

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 $(i = 0) \vee (i = 1)$, an element $b : A[\varphi \mapsto a]$ is a line:

$$a(i0) \xrightarrow{\quad b \quad} a(i1)$$

# Box principle

We can now formulate the box principle in type theory:

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \qquad \Gamma \vdash a_0 : A(i0)[\varphi \mapsto u(i0)]}{\Gamma \vdash \mathsf{comp}^i\ A\ [\varphi \mapsto u]\ a_0 : A(i1)[\varphi \mapsto u(i1)]}$$

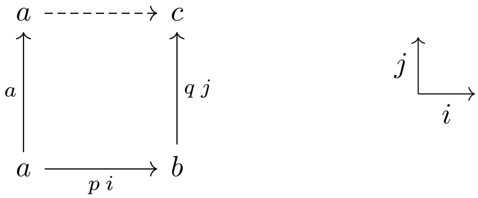
$u$ is a partial path connected at $i = 0$ specifying the sides of the box
$a_0$ is the bottom of the box
$\mathsf{comp}^i$ witnesses that $u$ is connected at $i = 1$

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:

$$\frac{\Gamma \vdash p : \mathsf{Path}\ A\ a\ b \qquad \Gamma \vdash q : \mathsf{Path}\ A\ b\ c}{\Gamma \vdash \langle i \rangle\ \mathsf{comp}^j\ A\ [(i=0) \mapsto a, (i=1) \mapsto q\ j]\ (p\ i) : \mathsf{Path}\ A\ a\ c}$$



# Kan composition: transport

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

$$\frac{\Gamma, i : \mathbb{I} \vdash A \qquad \Gamma \vdash a : A(i0)}{\Gamma \vdash \mathsf{transport}^i\ A\ a = \mathsf{comp}^i\ A\ []\ a : A(i1)}$$

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

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

showing that the canonical map

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

is an equivalence

# Univalence?

The following is an alternative characterization of univalence:

**Univalence axiom**

*For any type* $A : \mathsf{U}$ *the type* $(T : \mathsf{U}) \times \mathsf{Equiv}\ T\ A$ *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* isContr $A$ *is inhabited iff we have an operation:*

$$\frac{\Gamma, \varphi \vdash u : A}{\Gamma \vdash \mathsf{ext}\ [\varphi \mapsto u] : A[\varphi \mapsto u]}$$

# Univalence?

**Lemma**

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

$$\frac{\Gamma, \varphi \vdash u : A}{\Gamma \vdash \mathsf{ext}\ [\varphi \mapsto u] : A[\varphi \mapsto u]}$$

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

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

extends to a total element

# Example: unary and binary numbers

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

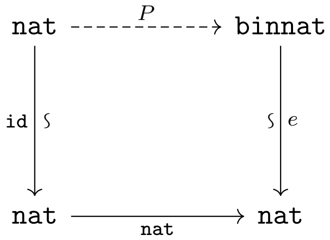
$$e : \texttt{binnat} \to \texttt{nat}$$

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

$$\texttt{nat} \overset{P}{\dashrightarrow} \texttt{binnat}$$

# Example: unary and binary numbers

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

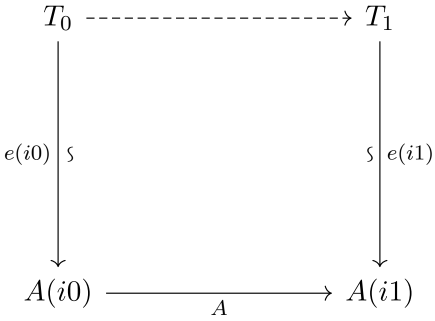



We write

$$i : \mathbb{I} \vdash P = \text{Glue } [(i = 0) \mapsto (\texttt{nat}, \texttt{id}), (i = 1) \mapsto (\texttt{binnat}, e)] \ \texttt{nat}$$

# Glue: more generally

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



# Glue: even more generally

We assume that we are given

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

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

# Glue: even more generally

We assume that we are given

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

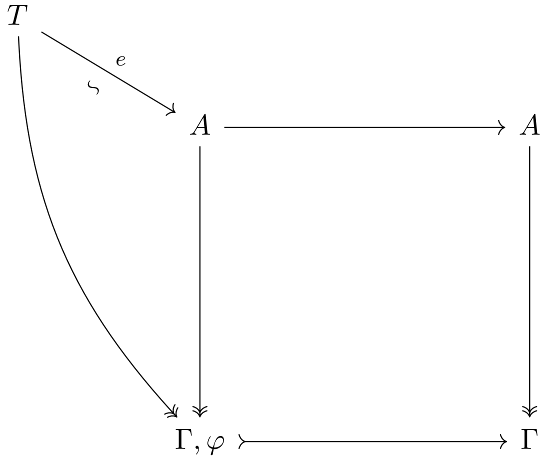
From this we define

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

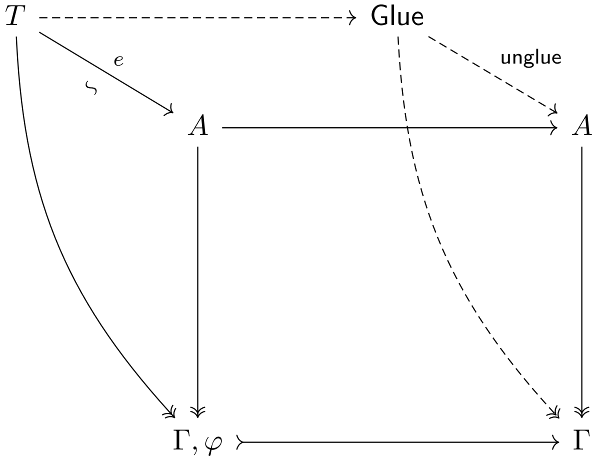
such that $\mathsf{Glue}\ [\varphi \mapsto (T, e)]\ A$ and unglue are extensions of $T$ and $e$:

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

# Glue: diagrammatically



# Glue: diagrammatically



# Rules for Glue

$$\frac{\Gamma \vdash A \qquad \Gamma, \varphi \vdash T \qquad \Gamma, \varphi \vdash e : \mathsf{Equiv}\ T\ A}{\Gamma \vdash \mathsf{Glue}\ [\varphi \mapsto (T, e)]\ A}$$

$$\frac{\Gamma, \varphi \vdash e : \mathsf{Equiv}\ T\ A \qquad \Gamma, \varphi \vdash t : T \qquad \Gamma \vdash a : A[\varphi \mapsto e\ t]}{\Gamma \vdash \mathsf{glue}\ [\varphi \mapsto t]\ a : \mathsf{Glue}\ [\varphi \mapsto (T, e)]\ A}$$

$$\frac{\Gamma \vdash b : \mathsf{Glue}\ [\varphi \mapsto (T, e)]\ A}{\Gamma \vdash \mathsf{unglue}\ b : A}$$

together with equality judgments

# Composition for Glue

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

$$\Gamma, \psi, i : \mathbb{I} \vdash b : B \qquad\qquad \Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$$

# Composition for Glue

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

$$\Gamma, \psi, i : \mathbb{I} \vdash b : B \qquad \Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$$

The algorithm computes

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

such that:

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

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

# Composition for Glue

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

$$\Gamma, \psi, i : \mathbb{I} \vdash b : B \qquad\qquad \Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$$

The algorithm computes

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

such that:

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

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

```
comp(Glue [phi ↦ T,f] A) =
\H,sigma,psi,b, b0.
  let a = unglue(b) in
  let a0 = unglue(b0) in
  let a'1 = comp (cA)sigma [psi ↦ a ] a0 in
  let t'1 = comp (cT)sigma [psi ↦ b] b0 in
  let g = (f.1)sigma in
  let w = pres g [psi ↦ b] b0 in
  let phi' = forall (phi)sigma in
  let phi1 = (phi)sigma[1] in
  let st = if phi' then t'1 else b[1] in
  let sw = if phi' then w else <> ((g b)[1])p in
  let cF = fiber-comp (H, phi1) (cT)sigma[1] (cA)sigma[1] g[1] a'1 in
  let z = equiv cF g[1] [phi' ∨ psi ↦ (st,sw)] a'1 in
  let t1 = z.1 in
  let alpha = z.2 in
  let x = if (phi1)p then (alpha)p @ q else a[1]p in
  let a1 = comp (cA)sigma[1]p [phi1 ∨ psi ↦ x] a'1 in
  glue [phi1 ↦ t1 ] a1
```

# Composition for the universe from Glue

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

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

Using transport we can construct[3]

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

[3] Note that $\mathsf{equiv}^i\ E$ 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

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

Using transport we can construct[3]

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

Using this we can define the composition for the universe:

$$\begin{aligned}\Gamma \vdash\ & \mathsf{comp}^i\ \mathsf{U}\ [\varphi \mapsto E]\ A = \\ & \quad \mathsf{Glue}\ [\varphi \mapsto (E(i1), \mathsf{equiv}^i\ E(i/1-i))]\ A : \mathsf{U}[\varphi \mapsto E(i1)]\end{aligned}$$

[3] Note that $\mathsf{equiv}^i\ E$ binds $i$ in $E$

# Proof of univalence

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

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

extends to a total element

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

# Proof of univalence

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

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

extends to a total element

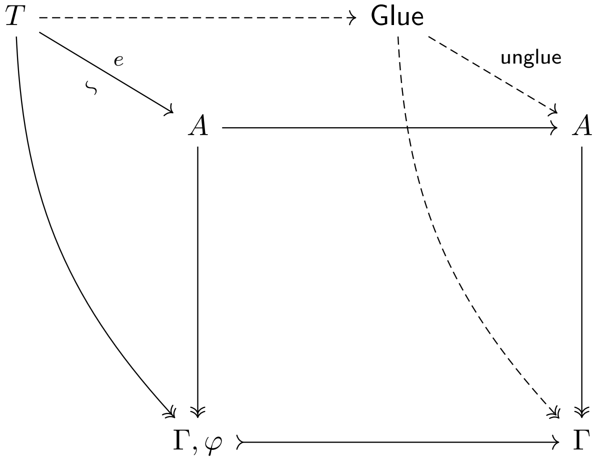
$$\Gamma \vdash (T', e') : ((T' : \mathsf{U}) \times \mathsf{Equiv}\ T'\ A)[\varphi \mapsto (T, e)]$$

This is exactly what Glue gives us!

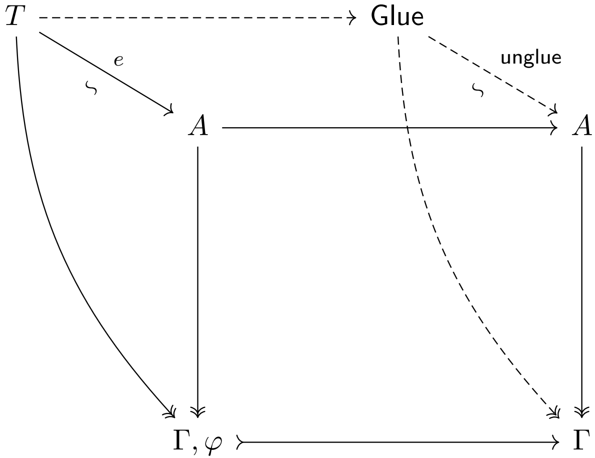
$$T' = \mathsf{Glue}\ [\varphi \mapsto (T, e)]\ A \qquad\qquad e' = (\mathsf{unglue}, ?)$$

For ? we need to prove that unglue is an equivalence

# Proof of univalence



# Proof of univalence



# Proof of univalence

So we get:

**Corollary**

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

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

**Corollary**

*For any term*

$$t : (A\ B : \mathsf{U}) \to \mathsf{Path}\ \mathsf{U}\ A\ B \to \mathsf{Equiv}\ A\ B$$

*the map* $t\ A\ B : \mathsf{Path}\ \mathsf{U}\ A\ B \to \mathsf{Equiv}\ A\ B$ *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 **not** hold definitionally

This is because the computation rule

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

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

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

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

# Identity types

We define a type Id $A\ a_0\ a_1$ with the introduction rule

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

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

The intuition is that $\varphi$ specifies where $\omega$ is degenerate

# Identity types

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

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

Using this we define

$$\frac{\Gamma, x : A, \alpha : \mathsf{Id}\ A\ a\ x \vdash C \qquad \Gamma \vdash \beta : \mathsf{Id}\ A\ a\ b \qquad \Gamma \vdash d : C(a, \mathsf{r}(a))}{\Gamma \vdash J\ C\ b\ \beta\ d = \mathsf{comp}^i\ C(\omega\ i, \beta^*(i))\ [\varphi \mapsto d]\ d : C(b, \beta)}$$

so that $J\ C\ a\ \mathsf{r}(a)\ d = d$ definitionally

# Identity types: univalence

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

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

**But** $\simeq$ is expressed using Path

# Identity types: univalence

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

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

**But** $\simeq$ is expressed using Path

But as Path and Id are equivalent we get

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

# Identity types: univalence

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

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

**But** $\simeq$ is expressed using Path

But as Path and Id are equivalent we get

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

So CTT+Id-types is an extension of MLTT+UA

# cubicaltt

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:

```
thmUniv (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) =
    equivFunFib U (λ(X : U) → Id U X A) (λ(X : U) → equiv X A)
                  (t A) (lemSinglContr' U A) (lem1 A)

univalence (A X : U) : isEquiv (Id U X A) (equiv X A) (transEquiv A X) =
  thmUniv transEquiv A X

corrUniv (A B : U) : equiv (Id U A B) (equiv A B) =
  (transEquiv B A,univalence B A)
```

# Normal form of univalence

We can compute and typecheck the normal form of `thmUniv`:

```
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 (<!0> U) X A) → (Sigma (X → A) (λ(f : X → A) → (y : A)
  → Sigma (Sigma X (λ(x : X) → IdP (<!0> A) y (f x))) (λ(x : Sigma X
  (λ(x : X) → IdP (<!0> A) y (f x))) → (y0 : Sigma X (λ(x0 : X) →
  IdP (<!0> A) y (f x0))) → IdP (<!0> Sigma X (λ(x0 : X) → IdP (<!0>
  A) y (f x0))) x y0)))) → λ(A x : U) → ...
```

# Normal form of univalence

We can compute and typecheck the normal form of `thmUniv`:

```
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 (<!0> U) X A) → (Sigma (X → A) (λ(f : X → A) → (y : A)
  → Sigma (Sigma X (λ(x : X) → IdP (<!0> A) y (f x))) (λ(x : Sigma X
  (λ(x : X) → IdP (<!0> A) y (f x))) → (y0 : Sigma X (λ(x0 : X) →
  IdP (<!0> A) y (f x0))) → IdP (<!0> Sigma X (λ(x0 : X) → IdP (<!0>
  A) y (f x0))) x y0)))) → λ(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 Z as a quotient of `nat * nat`
- Fundamental group of the circle (compute winding numbers)
- Z as a HIT
- $\mathbb{T} \simeq \mathbb{S}^1 \times \mathbb{S}^1$ (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[4]

[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 `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