

# 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

May 19, 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<sup>1</sup>

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

---

<sup>1</sup>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
- ④ Higher inductive types

# 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,  $\mathbb{I}$ :

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

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

## Path types

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

$$A(i0) \xrightarrow{A} A(i1)$$

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

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

and so on...

## Path types

$$\frac{\Gamma \vdash A \quad \Gamma, i : \mathbb{I} \vdash t : A}{\Gamma \vdash \langle i \rangle t : \text{Path } A \ t(i0) \ t(i1)}$$

$$\frac{\Gamma \vdash t : \text{Path } A \ u_0 \ u_1 \quad \Gamma \vdash r : \mathbb{I}}{\Gamma \vdash t \ r : A}$$

$$\frac{\Gamma \vdash t : \text{Path } A \ u_0 \ u_1}{\Gamma \vdash t \ 0 = u_0 : A}$$

$$\frac{\Gamma \vdash t : \text{Path } A \ u_0 \ u_1}{\Gamma \vdash t \ 1 = u_1 : A}$$

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

# Path types

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

$$t(i0) \xrightarrow{t} \rightarrow_i t(i1)$$

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

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

$$a \xrightarrow{t} \rightarrow_i b$$

$$b \xrightarrow{t (1-i)} \rightarrow_i a$$



## Path types are great!

Function extensionality for path types can be proved as:

$$\frac{\Gamma \vdash f, g : (x : A) \rightarrow B \quad \Gamma \vdash p : (x : A) \rightarrow \text{Path } B (f x) (g x)}{\Gamma \vdash \langle i \rangle \lambda x : A. p x i : \text{Path } ((x : A) \rightarrow B) f g}$$

We can also prove contractibility of singletons<sup>2</sup>:

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

But we cannot yet compose paths...

---

<sup>2</sup>or “Vacuum Cleaner Power Cord Principle”

## Kan composition operations

We want to be able to compose paths:

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

We do this by computing the dashed line in:

$$\begin{array}{ccc} a & \text{-----} & c \\ \uparrow & & \uparrow \\ a & & b \\ & \xrightarrow{p} & \\ & & \end{array}$$

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

$i : \mathbb{I}, (i = 0) \vee (i = 1) \vdash A$	$A(i0) \bullet$	$\bullet A(i1)$
$i j : \mathbb{I}, (i = 0) \vee (i = 1) \vee (j = 0) \vdash A$	$A(i0)(j1)$ $\uparrow$ $A(i0)$	$A(i1)(j1)$ $\uparrow$ $A(i1)$
	$A(i0)(j0)$	$A(i1)(j0)$
	$\xrightarrow{A(j0)}$	

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**:

$$\Gamma, \varphi \rightarrow \Gamma$$

## 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:

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

## Box principle

We can now formulate the box principle in type theory:

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

$u$  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  $i = 1$

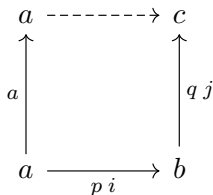
The equality judgments for the composition operation are defined by induction on  $A$  – this is the main part of the system



## Kan composition: example

With composition we can justify transitivity of path types:

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



## Kan composition: transport

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

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

Together with contractibility of singletons we can prove path induction

# 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:

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

showing that the canonical map

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

is an equivalence

# Univalence?

The following is an alternative characterization of univalence:

## Univalence axiom

*For any type  $A : \mathcal{U}$  the type  $(T : \mathcal{U}) \times \text{Equiv } T \ A$  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?

We can prove:

## Lemma

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

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

# Univalence?

We can prove:

## Lemma

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

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

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

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

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

$$e : \text{binnat} \rightarrow \text{nat}$$

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

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

## Example: unary and binary numbers

$P$  should also store information about  $e$ ,<sup>3</sup> we achieve this by “glueing”:

$$\begin{array}{ccc} \mathbf{nat} & \overset{P}{\dashrightarrow} & \mathbf{binnat} \\ \downarrow \text{id} \wr & & \downarrow \wr e \\ \mathbf{nat} & \xrightarrow{\text{nat}} & \mathbf{nat} \end{array}$$

We write

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

---

<sup>3</sup> “Evidence matters!” - Bob Harper on Monday

## 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:

$$\begin{array}{ccc} T_0 & \text{-----} & T_1 \\ \downarrow e(i0) \wr s & & \downarrow s \wr e(i1) \\ A(i0) & \xrightarrow{A} & A(i1) \end{array}$$

## Glue: even more generally

We assume that we are given

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

## Glue: even more generally

We assume that we are given

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

From this we define

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

## Glue: even more generally

We assume that we are given

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

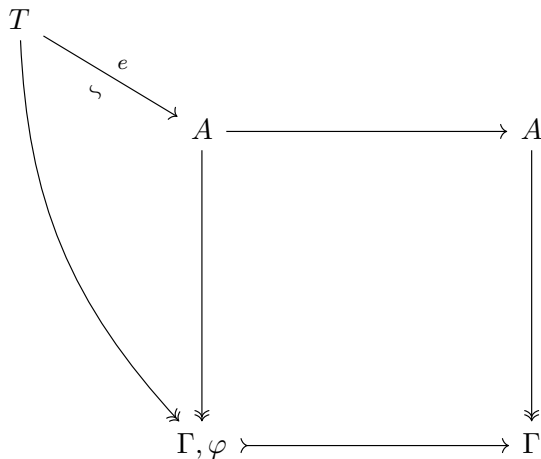
From this we define

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

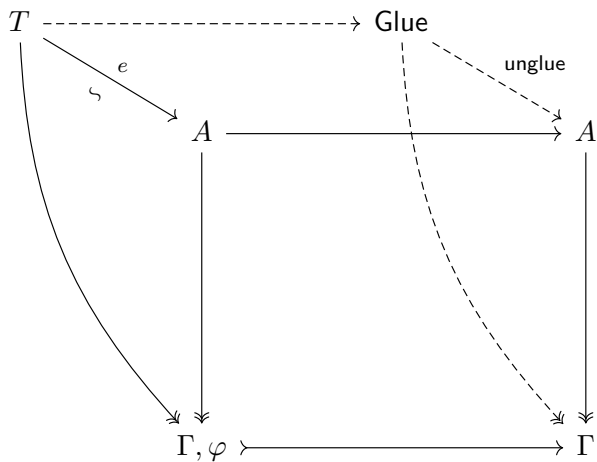
such that  $\text{Glue } [\varphi \mapsto (T, e)] A$  and  $\text{unglue}$  are extensions of  $T$  and  $e$ :

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

## Glue: diagrammatically



# Glue: diagrammatically





## Rules for Glue

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

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

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

together with equality judgments

## Composition for Glue

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

$$\Gamma, \psi, i : \mathbb{I} \vdash b : B$$

$$\Gamma \vdash b_0 : B(i0)[\psi \mapsto b(i0)]$$

## Composition for Glue

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

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

The algorithm computes

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

such that:

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

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

## Composition for Glue

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

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

The algorithm computes

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

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 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

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

Using transport we can construct<sup>4</sup>

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

---

<sup>4</sup>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

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

Using transport we can construct<sup>4</sup>

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

Using this we can define the composition for the universe:

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

---

<sup>4</sup>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

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

extends to a total element

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



## Proof of univalence

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

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

extends to a total element

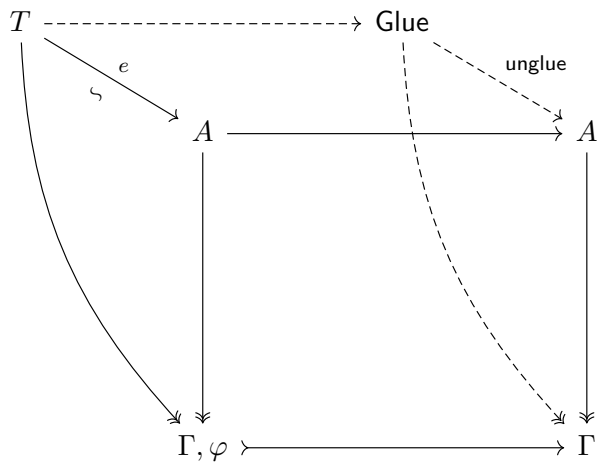
$$\Gamma \vdash (T', e') : ((T' : \mathbf{U}) \times \mathbf{Equiv} \ T' \ A)[\varphi \mapsto (T, e)]$$

This is exactly what Glue gives us!

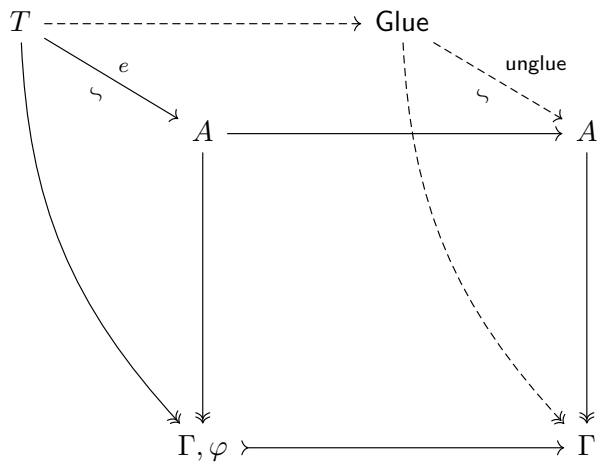
$$T' = \mathbf{Glue} \ [\varphi \mapsto (T, e)] \ A \qquad e' = (\mathbf{unglue}, ?)$$

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  $A : \mathcal{U}$  the type  $(T : \mathcal{U}) \times \text{Equiv } T \ A$  is contractible*

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

## Corollary

*For any term*

$$t : (A \ B : \mathcal{U}) \rightarrow \text{Path } \mathcal{U} \ A \ B \rightarrow \text{Equiv } A \ B$$

*the map  $t \ A \ B : \text{Path } \mathcal{U} \ A \ B \rightarrow \text{Equiv } A \ B$  is an equivalence*

# 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  $\mathbb{Z}$  as a quotient of  $\text{nat} * \text{nat}$
- Fundamental group of the circle (compute winding numbers)
- $\mathbb{Z}$  as a HIT
- $\mathbb{T} \simeq \mathbb{S}^1 \times \mathbb{S}^1$  (by Dan Licata, 60 lines of code)
- ...

Please contribute!

<https://github.com/mortberg/cubicaltt/>



## 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)

# Thank you for your attention!



Figure: Cat filling operation