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
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!”
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, \( \mathbb{I} \):

\[
\begin{align*}
\text{formal symbols/names representing directions/dimensions} & \quad 0 \mid 1 \mid \bar{i} \mid 1 - r \mid r \land s \mid r \lor s \\
i, j, k \ldots &
\end{align*}
\]
Path types

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

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

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

\[
\begin{array}{c}
\begin{array}{c}
A(i_0)(j_1) \\
A(i_0)
\end{array}
\xrightarrow{A(j_1)}
\begin{array}{c}
A(i_1)(j_1) \\
A(i_1)
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
A(i_0)(j_0) \\
A(i_0)
\end{array}
\xrightarrow{A(j_0)}
\begin{array}{c}
A(i_1)(j_0) \\
A(i_1)
\end{array}
\end{array}
\]

and so on...
Path types

\[
\begin{align*}
\Gamma \vdash A & \quad \Gamma, i : \mathbb{I} \vdash t : A \\
\Gamma \vdash \langle i \rangle t : \text{Path } A \ t(i0) \ t(i1) \\
\Gamma \vdash t : \text{Path } A \ u_0 \ u_1 & \quad \Gamma \vdash r : \mathbb{I} \\
\Gamma \vdash t r : A \\
\Gamma \vdash t 0 = u_0 : A & \quad \Gamma \vdash t 1 = u_1 : A \\
\Gamma \vdash A & \quad \Gamma, i : \mathbb{I} \vdash t : A \\
\Gamma \vdash (\langle i \rangle t) r = t(i/r) : A
\end{align*}
\]
Path types

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

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

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

$$a \xrightarrow{t} b \quad b \xrightarrow{t(1-i)} a$$
Path types are great!

Function extensionality for path types can be proved as:

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

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:

\[ a \xrightarrow{p} b \quad b \xrightarrow{q} c \]

We do this by computing the dashed line in:

\[
\begin{array}{ccc}
  a & \longrightarrow & c \\
  ^{a} & & \downarrow^{q} \\
  \downarrow^{a} & & \\
  a & \longrightarrow & b
\end{array}
\]

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

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

$$a(i0) \overset{b}{\rightarrow} a(i1)$$
Box principle

We can now formulate the box principle in type theory:

\[
\Gamma, i : \Pi \vdash A \\
\Gamma, \varphi, i : \Pi \vdash u : A \\
\Gamma \vdash a_0 : A(i0)[\varphi \mapsto u(i0)]
\]

\[
\Gamma \vdash \text{comp}^i A[\varphi \mapsto u] \quad a_0 : A(i1)[\varphi \mapsto u(i1)]
\]

\(u\) 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
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 $\varphi = 0_F$ corresponds to transport:

\[
\frac{\Gamma, i : \mathbb{1} \vdash A \quad \Gamma \vdash a : A(i0)}{\Gamma \vdash \text{transport}^i A \ a = \text{comp}^i A \ [\] \ 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:

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

showing that the canonical map

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

is an equivalence
The following is an alternative characterization of univalence:

**Univalence axiom**

For any type $A : U$ the type $(T : U) \times \text{Equiv } T \ A$ 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.
We can prove:

**Lemma**

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

\[
\Gamma, \varphi \vdash u : A \\
\Gamma \vdash \text{ext} [\varphi \mapsto u] : A[\varphi \mapsto u]
\]
Univalence?

We can prove:

**Lemma**

The type isConstr $A$ 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

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

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 $P(i1) = \text{binnat}$:

$$\text{nat} \longrightarrow^{P} \text{binnat}$$
Example: unary and binary numbers

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

\[
\begin{array}{c}
\text{nat} \xrightarrow{P} \text{binnat} \\
\downarrow \text{id} \hspace{2cm} \downarrow e \\
\text{nat} \xrightarrow{\text{nat}} \text{nat}
\end{array}
\]

We write

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

\[^{3} \text{“Evidence matters!” - Bob Harper on Monday}\]
In the case when $\varphi$ is $(i = 0) \lor (i = 1)$ the glueing operation can be illustrated as the dashed line in:

$$
\begin{array}{c}
T_0 \longrightarrow \longrightarrow \rightarrow T_1 \\
\downarrow \hspace{4cm} \downarrow \\
A(i0) \hspace{4cm} A(i1)
\end{array}
$$

$$
e(i0) \neglop \neglop e(i1)
\vspace{1cm}
A
\vspace{1cm}
A$$
Glue: even more generally

We assume that we are given

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

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

\[
\begin{array}{c}
A \\
\rightarrow \\
A
\end{array}
\]
Glue: diagrammatically

\[ T \rightarrow \text{Glue} \]

\[ \Gamma, \varphi \rightarrow \Gamma \]

\[ e \]

\[ \varsigma \]

\[ \text{unglue} \]
### Rules for Glue

\[
\begin{align*}
\Gamma &\vdash A \\
\Gamma, \varphi &\vdash T \\
\Gamma, \varphi &\vdash e : \text{Equiv } T \ A
\end{align*}
\]

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

\[
\begin{align*}
\Gamma, \varphi &\vdash e : \text{Equiv } T \ A \\
\Gamma, \varphi &\vdash t : T \\
\Gamma &\vdash a : A[\varphi \mapsto e \ t]
\end{align*}
\]

\[
\Gamma \vdash \text{glue } [\varphi \mapsto t] \ a : \text{Glue } [\varphi \mapsto (T, e)] \ A
\]

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

\[
\Gamma \vdash \text{unglue } b : A
\]

together with equality judgments
Composition for Glue

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

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

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

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

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 $i$. 

Anders Mörterberg

Glue and univalence

May 19, 2016
Composition for Glue

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

$$\Gamma, \psi, i : \Pi \vdash b : B$$

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

The algorithm computes

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

such that:

$$\Gamma \vdash b_1 : B(i1)[\psi \mapsto b(i1)]$$

$$\Gamma, \delta \vdash b_1 : T(i1)$$

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

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

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

\[ E(i0) = A \quad \quad \quad \quad E(i1) = B \]

Using transport we can construct\(^4\)

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

---

\(^4\)Note that equiv\(^i\) $E$ binds $i$ in $E$
Composition for the universe from Glue

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

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

Using transport we can construct\(^4\)

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

Using this we can define the composition for the universe:

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

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

\(^4\)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!

\[ T' = \text{Glue } [\varphi \mapsto (T, e)] \ A \quad e' = (\text{un glue}, ?) \]

For ? we need to prove that unglue is an equivalence
Proof of univalence

\[
\begin{array}{c}
T \\
\downarrow\scriptstyle\sim \\
A \\
\downarrow \\
\Gamma, \phi \\
\downarrow \\
\Gamma
\end{array}
\xrightarrow{e} \quad \begin{array}{c}
\text{Glue} \\
\downarrow \text{unglue} \\
A \\
\downarrow \\
\Gamma
\end{array}
\]
Proof of univalence

\[ T \rightarrow \text{Glue} \]

\[ e \]

\[ \sim \]

\[ \Gamma, \varphi \rightarrow \Gamma \]

\[ A \rightarrow A \]

\[ \text{unglue} \]
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*
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:

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

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

\[
\text{corrUniv} \ (A B : U) : \text{equiv} \ (\text{Id} \ U \ A \ B) \ (\text{equiv} \ A \ B) = \\
(\text{transEquiv} \ B \ A, \text{univalence} \ B \ A)
\]
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) = \(\lambda t : (A X : U) \to 
(IdP (\langle!0\rangle U) X A) \to (Sigma (X \to A) (\lambda (f : X \to A) \to (y : A) \to 
Sigma (Sigma X (\lambda (x : X) \to IdP (\langle!0\rangle A) y (f x))) (\lambda (x : Sigma X 
(\lambda (x : X) \to IdP (\langle!0\rangle A) y (f x))) \to (y0 : Sigma X (\lambda (x0 : X) \to 
IdP (\langle!0\rangle A) y (f x0)) \to IdP (\langle!0\rangle Sigma X (\lambda (x0 : X) \to IdP (\langle!0\rangle 
A) y (f x0))) x y0)))) \to \lambda (A x : U) \to ...
\end{verbatim}

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

Anders Mörberg
Normal form of univalence

We can compute and typecheck the normal form of \texttt{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 $\text{nat} \times \text{nat}$
- Fundamental group of the circle (compute winding numbers)
- $\mathbb{T} \simeq \mathbb{S}^1 \times \mathbb{S}^1$ (by Dan Licata, 60 lines of code)
- ...

Please contribute!

https://github.com/mortberg/cubicaltt/
Current and future work

- Normalization: Any term of type \( \mathbb{nat} \) 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