Computational Higher-Dimensional Type Theory

Carlo Angiuli\textsuperscript{1}  Robert Harper\textsuperscript{1}  Todd Wilson\textsuperscript{2}

\textsuperscript{1}Carnegie Mellon University

\textsuperscript{2}California State University, Fresno

January 20, 2017
Homotopy Type Theory (HoTT)

Extends Martin-Löf dependent type theory with:

- Univalence axiom.
- Higher inductive types.

Captures higher-dimensional (homotopical, topological) structure.

Although this talk isn’t about HoTT, let’s start by reviewing it.
Homotopy Type Theory (HoTT)

Useful for constructive, mechanized (in Coq/Agda/Lean) proofs of theorems from algebraic topology and homotopy theory.

- Seifert-van Kampen theorem (Favonia, Shulman).
- Eilenberg-Mac Lane spaces (Licata, Finster).
- Mayer-Vietoris theorem (Cavallo).
- Blakers-Massey theorem (Favonia, Finster, Licata, Lumsdaine).
- Cayley-Dickson construction (Buchholtz, Rijke).
Identity type $\text{Id}_A(M, N)$ says that $M, N$ are equal.

$\text{Id}_A(M, N) \implies$ can always replace $M$ with $N$.

$\text{Id}_{\text{Type}}(A, B) \implies$ can coerce elements of $A$ to $B$.

Univalence*: Any isomorphism between $A, B$ yields $\text{Id}_{\text{Type}}(A, B)$.

Univalence says all isomorphisms yield proofs of identity, whose coercions are implemented by the isomorphism.
Higher Inductive Types

Inductive types with constructors for $A$ and $\text{Id}_A(M, N)$!

\[ \Gamma \vdash \text{base} : S^1 \]
\[ \Gamma \vdash \text{loop} : \text{Id}_{S^1}(\text{base}, \text{base}) \]

We draw this HIT as a circle because it actually behaves like one, when identity proofs are interpreted as paths.
Higher Inductive Types

Inductive types with constructors for $A$ and $\text{Id}_A(M, N)$!

\[
\begin{align*}
\Gamma & \vdash \text{base} : S^1 \\
\Gamma & \vdash \text{loop} : \text{Id}_{S^1}(\text{base}, \text{base})
\end{align*}
\]

Higher-dimensional interpretation: identity = paths.

We draw this HIT as a circle because it actually behaves like one, when identity proofs are interpreted as paths.
Propositions-as-Types Correspondence

Also known as the Curry-Howard isomorphism, or the Brouwer-Heyting-Kolmogorov explanation.

\[
\text{logics} \iff \text{programming languages} \\
\text{propositions} \iff \text{types} \\
\text{proofs of a proposition} \iff \text{programs of a type}
\]

A key feature of type theory is the correspondence between proofs and programs.
Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can’t just postulate new programs in existing types!

datatype bool = true | false

if ... then 0 else 1 : int

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.
Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can’t just postulate new programs in existing types!

datatype bool = true | false | file_not_found

if file_not_found then 0 else 1 : int

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.
Proofs as Programs?

Adding new axioms (UA, HITs) is fine in a logic, but in a PL, you can’t just postulate new programs in existing types!

```ml
datatype bool = true | false | file_not_found

if file_not_found then 0 else 1 : int
```

Destroys int!

---

Axioms disrupt PAT, causing existing programs to become stuck. This ruins computation at every type.
Proofs as Programs?

Exactly what happens with UA+HITs in HoTT: new $\text{Id}_A(M, N)$
proofs not handled by the $\text{Id}$ eliminator!

Inconvenient, even if you only care about logic.
Guillaume Brunerie successfully computed an invariant as $\mathbb{Z}/k\mathbb{Z}$ where $\vdash k : \mathbb{N}$ (14 pages, 2013).

Required a PhD thesis (129 pages, 2016) to show $k = 2$.

Propositions-as-types $\implies k$ computes to 2!
Computational Cubical Type Theory

We define a (non-HoTT) higher-dimensional type theory for which propositions-as-types works. Core idea is to extend:

Nuprl, Constable, et al. (1985–). Computational type theory.

Computational Type Theory

Given a programming language \( M \downarrow V \), types are defined as classifications of programs according to their behavior.

\[
\begin{align*}
\cdot \quad & M \in \text{bool} \iff M \downarrow \text{true} \text{ or } M \downarrow \text{false} \\
\cdot \quad & M \in A \rightarrow B \iff M \downarrow \lambda a. M' \land \\
& \forall N \in A, \ M'[N/a] \in B
\end{align*}
\]

Closely related to logical relations and to refinements!

We adopt the \( \Rightarrow \) and \( \in \) notation to avoid confusion with other type theories.
The familiar rules of type theory hold relative to these definitions!

\[
M \in \text{bool} \rightarrow \text{bool} \quad N \in \text{bool} \\
\overline{M N \in \text{bool}}
\]
The familiar rules of type theory hold relative to these definitions!

\[
\begin{align*}
M \in \text{bool} \rightarrow \text{bool} & \quad N \in \text{bool} \\
M \ N \in \text{bool}
\end{align*}
\]

\[
\Rightarrow
\]

\[
\begin{align*}
M \downarrow \lambda a. M' \quad & \forall N' \in \text{bool}, \ M'[N'/a] \in \text{bool} \\
N \downarrow \text{true or false}
\end{align*}
\]

\[
M \ N \downarrow \text{true or false}
\]
Computational Type Theory

Constructive (à la Brouwer): truth is defined by algorithms.

- Not defined by enumerating proof rules.
- Programs have many types, some more obvious than others!
  (Ranges from “read the program” to “prove a theorem.”)
Types Internalize Judgments

Types internalize concepts present in the judgmental framework.

\[
\begin{align*}
A \text{ true} & \quad B \text{ true} \\
\hline
A \land B \text{ true} \\
\hline
A \text{ true} & \quad B \text{ true} \\
\hline
A \lor B \text{ true} & \quad A \lor B \text{ true}
\end{align*}
\]

Writing multiple premises to a rule implicitly invokes conjunction; writing multiple rules with the same conclusion implicitly invokes disjunction.
Types Internalize Judgments

Originally, closed $\text{Id}_A(M, N)$ determined by equality judgment.

In HoTT,

- $\text{Id}_{S^1}(\text{base}, \text{base})$ determined by definition of $S^1$.
- $\text{Id}_{\text{Type}}(A, B)$ determined by isomorphisms.
Path Judgments

*Canonicity for 2-Dimensional Type Theory*, Licata and Harper (POPL 2012): Define a judgment for paths.

\[
\Gamma \vdash M : A
\]

\[
\Gamma \vdash P : M \simeq N : A
\]

We can organize iterated path judgments cubically.
Path Judgments

*Canonicity for 2-Dimensional Type Theory*, Licata and Harper (POPL 2012): Define a judgment for paths.

\[ \Gamma \vdash M : A \]

\[ \Gamma \vdash P : M \simeq N : A \]

\[ \Gamma \vdash H : P \simeq Q : M \simeq N : A \]

We can organize iterated path judgments cubically.
Path Judgments

*Canonicity for 2-Dimensional Type Theory*, Licata and Harper (POPL 2012): Define a judgment for paths.

\[ \Gamma \vdash M : A \]

\[ \Gamma \vdash P : M \simeq N : A \]

\[ \Gamma \vdash H : P \simeq Q : M \simeq N : A \]
Path Judgments

*Canonicity for 2-Dimensional Type Theory*, Licata and Harper (POPL 2012): Define a judgment for paths.

\[
\Gamma \vdash M : A
\]

\[
\Gamma \vdash P : M \simeq N : A
\]

\[
\Gamma \vdash H : P \simeq Q : M \simeq N : A
\]

We can organize iterated path judgments cubically.
Path Judgments

*Canonicity for 2-Dimensional Type Theory*, Licata and Harper (POPL 2012): Define a judgment for paths.

\[ \Gamma \vdash M : A \]
\[ \Gamma \vdash P : M \simeq N : A \]
\[ \Gamma \vdash H : P \simeq Q : M \simeq N : A \]

We can organize iterated path judgments cubically.
Cubical Programs


Programs representing points, lines, squares, cubes...

$n$-dimensional programs parametrized by $n$ dimension variables.

- **base** is a point (no dimensions).
- **loop**$^x$ is a line (one dimension, $x$).
Cubical Programs

Imagine a square $M$ as a map $M(x, y): [0, 1]^2 \to \text{Term}$.

Substituting for a dimension computes an aspect.

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.
Imagine a square $M$ as a map $M(x, y) : [0, 1]^2 \rightarrow \text{Term}$.

Substituting for a dimension computes an aspect.

$M\langle 0/x \rangle$

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.
Cubical Programs

Imagine a square $M$ as a map $M(x, y) : [0, 1]^2 \rightarrow \text{Term}$. Substituting for a dimension computes an aspect.

$M\langle 0/x \rangle \quad M\langle 0/y \rangle$

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.
Cubical Programs

Imagine a square $M$ as a map $M(x, y) : [0, 1]^2 \rightarrow \text{Term}$.

Substituting for a dimension computes an aspect.

\[ M(0/x)(0/y) = M(0/y)(0/x) \]

Dimension substitutions compute aspects (faces, diagonals) of cubes. Substitution satisfies expected geometric laws.
Cubical Programs

Can evaluate programs of any dimension.

\[
\begin{align*}
\text{base val} & \quad \text{loop}_x \text{ val} \\
\text{loop}_0 \overset{\rightarrow}{\mapsto} \text{base} & \quad \text{loop}_1 \overset{\rightarrow}{\mapsto} \text{base}
\end{align*}
\]

expected

The bottom rules ensure that the faces of \( \text{loop}_x \) are both \( \text{base} \).
Cubical Judgments

Judgments at every dimension.

\[ M \text{ is a point} \quad \Gamma \gg M \in A [\emptyset] \]

\[ \ldots \text{line} \quad \Gamma \gg M \in A [x] \]

\[ \ldots \text{square} \quad \Gamma \gg M \in A [x, y] \]

\[ \ldots \text{cube} \quad \Gamma \gg M \in A [x, y, z] \]
The cubical judgments

\[ \Gamma \gg A \simeq B \quad \text{pretype } [\Psi] \]

\[ \Gamma \gg M \simeq N \in A \ [\Psi] \]

are defined by the cubical meaning explanations.
Closed Cubical Judgments

\[ A \text{ pretype } [\Psi] \]

means \[ A \Downarrow A_0 \] ,

and we specify the canonical \( \Psi \)-elements of \( A_0 \) , and

when two canonical \( \Psi \)-elements of \( A_0 \) are equal,

\[ \psi \text{ is an arbitrary dimension substitution from } \Psi \text{ to } \Psi'. \]
Closed Cubical Judgments

\[ A \quad \text{pretype} \quad \lbrack \Psi \rbrack \]

means \( \forall \psi : \Psi' \rightarrow \Psi, \ A\psi \downarrow A_0 \), and we specify the canonical \( \Psi' \)-elements of \( A_0 \), and when two canonical \( \Psi' \)-elements of \( A_0 \) are equal.

\( \psi \) is an arbitrary dimension substitution from \( \Psi \) to \( \Psi' \).
Closed Cubical Judgments

\[ A \doteq B \text{ pretype } [\Psi] \]

means \( \forall \psi : \Psi' \to \Psi, \ A\psi \downarrow A_0 \text{ and } B\psi \downarrow B_0, \)

and we specify the canonical \( \Psi' \)-elements of \( A_0 \) (resp., \( B_0 \)), and when two canonical \( \Psi' \)-elements of \( A_0 \) (resp., \( B_0 \)) are equal, and the canonical \( \Psi' \)-elements of \( A_0 \) and \( B_0 \) are the same, with the same equality.

\( \psi \) is an arbitrary dimension substitution from \( \Psi \) to \( \Psi' \).
Closed Cubical Judgments

\[ M \in A[\Psi] \]

means \( \forall \psi : \Psi' \to \Psi, \ M\psi \downarrow M_0 \),

and \( M_0 \) is a canonical \( \Psi' \)-element of \( A_0 \) (where \( A\psi \downarrow A_0 \)).

The highlighted condition only makes sense if we presuppose that \( A \) \textbf{pretype} [\( \Psi \)].
Closed Cubical Judgments

\[ M \in A[\Psi] \]

presupposing \textit{A pretype} \([\Psi]\),

means \(\forall \psi : \Psi' \to \Psi, \ M\psi \downarrow M_0\),

and \(M_0\) is a \textit{canonical} \(\Psi'\)-element of \(A_0\) (where \(A\psi \downarrow A_0\)).

\[ \text{The highlighted condition only makes sense if we presuppose that } A \text{ pretype } [\Psi]. \]
Closed Cubical Judgments

\[ M \Downarrow N \in A [\Psi] \]

presupposing \( A \) pretype \([\Psi]\),

means \( \forall \psi : \Psi' \rightarrow \Psi, \; M\psi \Downarrow M_0 \) and \( N\psi \Downarrow N_0 \),

and \( M_0 \) and \( N_0 \) is-a are equal canonical \( \Psi' \)-elements of \( A_0 \) (where \( A\psi \Downarrow A_0 \)).

The highlighted condition only makes sense if we presuppose that \( A \) pretype \([\Psi]\).
Open Cubical Judgments

\[ c : C \gg A \sqsupseteq B \text{ pretype } [\Psi] \]

when \( C \text{ pretype } [\Psi], \)
\[ \forall M \in C [\Psi], \]
\[ A [M/c] \doteq B [M/c] \text{ pretype } [\Psi]. \]

\[ c : C \gg N \sqsupseteq N' \in A [\Psi] \]

when \( C \text{ pretype } [\Psi], \)
\[ \forall M \in C [\Psi], \]
\[ N [M/c] \doteq N' [M/c] \in A [M/c] [\Psi]. \]

Open judgments mean that, for all equal elements of \( C', \) the corresponding closed judgments hold.
Open Cubical Judgments

\[ c : C \gg A \doteq B \text{ pretype } [\Psi] \]

when \( C \text{ pretype } [\Psi] \),
\[ \forall M \doteq M' \in C \ [\Psi], \]
\[ A \ [M/c] \doteq B \ [M'/c] \text{ pretype } [\Psi]. \]

\[ c : C \gg N \doteq N' \in A [\Psi] \]

when \( C \text{ pretype } [\Psi] \),
\[ \forall M \doteq M' \in C \ [\Psi], \]
\[ N \ [M/c] \doteq N' \ [M'/c] \in A \ [M/c] [\Psi]. \]

Open judgments mean that, for all equal elements of \( C \), the corresponding closed judgments hold.
Open Cubical Judgments

\[ c : C \gg A \doteq B \text{ pretype } [\Psi] \]

when \( C \) pretype \([\Psi]\),
\[ \forall \psi : \Psi' \rightarrow \Psi, \forall M \doteq M' \in C\psi \ [\Psi'], \]
\[ A\psi[M/c] \doteq B\psi[M'/c] \text{ pretype } [\Psi']. \]

\[ c : C \gg N \doteq N' \in A \ [\Psi] \]

when \( C \) pretype \([\Psi]\),
\[ \forall \psi : \Psi' \rightarrow \Psi, \forall M \doteq M' \in C\psi \ [\Psi'], \]
\[ N\psi[M/c] \doteq N'\psi[M'/c] \in A\psi[M/c] \ [\Psi']. \]

Open judgments mean that, for all equal elements of \( C' \), the corresponding closed judgments hold.
Definition
A partial equivalence relation is a symmetric and transitive relation.

 Canonical pretype equality: \( \approx_\Psi \) is a PER over \( \Psi \)-dim’l values.

 Canonical element equality: \( \approx_\Psi \) is a \( (\approx_\Psi) \)-indexed family of PERs over \( \Psi \)-dim’l values.
Cubical Type Systems

Definition
A cubical type system is a pair $(\approx^-, \approx_-)$.

\[
\begin{align*}
A \vdash B & \text{ pretype } [\Psi] \\
\forall \psi : \Psi' \to \Psi, \ A\psi \Downarrow A_0, B\psi \Downarrow B_0, \ A_0 \approx_{\Psi'} B_0
\end{align*}
\]

\[
\begin{align*}
M \vdash N \in A [\Psi] \\
\forall \psi : \Psi' \to \Psi, \ M\psi \Downarrow M_0, N\psi \Downarrow N_0, \ M_0 \approx_{A_0} N_0 \text{ where } A\psi \Downarrow A_0.
\end{align*}
\]

The judgments have meaning in any cubical type system.
Cubical Type Systems

We want a cubical type system with types!

A cubical type system has the (strict) booleans when:

1. \( \text{bool} \approx \Psi \text{bool} \)
2. \( M_0 \approx_{\text{bool}} N_0 \iff (M_0 = N_0 = \text{true} \lor M_0 = N_0 = \text{false}) \)

We place conditions on CTSes to ensure they have certain type formers.
Cubical Type Systems

Theorem

In every cubical type system with strict booleans,

\[ \Gamma \gg \text{bool pretype } [\Psi] \quad \Gamma \gg \text{true } \in \text{bool } [\Psi] \quad \ldots \]

Theorem (Canonicity)

If \( \cdot \gg M \in \text{bool } [\Psi] \) then \( M \downarrow \text{true or } M \downarrow \text{false} \).

Canonicity (which ensures proper PAT) here holds by definition; the hard part is proving the rules of type theory.
Coherence of Aspects

\[ M \langle 0/x \rangle \Downarrow V \quad M \langle 0/y \rangle \Downarrow V' \]

\[ V \langle 0/y \rangle \; ? = \; V' \langle 0/x \rangle \]

In the paper, we also have a coherence condition between evaluation and dimension substitution...
Kan Conditions

A type $[\Psi]$ when $A$ pretype $[\Psi]$ and satisfies Kan conditions.

Generalized coercion:

\[
M
\]

\[
\therefore
A^{\langle 0/x \rangle} \xrightarrow{\quad A \quad} A^{\langle 1/x \rangle}
\]

...and the Kan conditions, to ensure types have generalized coercion and box-filling.
Kan Conditions

A type $[\Psi]$ when $A$ pretype $[\Psi]$ and satisfies Kan conditions.

Generalized coercion:

$$ M \xrightarrow{\text{\ldots}} \text{coe}_{x:A}^{0\rightarrow 1}(M) $$

...and the Kan conditions, to ensure types have generalized coercion and box-filling.
A type \([Ψ]\) when a pretype \([Ψ]\) and satisfies Kan conditions.

Generalized coercion:

\[
M \xrightarrow{\text{coe}_{x, A}^0 (M)} \text{coe}_{x, A}^0 (1) (M)
\]

\[
\cap \quad \cap \quad \cap
\]

\[
A\langle 0/x \rangle \quad \xrightarrow{A} \quad A\langle 1/x \rangle
\]

...and the Kan conditions, to ensure types have generalized coercion and box-filling.
Kan Conditions

Box filling.
(Ensures symmetry, transitivity, associativity of transitivity...)

For any three sides of a square, the fourth exists; for any three or five sides of a cube, the sixth exists.
Kan Conditions

Box filling.
(Ensures symmetry, transitivity, associativity of transitivity...)

For any three sides of a square, the fourth exists; for any three or five sides of a cube, the sixth exists.
Kan Conditions

Proving transitivity:

\[ \begin{align*}
  & \quad M_1 \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ quad
So What?
Results

- A higher-dimensional type theory whose proofs run.
- Defined cubical logical relations / cubical meaning explanations / cubical realizability.
- First canonicity theorem for a higher-dimensional type theory!
  - Dependent functions, dependent pairs, identifications.
  - Some HITs (circle, weak booleans).
  - Univalence for exact isomorphisms. (New!)
  - Contains computational type theory.
Related Work

Instead of (cubical) meaning explanations, one could...

Define a logic \( \Gamma \vdash M : A \) by rules (\( M \) is a formal proof of \( A \)).

To recover computation, define proof reduction for \( \Gamma \vdash M : A \),

\[
\Gamma \vdash M \succ N : A
\]

where \( \Gamma \vdash N : A \).
Related Work

Cubical type theories in the logical tradition by

- Licata and Brunerie (2014).
- Cohen, Coquand, Huber, Mörtberg (2016).
  - Has univalence and universes.
  - Proof reduction is possible, satisfies canonicity (Huber, 2016).
Future Work

- Continue implementation in RedPRL (Sterling, et al.).
- Full univalence and universes?
- Other HITs?
Thanks!

cs.cmu.edu/~cangiuli