Computational Higher Type Theory

Robert Harper

Computer Science Department
Carnegie Mellon University

GDP70 Celebration
Edinburgh, UK

Happy Birthday Gordon!
Thanks

Joint work with Carlo Angiuli (CMU) and Todd Wilson (CSUF).

Thanks to Dan Licata for many discussions.

Supported by AFOSR MURI FA9550-15-1-0053.
Two Kinds of Type Theory

Two traditions in type theory, both embodied by Martin-Löf:

- **Formal**, or axiomatic, as in ITT and HoTT.
- **Computational**, or semantic, as in CMCP.

HoTT is a formal type theory with

- Univalence Axiom stating that equivalences are type identifications.
- Higher Inductive Types, supporting truncation, etc.
Two Kinds of Type Theory

Two traditions in type theory, both embodied by Martin-Löf:

- **Formal**, or **axiomatic**, as in ITT and HoTT.
- **Computational**, or **semantic**, as in CMCP.

HoTT is a formal type theory with

- **Univalence Axiom** stating that equivalences are type identifications.
- **Higher Inductive Types**, supporting **truncation**, etc.
Two kinds of type theory

Two traditions in type theory, both embodied by Martin-Löf:

- **Formal**, or axiomatic, as in ITT and HoTT.
- **Computational**, or semantic, as in CMCP.

HoTT is a formal type theory with

- **Univalence Axiom** stating that equivalences are type identifications.
- **Higher Inductive Types**, supporting truncation, etc.

What is the computational content of HoTT?
Meaning explanations define types and elements semantically:

- **Computational**: as programs with deterministic dynamics.
- **Mathematical**: using inchoate concepts of set and function.
Meaning explanations define types and elements semantically:

- **Computational**: as programs with deterministic dynamics.
- **Mathematical**: using inchoate concepts of set and function.

Computational meaning explanation: type theory as a **prog lang**.

- Types are **behavioral specifications**.
- Types and objects are **programs** that execute.
Meaning explanations define types and elements semantically:

- **Computational**: as programs with deterministic dynamics.
- **Mathematical**: using inchoate concepts of set and function.

Computational meaning explanation: type theory as a *prog lang*.

- Types are *behavioral specifications*.
- Types and objects are *programs* that execute.

Inverts conceptual order compared to formal type theory:

- Type theory as a theory of *truth*.
- Proof theory *accesses* the truth.
Start with computation on closed expressions (types and terms):

- Transition: $M \xrightarrow{\cdot} M'$, one step of execution.
- Termination: $M$ val is canonical/complete.
Start with *computation* on closed expressions (types and terms):
- **Transition:** $M \xrightarrow{} M'$, one step of execution.
- **Termination:** $M \text{ val}$ is canonical/complete.

Define *exact equality* of closed types and terms:
- **Type equality:** $A \equiv B \text{ type } [\Psi]$.
- **Term equality in a type:** $M \equiv N \in A [\Psi]$. 
Start with **computation** on closed expressions (types and terms):

- Transition: \( M \xrightarrow{} M' \), one step of execution.
- Termination: \( M \text{ val is canonical/complete} \).

Define **exact equality** of closed types and terms:

- Type equality: \( A \equiv B \) type \([\Psi]\).
- Term equality in a type: \( M \equiv N \in A \) \([\Psi]\).

Extend to open forms by **functionality** aka **extensionality**:

- Types: \( a_1:A_1, \ldots, a_n:A_n \Rightarrow A \equiv B \) type \([\Psi]\).
- Terms: \( a_1:A_1, \ldots, a_n:A_n \Rightarrow M \equiv N \in A \) \([\Psi]\).
Bezem, et al.: model of (a version of) HoTT valued in constructive cubical sets.

- Indirect account of computational content of HoTT.
- Simplicial account appears not to be constructive.
Bezem, et al.: model of (a version of) HoTT valued in constructive cubical sets.

- Indirect account of computational content of HoTT.
- Simplicial account appears not to be constructive.

Focus shifted to higher type theory.
Bezem, et al.: model of (a version of) HoTT valued in constructive cubical sets.
  • Indirect account of computational content of HoTT.
  • Simplicial account appears not to be constructive.

Focus shifted to higher type theory.
  • Judgmental treatment of identifications.
Bezem, et al.: model of (a version of) HoTT valued in constructive cubical sets.

- Indirect account of computational content of HoTT.
- Simplicial account appears not to be constructive.

Focus shifted to higher type theory.

- Judgmental treatment of identifications.
- Cubical structure seems most natural.
Bezem, et al.: model of (a version of) HoTT valued in constructive cubical sets.

- Indirect account of computational content of HoTT.
- Simplicial account appears not to be constructive.

Focus shifted to higher type theory.

- Judgmental treatment of identifications.
- Cubical structure seems most natural.

Formal cubical type theories are under active development (Coquand, et al.; this work.)
Syntax is organized cubically:

- **Points** correspond to ordinary terms and types.
- **Lines** represent **identifications** of elements and types.
Syntax is organized cubically:

- **Points** correspond to ordinary terms and types.
- **Lines** represent **identifications** of elements and types.

**Cartesian cubes** are specified by a **dimension context**, $\Psi$:
- Finite set of **dimension variables** $x, y, z, \ldots$. 
Syntax is organized cubically:

- **Points** correspond to ordinary terms and types.
- **Lines** represent **identifications** of elements and types.

**Cartesian cubes** are specified by a **dimension context**, $\Psi$:

- Finite set of **dimension variables** $x, y, z, \ldots$.

**Substitutions** $\psi : \Psi' \rightarrow \Psi$ send $x \in \Psi$ to $\psi(x) = 0/1/x' \in \Psi'$. 
Cubical Programming Language

Substitutions define the aspects of a cube $E$:

- **Faces**: $E\langle 0/x \rangle, E\langle 1/x \rangle$.
- **Diagonals**: $E\langle x', x'/x, y \rangle$.
- **Symmetries**: $E\langle y, x/x, y \rangle$.
- **Degeneracy**: silent/implicit.
Cubical Programming Language

Conventional functional programming constructs:

- Booleans, pairs, functions.
- Lazy dynamics (weak head reduction)

Unconventional functional programming constructs:

- Circle:
  \[ S^1, \text{base}, \text{loop} x, S^1 \text{-elim} a, A(M; M^b, x). M^l) \]

- Negation: not \( x \), a type line, and glueing, not \( x (M) \).

- Kan operations: coe, hcom.

The Kan operations are computational content of the Kan condition (cf, LB14, CCHM16).
Cubical Programming Language

Conventional functional programming constructs:

- Booleans, pairs, functions.
- Lazy dynamics (weak head reduction)

Unconventional functional programming constructs:

- **Circle**: $\mathbb{S}^1$, base, loop$_x$, $\mathbb{S}^1$-elim$_{a.A}(M; M_b, x. M_l)$.
- **Negation**: not$_x$, a type line, and glueing, notel$_x(M)$.
- **Kan operations**: coe, hcom.
Cubical Programming Language

Conventional functional programming constructs:

- Booleans, pairs, functions.
- Lazy dynamics (weak head reduction)

Unconventional functional programming constructs:

- **Circle**: $S^1$, base, $\text{loop}_x$, $S^1\text{-elim}_{a.\Delta}(M; M_b, x. M_l)$.
- **Negation**: $\text{not}_x$, a type line, and glueing, $\text{notel}_x(M)$.
- **Kan operations**: coe, hcom.

The Kan operations are computational content of the Kan condition (cf, LB14, CCHM16).
Kan Operations

**Coercion** along a type line: $\text{coe}_{x \cdot A}^{r \leadsto r'}(M)$.

- **Heterogeneous** along line $x \cdot A$.
- Evaluates $A$ to effect coercion from $A\langle r/x \rangle$ to $A\langle r'/x \rangle$.

**Composition**: $\text{hcom}_A^{\vec{r}_i}(r \leadsto r', M; y.N^\varepsilon_i)$. 
Kan Operations

Coercion along a type line: $\text{coe}_{x:A}^{r \rightsquigarrow r'}(M)$.

- Heterogeneous along line $x:A$.
- Evaluates $A$ to effect coercion from $A \langle r/x \rangle$ to $A \langle r'/x \rangle$.

Composition: $\text{hcom}_{A}^{\overrightarrow{r_{i}}}(r \rightsquigarrow r', M; \overrightarrow{y.N_{i}^{\varepsilon}})$.

- Homogeneous: within type, not line, $A$. 
Coercion along a type line: $\text{coe}_{x.A}^{r \rightsquigarrow r'}(M)$.

- Heterogeneous along line $x.A$.
- Evaluates $A$ to effect coercion from $A\langle r/x \rangle$ to $A\langle r'/x \rangle$.

Composition: $\text{hcom}_{A}^{\overrightarrow{r_i}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})$.

- Homogeneous: within type, not line, $A$.
- The start $r$ and end $r'$ dimensions.
Kan Operations

Coercion along a type line: $\text{coe}^{r \sim r'}_{x.A} (M)$.

- **Heterogeneous** along line $x.A$.
- Evaluates $A$ to effect coercion from $A\langle r/x \rangle$ to $A\langle r'/x \rangle$.

Composition: $hcom^{\overrightarrow{r_i}}_A (r \sim r', M; \overrightarrow{\mathcal{N}_i})$.

- **Homogeneous**: within type, not line, $A$.
- The start $r$ and end $r'$ dimensions.
- The cap $M$ is the starting cube.
Kan Operations

Coercion along a type line: $\text{coe}_{x.A}^r (M)$.
- Heterogeneous along line $x.A$.
- Evaluates $A$ to effect coercion from $A\langle r/x \rangle$ to $A\langle r'/x \rangle$.

Composition: $\text{hcom}_A^{\vec{r}_i} (r \sim r', M; \vec{y}.N_i^\xi)$.
- Homogeneous: within type, not line, $A$.
- The start $r$ and end $r'$ dimensions.
- The cap $M$ is the starting cube.
- The tubes $\vec{y}.N_i^\xi$ with extent $\vec{r}_i$ in dimension $\vec{y}_i$. 
Kan Operations

Coercion along a type line: $\text{coe}^{r \rightsquigarrow r'}_{x.A}(M)$.
- Heterogeneous along line $x.A$.
- Evaluates $A$ to effect coercion from $A\langle r/x \rangle$ to $A\langle r'/x \rangle$.

Composition: $\text{hcom}^{\overrightarrow{r_i}}_{A}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\xi})$.
- Homogeneous: within type, not line, $A$.
- The start $r$ and end $r'$ dimensions.
- The cap $M$ is the starting cube.
- The tubes $\overrightarrow{y.N_i^\xi}$ with extent $\overrightarrow{r_i}$ in dimension $\overrightarrow{y_i}$.
- Evaluates $A$ to define composite, which may or may not be the hcom itself.
Two-Dimensional Compositions

\[ x \quad \xrightarrow{y} \quad M \quad \xleftarrow{N^0} \quad hcom^x_A(0 \rightsquigarrow 0, M; y.N^0, y.N^1) \quad \xleftarrow{N^1} \quad y \]

\[ N^0 \langle 1/y \rangle \quad \xleftarrow{N^0} \quad M \quad \xrightarrow{M} \quad N^1 \langle 1/y \rangle \]
Two-Dimensional Compositions

\[
\begin{aligned}
&\text{\begin{tikzpicture}[baseline=(current  bounding  box)]
\node (a) at (0,0) {\mathcal{N}^0};
\node (b) at (0,-1) {\mathcal{N}^0 \langle 1/y \rangle};
\node (c) at (4,0) {\mathcal{N}^1};
\node (d) at (4,-1) {\mathcal{N}^1 \langle 1/y \rangle};
\draw[->] (a) to node [above] {$x$} (b);
\draw[->] (a) to node [left] {$y$} (c);
\draw[->] (b) to node [below] {$\mathcal{N}^0 \langle 1/y \rangle$} (d);
\draw[->] (c) to node [below] {$\mathcal{N}^1 \langle 1/y \rangle$} (d);
\end{tikzpicture}}
\end{aligned}
\]
Two-Dimensional Compositions

\[
\begin{align*}
N_0 & \langle 1/y \rangle \\
N_0^0 & \langle 1/y \rangle \\
N_1^1 & \langle 1/y \rangle \\
\end{align*}
\]

\[ hcom_A^x (0 \leadsto z, M; y.N^0, y.N^1) \]
Cubical Meaning Explanation

Explanation proceeds in stages:

- Define the **canonical** types and their elements at each dimension $\Psi$.
- Define **pre-types** to be cubical, ie with coherent aspects.
- Define **types** to be Kan pre-types.
Cubical Meaning Explanation

Explanation proceeds in stages:

- Define the **canonical** types and their elements at each dimension $\Psi$.
- Define **pre-types** to be cubical, ie with coherent aspects.
- Define **types** to be Kan pre-types.

The main **criteria** for a higher type system:

- All aspects of a type or element must be types or elements.
- Taking aspects must **commute** with evaluation.
- Equal types must have the same element equality.
- Equal types must be **equally Kan**.
A cubical type system consists of a family of per’s:
A cubical type system consists of a family of per’s:

- Canonical types: $A_0 \approx_\Psi B_0$.

- Canonical elements of a canonical type: $M_0 \approx_\Psi A_0 N_0$.

- Type equality: If $A_0 \approx_\Psi B_0$, then $\approx_\Psi A_0$ is $\approx_\Psi B_0$.

- Extend to general closed expressions by evaluation:
  - $A \sim_\Psi B$ iff $A \mapsto^{-\rightarrow} \Psi^* A_0$ and $B \mapsto^{-\rightarrow} \Psi^* B_0$ and $A_0 \approx_\Psi B_0$.
  - $M \sim_\Psi A N$ iff $M \mapsto^{-\rightarrow} \Psi^* M_0$, $N \mapsto^{-\rightarrow} \Psi^* N_0$, $A \mapsto^{-\rightarrow} \Psi^* A_0$, and $M_0 \approx_\Psi A_0 N_0$.
A cubical type system consists of a family of per’s:

- **Canonical types**: $A_0 \approx \Psi B_0$.
- **Canonical elements** of a canonical type: $M_0 \approx_{A_0} N_0$. 


Cubical Type Systems

A cubical type system consists of a family of per’s:

- **Canonical types**: $A_0 \approx^\psi B_0$.
- **Canonical elements** of a canonical type: $M_0 \approx^\psi_{A_0} N_0$.
- **Type equality**: If $A_0 \approx^\psi B_0$, then $\approx^\psi_{A_0}$ is $\approx^\psi_{B_0}$.
A **cubical type system** consists of a family of per’s:

- **Canonical types**: $A_0 \approx \Psi B_0$.
- **Canonical elements** of a canonical type: $M_0 \approx_{A_0} N_0$.
- **Type equality**: If $A_0 \approx \Psi B_0$, then $\approx_{A_0}$ is $\approx_{B_0}$.

Extend to **general** closed expressions by evaluation:
A cubical type system consists of a family of per's:

- Canonical types: \( A_0 \approx \psi B_0 \).
- Canonical elements of a canonical type: \( M_0 \approx_{A_0} N_0 \).
- Type equality: If \( A_0 \approx \psi B_0 \), then \( \approx_{A_0} \) is \( \approx_{B_0} \).

Extend to general closed expressions by evaluation:

- \( A \sim \psi B \) iff \( A \rightarrow^* A_0 \) and \( B \rightarrow^* B_0 \) and \( A_0 \approx \psi B_0 \).
A cubical type system consists of a family of per’s:

- **Canonical types**: \( A_0 \approx^\psi B_0 \).
- **Canonical elements** of a canonical type: \( M_0 \approx^A_0 N_0 \).
- **Type equality**: If \( A_0 \approx^\psi B_0 \), then \( \approx^A_0 \) is \( \approx^B_0 \).

Extend to general closed expressions by evaluation:

- \( A \sim^\psi B \) iff \( A \mapsto^* A_0 \) and \( B \mapsto^* B_0 \) and \( A_0 \approx^\psi B_0 \).
- \( M \sim^A_N \) iff \( M \mapsto^* M_0, N \mapsto^* N_0, A \mapsto^* A_0 \), and \( M_0 \approx^A_0 N_0 \).
Pre-types A pretype $\Psi$ must have coherent aspects:

Pre-types $A$ pretype $[\Psi]$ must have coherent aspects:
Pre-Types: Coherent Aspects

Pre-types $A$ pretype $[\Psi]$ must have coherent aspects:

- Let $\psi_1 : \Psi_1 \rightarrow \Psi$ and $\psi_2 : \Psi_2 \rightarrow \Psi_1$. Similarly for exact equality of types and of elements: substitute-then-evaluate is functorial.
Pre-types $A$ pretype $[\Psi]$ must have coherent aspects:

- Let $\psi_1 : \Psi_1 \to \Psi$ and $\psi_2 : \Psi_2 \to \Psi_1$.
- Let $A\psi_1 \mapsto^* A_1 \text{ val}$, and $A_1\psi_2 \mapsto^* A_2 \text{ val}$, and $A\psi_2\psi_1 \mapsto^* A_{12} \text{ val}$.
Pre-types \( A \) pretype \([\Psi]\) must have coherent aspects:

- Let \( \psi_1 : \Psi_1 \to \Psi \) and \( \psi_2 : \Psi_2 \to \Psi_1 \).
- Let \( A\psi_1 \mapsto^* A_1 \text{ val}, \) and \( A_1 \psi_2 \mapsto^* A_2 \text{ val}, \) and \( A\psi_2\psi_1 \mapsto^* A_{12} \text{ val} \).
- Require:

\[
\begin{array}{c}
A \xrightarrow{\psi_1} A_1 \\
\downarrow \psi_1 \psi_2 \\
A_{12} \approx_{\psi_2} A_2 \\
\downarrow \psi_2 
\end{array}
\]

Similarly for exact equality of types and of elements: substitute-then-evaluate is functorial.
Pre-Types: Coherent Aspects

Pre-types $A$ pretype $[\Psi]$ must have coherent aspects:

- Let $\psi_1 : \Psi_1 \to \Psi$ and $\psi_2 : \Psi_2 \to \Psi_1$.
- Let $A\psi_1 \mapsto^* A_1 \text{ val}$, and $A_1 \psi_2 \mapsto^* A_2 \text{ val}$, and $A\psi_2 \psi_1 \mapsto^* A_{12} \text{ val}$.
- Require:

\[
\begin{array}{c}
A & \xrightarrow{\psi_1} & A_1 \\
\updownarrow & \scriptstyle{\psi_1 \psi_2} & \mathrel{\updownarrow} \scriptstyle{\psi_2} \\
A_{12} & \approx_{\psi_2} & A_2
\end{array}
\]

Similarly for exact equality of types and of elements: substitute-then-evaluate is functorial.
A pretype $[\Psi]$ is cubical: its values have coherent aspects:

- If $\psi : \Psi' \rightarrow \Psi$ and $M \approx_{A\psi} N$, then $M \vdash N \in A\psi [\Psi']$. 

Pre-Types and Types
Pre-Types and Types

A pretype $[\Psi]$ is cubical: its values have coherent aspects:

- If $\psi : \Psi' \rightarrow \Psi$ and $M \approx_{A\psi} N$, then $M \equiv N \in A\psi [\Psi']$.

A type is a Kan pre-type:
A pretype $[\Psi]$ is cubical: its values have coherent aspects:

- If $\psi : \Psi' \to \Psi$ and $M \simeq_{A\psi} N$, then $M \triangleq N \in A_\psi [\Psi']$.

A type is a Kan pre-type:

- Supports coercion and composition.
A pretype $[\Psi]$ is cubical: its values have coherent aspects:

- If $\psi : \Psi' \to \Psi$ and $M \approx_{A\psi}^\Psi N$, then $M \underset{\psi}{=} N \in A\psi [\Psi']$.

A type is a Kan pre-type:

- Supports coercion and composition.
- Certain equational requirements are met.
Kan Conditions for Coercion

For any $\psi : (\Psi', x) \to \Psi$, if

$$M \in A_\psi\langle r/x \rangle [\Psi'],$$

then

$$\text{coe}_{x.\ A_\psi}(M) \in A_\psi\langle r'/x \rangle [\Psi'].$$


Kan Conditions for Coercion

For any $\psi : (\Psi', x) \to \Psi$, if

$$M \in A_\psi \langle r/x \rangle [\Psi'],$$

then

$$\text{coe}_{x. A_\psi}^r (M) \in A_\psi \langle r'/x \rangle [\Psi'].$$

For any $\psi : (\Psi', x) \to \Psi$, if

$$M \in A_\psi \langle r/x \rangle [\Psi'],$$

then

$$\text{coe}_{x. A_\psi}^{r \rightsquigarrow r'} (M) \simeq M \in A_\psi \langle r/x \rangle [\Psi'].$$
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A_{\psi} [\Psi']$,
Kan Conditions for Composition

For any \( \psi : \Psi' \to \Psi \), if

- \( M \in A_{\psi}[\Psi'] \),
- \( N_{i \varepsilon} \equiv N_{j \varepsilon'} \in A_{\psi}[\Psi', y | r_i = \varepsilon, r_j = \varepsilon'] \) (all \( i, j, \varepsilon, \) and \( \varepsilon' \))
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A\psi [\Psi']$,
- $N_i^{\varepsilon} \doteq N_j^{\varepsilon'} \in A\psi [\Psi', y \mid r_i = \varepsilon, r_j = \varepsilon']$ (all $i, j, \varepsilon$, and $\varepsilon'$)
- $N_i^{\varepsilon} \langle r/y \rangle \doteq M \in A\psi [\Psi' \mid r_i = \varepsilon]$ (all $i$ and $\varepsilon$)
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A\psi[\Psi']$,
- $N^\varepsilon_i \vdash N^\varepsilon_j \in A\psi[\Psi', y | r_i = \varepsilon, r_j = \varepsilon']$ (all $i, j, \varepsilon, \varepsilon'$)
- $N^\varepsilon_i \langle r/y \rangle \vdash M \in A\psi[\Psi' | r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A_\psi [\Psi']$,
- $N_i^\varepsilon = N_j^\varepsilon' \in A_\psi [\Psi', y | r_i = \varepsilon, r_j = \varepsilon']$ (all $i, j, \varepsilon,$ and $\varepsilon'$)
- $N_i^\varepsilon \langle r/y \rangle = M \in A_\psi [\Psi' | r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then

- $\text{hcom}_{\overrightarrow{A_\psi}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \in A_\psi [\Psi']$. 
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A_{\psi} [\Psi']$,
- $N_{j}^{\varepsilon} \equiv N_{j}^{\varepsilon'} \in A_{\psi} [\Psi', y | r_{i} = \varepsilon, r_{j} = \varepsilon']$ (all $i, j, \varepsilon$, and $\varepsilon'$)
- $N_{i}^{\varepsilon} \langle r / y \rangle \vDash M \in A_{\psi} [\Psi' | r_{i} = \varepsilon]$ (all $i$ and $\varepsilon$)

then

- $hcom_{A_{\psi}} (r \rightsquigarrow r', M; \overline{y.N_{i}^{\varepsilon}}) \in A_{\psi} [\Psi']$.
- $hcom_{A_{\psi}} (r \rightsquigarrow r, M; \overline{y.N_{i}^{\varepsilon}}) \vDash M \in A_{\psi} [\Psi']$. 
Kan Conditions for Composition

For any $\psi : \Psi' \to \Psi$, if

- $M \in A_{\psi} [\Psi']$,
- $N^\varepsilon_i \doteq N^\varepsilon_j \in A_{\psi} [\Psi', \langle y \mid r_i = \varepsilon, r_j = \varepsilon' \rangle]$ (all $i, j, \varepsilon, \varepsilon'$)
- $N^\varepsilon_i \langle r / y \rangle \doteq M \in A_{\psi} [\Psi' \mid r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then

- $hcom^{\overrightarrow{r_i}}_{A_{\psi}} (r \rightsquigarrow r', M; \overrightarrow{y.N^\varepsilon_i}) \in A_{\psi} [\Psi']$.
- $hcom^{\overrightarrow{r_i}}_{A_{\psi}} (r \rightsquigarrow r, M; \overrightarrow{y.N^\varepsilon_i}) \doteq M \in A_{\psi} [\Psi']$.
- $hcom^{\overrightarrow{r_i}}_{A_{\psi}} (r \rightsquigarrow r', M; \overrightarrow{y.N^\varepsilon_i}) \doteq N^\varepsilon_i \langle r' / y \rangle \in A_{\psi} [\Psi']$ if $r_i = \varepsilon$. Constraints limit applicable substitutions; conditions can be vacuous.
Kan Conditions for Composition

For any $\psi : \Psi' \rightarrow \Psi$, if

- $M \in A_\psi [\Psi']$,
- $N_i^\varepsilon \vdash N_j^\varepsilon' \in A_\psi [\Psi', y \mid r_i = \varepsilon, r_j = \varepsilon']$ (all $i, j, \varepsilon$, and $\varepsilon'$)
- $N_i^\varepsilon \langle r/y \rangle \vdash M \in A_\psi [\Psi' \mid r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then

- $\text{hcom}_{A_\psi}^{\vec{r}_i} (r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \in A_\psi [\Psi']$.
- $\text{hcom}_{A_\psi}^{\vec{r}_i} (r \rightsquigarrow r, M; \overrightarrow{y.N_i^\varepsilon}) \vdash M \in A_\psi [\Psi']$.
- $\text{hcom}_{A_\psi}^{\vec{r}_i} (r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \vdash N_i^\varepsilon \langle r'/y \rangle \in A_\psi [\Psi']$ if $r_i = \varepsilon$.

Constraints limit applicable substitutions; conditions can be vacuous.
The Booleans are defined as a higher inductive type.
The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
- Canonical hcom’s are values.
The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
- Canonical hcom’s are values.
- (Could also have strict version.)
Defining Booleans

The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
-Canonical hcom’s are values.
  - (Could also have strict version.)

The dynamics of the conditional accounts for

- true and false, as usual.
- Canonical hcom’s.
Defining Booleans

The Booleans are defined as a higher inductive type.
- Innocent of its status as a set.
- Canonical hcom’s are values.
- (Could also have strict version.)

The dynamics of the conditional accounts for
- true and false, as usual.
- Canonical hcom’s.

The following “rules” are theorems, not definitions.
Boolean Dynamics

\[
\begin{align*}
\text{bool val} & \quad \text{hcom}_{\text{bool}}^{\vec{r}_i}(r \rightsquigarrow r', M; y.N_i^\varepsilon) \mapsto N_i^\varepsilon\langle r'/y \rangle \\
\text{true val} & \quad \text{false val}
\end{align*}
\]
$M \mapsto M'$

if $a.A(M; T, F) \mapsto if_a.A(M'; T, F)$

if $a.A(true; T, F) \mapsto T$

if $a.A(false; T, F) \mapsto F$

$r \neq r'$

$H = hcom_{bool}^{x_1, \ldots, x_n}(r \rightsquigarrow z, M; y.\overrightarrow{N_i})$

if $a.A(hcom_{bool}^{x_1, \ldots, x_n}(r \rightsquigarrow r', M; y.\overrightarrow{N_i}); T, F)$

$\mapsto$

$com_{z.A[H/a]}^{x_1, \ldots, x_n}(r \rightsquigarrow r', if_a.A(M; T, F); y.if_a.A(N_i; T, F))$

$\mapsto$

$coe_{x.bool}^{r \rightsquigarrow r'}(M) \mapsto M$
A CTS has booleans if $\text{bool} \approx_{\Psi} \text{bool}$ and $\approx_{\text{bool}}$ is least s.t.
A CTS has booleans if $\text{bool} \simeq \psi \text{bool}$ and $\simeq_{\text{bool}}$ is least s.t.

- $\text{true} \simeq_{\text{bool}} \text{true}$,
A CTS has booleans if $\text{bool} \approx \Psi$ bool and $\approx_{\text{bool}}$ is least s.t.

- $\text{true} \approx_{\text{bool}} \text{true}$,
- $\text{false} \approx_{\text{bool}} \text{false}$, and

Generally, values of positive type include compositions in higher dimensions.
A CTS has booleans if bool $\approx^\psi$ bool and $\approx^\psi_{\text{bool}}$ is least s.t.

- true $\approx^\psi_{\text{bool}}$ true,
- false $\approx^\psi_{\text{bool}}$ false, and
- $\text{hcom}_{\text{bool}}^{\bar{x}_i}(r \rightsquigarrow r', M; \bar{y}.N_i^\varepsilon) \approx^\psi_{\text{bool}}^{\bar{x}_i} \text{hcom}_{\text{bool}}^{\bar{x}_i}(r \rightsquigarrow r', O; \bar{y}.P_i^\varepsilon)$

when
A CTS has booleans if bool $\approx_\Psi \text{bool}$ and $\approx_\text{bool}$ is least s.t.

- true $\approx_\text{bool} \text{true}$,
- false $\approx_\text{bool} \text{false}$, and
- $h\text{com}_{\text{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i}) \approx_\Psi \overrightarrow{x_i} h\text{com}_{\text{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P_i})$

when

- $r \neq r'$,
A CTS has booleans if \( \text{bool} \approx^\Psi \text{bool} \) and \( \approx^\Psi_{\text{bool}} \) is least s.t.

- \( \text{true} \approx^\Psi_{\text{bool}} \text{true} \),
- \( \text{false} \approx^\Psi_{\text{bool}} \text{false} \), and
- \( \text{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \leadsto r', M; \overrightarrow{y.N_i}) \approx^\Psi^{\overrightarrow{x_i}}_{\text{bool}} \text{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \leadsto r', O; \overrightarrow{y.P_i}) \) when
  - \( r \neq r' \),
  - \( M \Downarrow O \in \text{bool} \ [\Psi] \),
A CTS has booleans if $\text{bool} \approx_{\Psi} \text{bool}$ and $\approx_{\text{bool}}$ is least s.t.

- $\text{true} \approx_{\text{bool}} \text{true}$,
- $\text{false} \approx_{\text{bool}} \text{false}$, and
- $\text{hcom}_{\text{bool}}(r \rightsquigarrow r', M; y.N_i) \approx_{\Psi, x} \text{hcom}_{\text{bool}}(r \rightsquigarrow r', O; y.P_i)$ when
  - $r \neq r'$,
  - $M \Downarrow O \in \text{bool}[\Psi]$,
  - $N_i \Downarrow N'_j \in \text{bool}[\Psi, y | x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$. 

Generally, values of positive type include compositions in higher dimensions.
A CTS has booleans if \( \text{bool} \approx \Psi \text{bool} \) and \( \approx_{\text{bool}} \) is least s.t.

- \( \text{true} \approx_{\text{bool}} \text{true} \),
- \( \text{false} \approx_{\text{bool}} \text{false} \), and
- \( \text{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i}) \approx_{\text{bool}} \text{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P_i}) \)

when

- \( r \neq r' \),
- \( M \doteq O \in \text{bool} [\Psi] \),
- \( N_i^\varepsilon \doteq N_j^\varepsilon' \in \text{bool} [\Psi, y | x_i = \varepsilon, x_j = \varepsilon'] \) for all \( i, j, \varepsilon, \varepsilon' \),
- \( N_i^\varepsilon \doteq P_i^\varepsilon \in \text{bool} [\Psi, y | x_i = \varepsilon] \) for all \( i, \varepsilon \), and
A CTS has booleans if $\text{bool} \approx^\psi \text{bool}$ and $\approx^\psi_{\text{bool}}$ is least s.t.

- true $\approx^\psi_{\text{bool}}$ true,
- false $\approx^\psi_{\text{bool}}$ false, and
- $\text{hcom}_{\text{bool}}(r \rightsquigarrow r', M; y.N_i^\varepsilon) \approx^\psi_{\text{bool}}^x \text{hcom}_{\text{bool}}(r \rightsquigarrow r', O; y.P_i^\varepsilon)$ when
  - $r \neq r'$,
  - $M \vdash O \in \text{bool}[\Psi]$,
  - $N_i^\varepsilon \vdash N_j^\varepsilon' \in \text{bool}[\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$,
  - $N_i^\varepsilon \vdash P_i^\varepsilon \in \text{bool}[\Psi, y \mid x_i = \varepsilon]$ for all $i, \varepsilon$, and
  - $N_i^\varepsilon(r/y) \vdash M \in \text{bool}[\Psi \mid x_i = \varepsilon]$ for all $i, \varepsilon$.

Generally, values of positive type include compositions in higher dimensions.
Canonical Booleans

A CTS has booleans if \( \text{bool} \approx_{\Psi} \) \text{bool} and \( \approx_{\text{bool}} \) is least s.t.

- \( \text{true} \approx_{\text{bool}} \text{true} \),
- \( \text{false} \approx_{\text{bool}} \text{false} \), and
- \( \text{hcom}_{\text{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i}) \approx_{\Psi,\chi} \text{hcom}_{\text{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P_i}) \) when
  - \( r \neq r' \),
  - \( M \vdash O \in \text{bool}[\Psi] \),
  - \( N_i^{\varepsilon} \vdash N_j^{\varepsilon'} \in \text{bool}[\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon'] \) for all \( i, j, \varepsilon, \varepsilon' \),
  - \( N_i^{\varepsilon} \vdash P_i^{\varepsilon} \in \text{bool}[\Psi, y \mid x_i = \varepsilon] \) for all \( i, \varepsilon \), and
  - \( N_i^{\varepsilon}\langle r/y \rangle \vdash M \in \text{bool}[\Psi \mid x_i = \varepsilon] \) for all \( i, \varepsilon \).

Generally, values of positive type include compositions in higher dimensions.
Not as a Line of Types

Define $\text{not}_x$ as a **line of types** between bool and bool.
Not as a Line of Types

Define $\text{not}_x$ as a line of types between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.
Not as a Line of Types

Define $\text{not}_x$ as a line of types between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.
- Example of univalence principle.
**Not as a Line of Types**

Define $\text{not}_x$ as a line of types between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.
- Example of univalence principle.

The term $\text{notel}_x(M) \in \text{not}_x [\Psi, x]$ is a line in $\text{not}_x$:

\[
\begin{array}{c}
\xymatrix{
\text{bool} & \text{not}_x \\
\text{bool} & x \\
\text{not}(-) & y \\
& \text{bool}
}
\end{array}
\]

\[
\begin{array}{c}
\xymatrix{
\text{not}_x & \text{bool} \\
& \text{not}_x
}
\end{array}
\]

Cf. CCHM gluing of equivalences to a line of types.
Not as a Line of Types

Define $\text{not}_x$ as a line of types between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.
- Example of univalence principle.

The term $\text{notel}_x(M) \in \text{not}_x [\Psi, x]$ is a line in $\text{not}_x$:

\[
\begin{array}{c}
\begin{array}{c}
x \\
y \downarrow
\end{array}
\end{array}
\xrightarrow{\text{not}(-)}
\begin{array}{c}
\begin{array}{c}
\text{bool} \\
\text{id}
\end{array}
\end{array}
\xrightarrow{\text{id}}
\begin{array}{c}
\begin{array}{c}
\text{bool} \\
\text{not}_x
\end{array}
\end{array}
\xrightarrow{\text{id}}
\]
Not as a Line of Types

Define $\text{not}_x$ as a **line of types** between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.
- Example of univalence principle.

The term $\text{notel}_x(M) \in \text{not}_x [\Psi, x]$ is a line in $\text{not}_x$:

Cf. CCHM gluing of equivalences to a line of types.
Computational Higher Type Theory

Canonicity Theorem: closed points of bool evaluate to true or false.

Validates higher-dimensional type theory from a computational, even Brouwerian, perspective.
Canonicity Theorem: closed points of bool evaluate to true or false.

Validates higher-dimensional type theory from a computational, even Brouwerian, perspective.

Huber has since proved canonicity for the CCHM theory, relative to weak head reduction.
Whither Proof Theory?

Validates expected formal rules where applicable.

- **NuPRL** rules for given constructs are valid.
- **LB14** rules for Kan cubical type theories are valid.
Whither Proof Theory?

Validates expected formal rules where applicable.

- NuPRL rules for given constructs are valid.
- LB14 rules for Kan cubical type theories are valid.

May be seen as abstract cubical extensional realizability interpretation.

- Abstract = open-ended: Church’s Law not accepted.
- Extensional = full universal properties for usual type constructors.
- Realizability for certain formal theories, but that’s not the point.
Ongoing and Future Work

Full account of univalence.
Ongoing and Future Work

Full account of univalence.

- Glueing for composition of types not applicable (diagonals).
- Canonical composites define a line of (Kan) types.
- Line of types $\text{ua}_x(E)$ for each equivalence $E$.
- Ensures equivalence between identifications and equivalences.

Implementation in Sterling's RedPRL (redprl.org).

NuPRL-like refinement rules.

Extended with names for cubical programs.
Ongoing and Future Work

Full account of univalence.

- Glueing for composition of types not applicable (diagonals).
- Canonical composites define a line of (Kan) types.
- Line of types $\text{ua}_x(E)$ for each equivalence $E$.
- Ensures equivalence between identifications and equivalences.

Implementation in Sterling’s RedPRL (redprl.org).

- NuPRL-like refinement rules.
- Extended with names for cubical programs.
Stuart F Allen, Mark Bickford, Robert L Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran.
Innovations in computational type theory using Nuprl.

Carlo Angiuli and Robert Harper.
Computational higher type theory II: Dependent cubical realizability.
Preprint, June 2016.

Computational higher type theory I: Abstract cubical realizability.
Preprint, April 2016.

Marc Bezem, Thierry Coquand, and Simon Huber.
A model of type theory in cubical sets.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg.
Cubical type theory: a constructive interpretation of the univalence axiom.
(To appear), January 2016.

Daniel R. Licata and Guillaume Brunerie.
A cubical type theory, November 2014.
Talk at Oxford Homotopy Type Theory Workshop.