Computational Higher Type Theory

Robert Harper

Computer Science Department
Carnegie Mellon University

HoTT Workshop 2016
Leeds, UK
Thanks

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

Thanks to Dan Licata for many conversations.

Thanks to HoTT Organizers for the invitation!

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.

Univalence Axiom, subsuming Function Extensionality.

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.

Most work in HoTT has taken place in the formal setting.

- **Univalence Axiom**, subsuming Function Extensionality.
- **Higher Inductive Types**, supporting truncation, etc.
Formal type theory is inductively defined by rules:

- **Formation**: $\Gamma \vdash A \text{ type}, \Gamma \vdash M : A$.
- **Definitional equivalence**: $\Gamma \vdash A \equiv B, \Gamma \vdash M \equiv N : A$. 

Axioms and rules are chosen to ensure:

- Not non-constructive, e.g., no unrestricted LEM.
- Formal correspondence to logics, e.g., HA, IHOL.
- Decidability of all assertions.

Choice of rules can be delicate, e.g., what is definitional equivalence?
Formal type theory is inductively defined by rules:

- **Formation**: $\Gamma \vdash A$ type, $\Gamma \vdash M : A$.
- **Definitional equivalence**: $\Gamma \vdash A \equiv B$, $\Gamma \vdash M \equiv N : A$.

Axioms and rules are chosen to ensure:

- **Not non-constructive**, eg no unrestricted LEM.
- **Formal correspondence to logics**, eg HA, IHOL.
- **Decidability** of all assertions.
Formal type theory is inductively defined by rules:

- **Formation**: $\Gamma \vdash A \text{ type}, \Gamma \vdash M : A$.
- **Definitional equivalence**: $\Gamma \vdash A \equiv B, \Gamma \vdash M \equiv N : A$.

Axioms and rules are chosen to ensure:

- **Not non-constructive**, eg no unrestricted LEM.
- **Formal correspondence to logics**, eg HA, IHOL.
- **Decidability** of all assertions.

Choice of rules can be delicate, eg what is definitional equivalence?
Formal Type Theory

Emphasis is on formal proof.

- $\Gamma \vdash M : A$ encodes proof checking.
- Tactics and decision procedures find proofs.
Formal Type Theory

Emphasis is on **formal proof**.

- \( \Gamma \vdash M : A \) encodes proof **checking**.
- Tactics and decision procedures **find** proofs.

Inductive definition yields a mapping out property:

- Assign **meaning** to types and terms.
- Associate **invariants** with types, eg normalization.
Emphasis is on formal proof.

- $\Gamma \vdash M : A$ encodes proof checking.
- Tactics and decision procedures find proofs.

Inductive definition yields a mapping out property:

- Assign meaning to types and terms.
- Associate invariants with types, eg normalization.

Adding axioms disrupts these properties!
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.
Computational Meaning Explanation


Start with \textit{computation} on closed expressions (types and terms):

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

- Transition: $M \rightarrow M'$, 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: $M \equiv N \in A [\Psi]$. 
Computational Meaning Explanation


Start with computation on closed expressions (types and terms):

- Transition: $M \rightsquigarrow M'$, 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: $a_1:A_1, \ldots, a_n:A_n \Rightarrow M \equiv N \in A [\Psi]$. 
Computational Meaning Explanation

Judgments are not intended to be decidable.

- Quantifier complexity is arbitrarily high, not merely r.e.
- Specifies execution behavior, not syntactic formation.
Computational Meaning Explanation

Judgments are not intended to be decidable.

- Quantifier complexity is arbitrarily high, not merely r.e.
- Specifies execution behavior, not syntactic formation.

Two essential moves for higher-dimensionality:

- Judgmental account of **identifications**.
- **Exact equality** of types and elements at all dimensions.
Syntax is organized cubically:

- **Points** correspond to ordinary terms and types.
- **Lines** represent identifications.
- **Squares** represent homotopies, etc.
Syntax is organized \textit{cubically}:

- \textbf{Points} correspond to ordinary terms and types.
- \textbf{Lines} represent identifications.
- \textbf{Squares} represent homotopies, etc.

\textbf{Cartesian cubes} are specified by a \textit{dimension context}, \( \Psi \):

- Finite set of \textit{dimension variables} \( x, y, z, \ldots \).
Syntax is organized **cubically**:
- **Points** correspond to ordinary terms and types.
- **Lines** represent identifications.
- **Squares** represent homotopies, etc.

**Cartesian cubes** are specified by a **dimension context**, **Ψ**:
- Finite set of **dimension variables** \(x, y, z, \ldots\)

**Substitutions** \(\psi : \Psi' \to \Psi\) send \(x \in \Psi\) to \(\psi(x) = 0/1/x' \in \Psi'\).
Substitutions define the **aspects** of a cube $E$:

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

Conventional functional programming constructs:

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

Conventional functional programming constructs:
- Booleans, pairs, functions.
- Lazy dynamics (weak head reduction)

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

Conventional functional programming constructs:

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

Unconventional functional programming constructs:

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

The Kan operations are computational content of the Kan condition (cf, LB14, CCHM16).
Coercion along a type line: $\text{coe}_{x.A}^{r \leadsto r'}(M)$.

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

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

- Homogeneous: within type, not line, \( A \).
Kan Operations

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

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

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

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

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

Composition: $\text{hcom}_A^{\vec{r}_i} (r \rightsquigarrow r', M; \vec{N}_i)$.
- 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: \( \text{coe}^{r \Rightarrow r'}_{x.A}(M) \).
- **Heterogeneous** along line \( x.A \).
- Evaluates \( A \) to effect coercion from \( A(r/x) \) to \( A(r'/x) \).

**Composition**: \( \text{hcom}^{r_i}_{A} (r \Rightarrow r', M; y.N_{i}^{\xi}) \).
- **Homogeneous**: within type, not line, \( A \).
- The **start** \( r \) and **end** \( r' \) dimensions.
- The **cap** \( M \) is the starting cube.
- The **tubes** \( y.N_{i}^{\xi} \) with extent \( r_{i} \) in dimension \( y_{i} \).
Kan Operations

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

**Composition**: $\text{hcom}_{A}^{\overrightarrow{r_i}} (r \sim r', M; \overrightarrow{y_i})$.
- **Homogeneous**: within type, not line, $A$.
- The start $r$ and end $r'$ dimensions.
- The cap $M$ is the starting cube.
- The tubes $\overrightarrow{y_i}$ 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

\[
N_0 \langle 1/y \rangle \xrightarrow{\text{hcom}_A^x(0 \rightsquigarrow 0, M; y.N^0, y.N^1)} N_1 \langle 1/y \rangle
\]
Two-Dimensional Compositions

\[ y \xrightarrow{x} N_0 \langle \text{1/y} \rangle \]

\[ \text{hcom}^x_A(0 \rhd \text{1}, M; y.N^0, y.N^1) \]

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

\[
\text{hcom}_A^x(0 \rightsquigarrow z, M; y.N^0, y.N^1)
\]
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**: $A_0 \approx^\psi B_0$. 

---

Cubical Type Systems

---

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

- **Canonical types**: $A_0 \approx^\psi B_0$. 

---

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

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

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

Extend to general closed expressions by evaluation:
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 $\approx^\psi_{B_0}$.

Extend to general closed expressions by evaluation:

- $A \sim^\psi B$ iff $A \stackrel{*}{\longrightarrow} A_0$ and $B \stackrel{*}{\longrightarrow} B_0$ 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 $\approx_{A_0} \approx_{B_0}$.

Extend to general closed expressions by evaluation:

- $A \sim \Psi B$ iff $A \rightarrow^* A_0$ and $B \rightarrow^* B_0$ and $A_0 \approx \Psi B_0$.
- $M \sim A N$ iff $M \rightarrow^* M_0$, $N \rightarrow^* N_0$, $A \rightarrow^* A_0$, and $M_0 \approx_{A_0} N_0$. 
Pre-Types: Coherent Aspects

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

- Let $\psi_1 : \Psi \rightarrow \Psi$ and $\psi_2 : \Psi \rightarrow \Psi_1$.
- Let $A \psi_1 \mapsto \rightarrow \Psi_1 \text{val}$, and $A \psi_2 \mapsto \rightarrow \Psi_2 \text{val}$, and $A \psi_2 \psi_1 \mapsto \rightarrow \Psi_{12} \text{val}$.
- Require: $A \Psi_1 \approx \Psi_2 \Psi_1 \approx \Psi_2$.

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 $\psi_2 : \Psi_2 \to \Psi_1$. 

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 $A\psi_1 \mapsto^* A_1$ val, and $A_1 \psi_2 \mapsto^* A_2$ val, and $A\psi_2 \psi_1 \mapsto^* A_{12}$ val.
- Require:

$$
\begin{array}{c}
  A \xrightarrow{\psi_1} A_1 \\
  \downarrow \psi_1 \psi_2 \quad \downarrow \psi_2 \\
  A_{12} \approx^{\psi_2} A_2 
\end{array}
$$
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 $A\psi_1 \to^* A_1$ val, and $A_1\psi_2 \to^* A_2$ val, and $A\psi_2\psi_1 \to^* A_{12}$ val.
- Require:

$$
A \xrightarrow{\psi_1} A_1 \\
\xrightarrow{\psi_1 \psi_2} A_{12} \approx_{\Psi_2} A_2
$$

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

Pre-Types and Types
Pre-Types and Types

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

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

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

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

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 $M \approx^{\Psi'}_{A_{\psi}} N$, then $M \equiv N \in A_{\psi} [\Psi']$.

A type is a Kan pre-type:

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

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

$$M \in A_{\psi'}(r/x)[\Psi'],$$

then

$$\text{coe}_{x.A_{\psi}}(M) \in A_{\psi}(r'/x)[\Psi'].$$
Kan Conditions for Coercion

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

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

then

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

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

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

then

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

Kan Conditions for Composition

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

- $M \in A_\psi[\Psi']$, constraints limit applicable substitutions; conditions can be vacuous.
Kan Conditions for Composition

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

- $M \in A_\psi [\Psi']$,
- $N_i^\epsilon \triangleright N_j^\epsilon' \in A_\psi [\Psi', y | r_i = \epsilon, r_j = \epsilon']$ (all $i$, $j$, $\epsilon$, and $\epsilon'$)
Kan Conditions for Composition

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

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

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

- $M \in A\psi [\Psi']$,
- $N_{i}^{\varepsilon} \vdash N_{j}^{\varepsilon'} \in A\psi [\Psi', y \mid r_i = \varepsilon, r_j = \varepsilon']$ (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
Kan Conditions for Composition

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

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

then

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

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

- $M \in A\psi[\Psi']$,
- $N_{i}^{\varepsilon} \doteq N_{j}^{\varepsilon'} \in A\psi[\Psi', y | r_i = \varepsilon, r_j = \varepsilon']$ (all $i, j, \varepsilon$, and $\varepsilon'$)
- $N_{i}^{\varepsilon}\langle r/y \rangle \doteq M \in A\psi[\Psi' | r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then

- $\text{hcom}_{\overleftarrow{A\psi}}^{r_i}(r \rightsquigarrow r', M; y.\overrightarrow{N_{i}^{\varepsilon}}) \in A\psi[\Psi']$.
- $\text{hcom}_{\overleftarrow{A\psi}}^{r_i}(r \rightsquigarrow r, M; y.\overrightarrow{N_{i}^{\varepsilon}}) \doteq M \in A\psi[\Psi']$. 

Constraints limit applicable substitutions; conditions can be vacuous.
Kan Conditions for Composition

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

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

then

1. $hcom_{A_\psi} \overset{\sim}{\overset{\rightarrow}{r_i}} (r \rightsquigarrow r', M; y.N_i^{\varepsilon}) \in A_\psi [\Psi']$.
2. $hcom_{A_\psi} \overset{\sim}{\overset{\rightarrow}{r_i}} (r \rightsquigarrow r, M; y.N_i^{\varepsilon}) \vdash M \in A_\psi [\Psi']$.
3. $hcom_{A_\psi} \overset{\sim}{\overset{\rightarrow}{r_i}} (r \rightsquigarrow r', M; y.N_i^{\varepsilon}) \vdash N_i^\varepsilon \langle r'/y \rangle \in A_\psi [\Psi']$ if $r_i = \varepsilon$. Constraints limit applicable substitutions; conditions can be vacuous.
Kan Conditions for Composition

For any $\psi : \Psi' \to \Psi$, 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 $\varepsilon'$)
- $N_i^\varepsilon \langle r/y \rangle \vdash M \in A\psi [\Psi' | r_i = \varepsilon]$ (all $i$ and $\varepsilon$)

then

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

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

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.
- Certain hcom’s are values.
The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
- Certain hcom’s are values.
- Could also define a strict variant.
The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
- Certain \texttt{hcom}’s are values.
- Could also define a \texttt{strict} variant.

The dynamics of the conditional accounts for

- \texttt{true} and \texttt{false}, as usual.
- \texttt{hcom}’s that are values.
Boolean Dynamics

\[ \overrightarrow{r_i} = x_1, \ldots, x_{i-1}, \varepsilon, r_{i+1}, \ldots, r_n \]

\[ \text{bool val} \]

\[ \text{hcom}_{\text{bool}}^r_\varepsilon (r \leadsto r', M; y.N_i^\varepsilon) \mapsto N_i^\varepsilon(r'/y) \]

\[ r = r' \]

\[ \text{hcom}_{\text{bool}}^{x_1, \ldots, x_n} (r \leadsto r', M; y.N_i^\varepsilon) \mapsto M \]

true val

false val

\[ r \neq r' \]

\[ \text{hcom}_{\text{bool}}^{x_1, \ldots, x_n} (r \leadsto r', M; y.N_i^\varepsilon) \text{ val} \]
Boolean Dynamics

\[
\begin{align*}
M & \mapsto M' \\
\text{if}_a.A(M; T, F) & \mapsto \text{if}_a.A(M'; T, F) \\
\text{if}_a.A(\text{true}; T, F) & \mapsto T \\
\text{if}_a.A(\text{false}; T, F) & \mapsto F \\
\end{align*}
\]

\[
\begin{align*}
{r \neq r'} & \\
H = \text{hcom}^{x_1, \ldots, x_n}_{\text{bool}}(r \mapsto z, M; \underbrace{y.N_{i}}_{r}) & \\
\text{if}_a.A(\text{hcom}^{x_1, \ldots, x_n}_{\text{bool}}(r \mapsto r', M; y.N_{i}); T, F) & \mapsto \\
\text{com}^{x_1, \ldots, x_n}_{z.A[H/a]}(r \mapsto r', \text{if}_a.A(M; T, F); y.\text{if}_a.A(N_{i}; T, F)) & \\
\text{coe}^{r \mapsto r'}_{x.\text{bool}}(M) & \mapsto M
\end{align*}
\]
A CTS has booleans if $\text{bool} \simeq \Psi \text{bool}$ and $\simeq_{\text{bool}}$ is least s.t.
A CTS has booleans if \( \text{bool} \approx \Psi \) bool and \( \approx_{\text{bool}} \) is least s.t.

- \( \text{true} \approx_{\text{bool}} \text{true} \),
A CTS has booleans if $\text{bool} \cong \Psi \text{bool}$ and $\cong_{\text{bool}}$ is least s.t.

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

Guarantees canonicity for closed points in bool: all evaluate to either true or false.
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
- $\text{hcom}_{\text{bool}}^\vec{x_i}(r \rightsquigarrow r', M; y.N_i) \approx_\Psi^\vec{x} \text{hcom}_{\text{bool}}^\vec{x_i}(r \rightsquigarrow r', O; y.P_i^\vec{\varepsilon})$

when
Canonical Booleans

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

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

Guarantees canonicity for closed points in bool: all evaluate to either true or false.
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
- \( \text{hcom}^\chi_{\text{bool}}(r \rightsquigarrow r', M; y . N_i^\varepsilon) \approx^\Psi^\chi_{\text{bool}} \text{hcom}^\chi_{\text{bool}}(r \rightsquigarrow r', O; y . P_i^\varepsilon) \)

when

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

- $\text{true} \approx^{\Psi} \text{bool} \text{true}$,
- $\text{false} \approx^{\Psi} \text{bool} \text{false}$, and
- $\text{hcom}^{\xrightarrow{\text{bool}}} (r \rightsquigarrow r', M; y.N_i) \approx^{\Psi, \text{bool}} \text{hcom}^{\xrightarrow{\text{bool}}} (r \rightsquigarrow r', O; y.P_i)$

when

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

 Guarantees canonicity for closed points in bool: all evaluate to either true or false.
Define $\text{not}_x$ as a type line between bool and bool.
Not as a Type Line

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

- Given by negation (swapping) as a (strict) equivalence.
Define $\text{not}_x$ as a **type line** between bool and bool.

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

Define $\text{not}_x$ as a type line 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 use of gluing [CCHM16]:

$$
\begin{array}{c}
\text{notel}_x(M) \\
\downarrow \\
\Psi
\end{array}
\xrightarrow{\text{not}_x}
\begin{array}{c}
\text{bool} \\
\downarrow \\
\text{bool}
\end{array}
\xrightarrow{\text{id}}
\begin{array}{c}
\text{bool} \\
\downarrow \\
\text{not}_x
\end{array}
\xrightarrow{\text{id}}
\begin{array}{c}
\text{bool} \\
\downarrow \\
\text{not}_x
\end{array}
\xrightarrow{\text{id}}
\begin{array}{c}
\text{bool}
\end{array}
$$
Identification type $\text{Id}_{X,A}(M, N)$ is dimension shift.
Identification type $\text{Id}_{x_A}(M, N)$ is dimension shift.

- Same as LB14 and CCHM16, but not HoTT.
Other Types Considered

Identification type $\text{Id}_{x.A}(M, N)$ is *dimension shift*.

- Same as LB14 and CCHM16, but not HoTT.
- Requires multiple tubes in hcom.
Other Types Considered

Identification type $\text{Id}_{x.\mathcal{A}}(M, N)$ is dimension shift.

- Same as LB14 and CCHM16, but not HoTT.
- Requires multiple tubes in hcom.
- Should be possible to define based path type, etc.
Other Types Considered

Identification type $\text{Id}_{x.A}(M, N)$ is dimension shift.

- Same as LB14 and CCHM16, but not HoTT.
- Requires multiple tubes in hcom.
- Should be possible to define based path type, etc.

The circle $\mathbb{S}^1$ is straightforward (no worse than bool).
Other Types Considered

Identification type $\text{Id}_x.A(M, N)$ is dimension shift.

- Same as LB14 and CCHM16, but not HoTT.
- Requires multiple tubes in hcom.
- Should be possible to define based path type, etc.

The circle $\mathbb{S}^1$ is straightforward (no worse than bool).

Dependent function and product types (Pi’s and Sigma’s) with full universal properties.
Whither Proof Theory?

Validates expected formal rules.

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

Validates expected formal rules.
- **NuPRL** rules for given constructs are valid.
- **LB14** rules for Kan cubical type theories are valid.

May be seen as **cubical extensional realizability** interpretation.
- Elicits **computational content** of proofs.
- Entails **canonicity**: Boolean points evaluate to true or false.
- Cubical **intensional** realizability via open terms?
Whither Proof Theory?

Validates expected formal rules.

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

May be seen as cubical extensional realizability interpretation.

- Elicits computational content of proofs.
- Entails canonicity: Boolean points evaluate to true or false.
- Cubical intensional realizability via open terms?

But why limit attention to these formal theories?
There is more to type theory than just known formal logics.

- **Richer notions of computation**: partiality, non-determinism, recursive types, exceptions, state, ..... [Constable, et al.]
There is more to type theory than just known formal logics.

- **Richer notions of computation**: partiality, non-determinism, recursive types, exceptions, state, .... [Constable, et al.]
- Internalize **exact equality** by handling pre-types as well as types, a la VV’s HTS.
Whither Proof Theory?

There is more to type theory than just known formal logics.

- **Richer notions of computation**: partiality, non-determinism, recursive types, exceptions, state, … [Constable, et al.]
- Internalize **exact equality** by handling pre-types as well as types, a la VV’s HTS.

Computational higher type theory as a programming language?
Whither Proof Theory?

There is more to type theory than just known formal logics.

- Richer notions of computation: partiality, non-determinism, recursive types, exceptions, state, …. [Constable, et al.]
- Internalize exact equality by handling pre-types as well as types, a la VV’s HTS.

Computational higher type theory as a programming language?

- Agda syntax and checking, but with a dynamics.
There is more to type theory than just known formal logics.

- **Richer notions of computation**: partiality, non-determinism, recursive types, exceptions, state, . . . . [Constable, et al.]
- **Internalize exact equality** by handling pre-types as well as types, a la VV’s HTS.

Computational higher type theory as a programming language?

- **Agda** syntax and checking, but with a dynamics.
- **Idris** for verified programming.
Whither Proof Theory?

There is more to type theory than just known formal logics.

- **Richer notions of computation**: partiality, non-determinism, recursive types, exceptions, state, ..... [Constable, et al.]
- **Internalize exact equality** by handling pre-types as well as types, a la VV’s HTS.

Computational higher type theory as a programming language?

- **Agda** syntax and checking, but with a dynamics.
- **Idris** for verified programming.

Computation model **induces** dynamics of explicitly typed languages.
Ongoing and Future Work

Full account of univalence for all types.
Full account of **univalence** for all types.

- Not tied to a **universe** (which are only for size issues).
- Currently exploring glueing [CCHM].
- Are cartesian cubes workable? (So far, so good.)
Ongoing and Future Work

Full account of **univalence** for all types.
- Not tied to a **universe** (which are only for size issues).
- Currently exploring glueing [CCHM].
- Are cartesian cubes workable? (So far, so good.)

Implementation in Sterling’s **RedPRL** (redprl.org).
- NuPRL-like refinement rules.
- Richer notion of tactics.
- Name generation is primitive (cf continuity principle).
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.