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.

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

Martin-Löf; Coquand; HoTT

Formal type theory is inductively defined by rules:

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

Formal Type Theory

Martin-Löf; Coquand; HoTT

Formal type theory is inductively defined by rules:

- Formation: $\Gamma \vdash A\,\mathsf{type}$, $\Gamma \vdash M : A$.
- 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

Martin-Löf; Coquand; HoTT

Formal type theory is inductively defined by rules:

- Formation: $\Gamma \vdash A\,\mathsf{type}$, $\Gamma \vdash M : A$.
- 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.

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.

Adding axioms disrupts these properties!

Semantic Type Theory

Martin-Löf; Constable, et al

Meaning explanations define types and elements semantically:

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

Semantic Type Theory

Martin-Löf; Constable, et al

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.

Semantic Type Theory

Martin-Löf; Constable, et al

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

Martin-Löf: Constr. Math. and Comp. Prog.

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

- Transition: $M \longmapsto M'$, one step of execution.
- Termination: M val is canonical/complete.

Computational Meaning Explanation

Martin-Löf: Constr. Math. and Comp. Prog.

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

- Transition: $M \longmapsto M'$, one step of execution.
- Termination: M val is canonical/complete.

Define exact equality of closed types and terms:

- Type equality: $A \doteq B$ type $[\Psi]$.
- Term equality in a type: $M \doteq N \in A$ $[\Psi]$.

Computational Meaning Explanation

Martin-Löf: Constr. Math. and Comp. Prog.

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

- Transition: $M \longmapsto M'$, one step of execution.
- Termination: M val is canonical/complete.

Define exact equality of closed types and terms:

- Type equality: $A \doteq B$ type $[\Psi]$.
- Term equality in a type: $M \doteq N \in A$ $[\Psi]$.

Extend to open forms by functionality aka extensionality:

- Types: $a_1{:}A_1, \ldots, a_n{:}A_n \gg A \doteq B$ type $[\Psi]$.
- Terms: $a_1{:}A_1, \ldots, a_n{:}A_n \gg M \doteq N \in A$ $[\Psi]$.

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.

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

Syntax is organized cubically:

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

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

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 $x, y, z, \ldots$.

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

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 $\psi(x) = 0/1/x' \in \Psi'$.

Cubical Programming Language

Substitutions define the aspects of a cube E:

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

$$
\begin{tikzcd}
\cdot \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge, row sep=huge]
E\langle 0/x\rangle\langle 0/y\rangle \arrow[r, "E\langle 0/y\rangle"] \arrow[d, "E\langle 0/x\rangle"'] \arrow[dr, phantom, "E"] & E\langle 1/x\rangle\langle 0/y\rangle \arrow[d, "E\langle 1/x\rangle"] \\
E\langle 0/x\rangle\langle 1/y\rangle \arrow[r, "E\langle 1/y\rangle"'] & E\langle 1/x\rangle\langle 1/y\rangle
\end{tikzcd}
$$

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: $\mathbb{S}^1, \mathsf{base}, \mathsf{loop}_x, \mathbb{S}^1\text{-}\mathsf{elim}_{a.A}(M; M_\mathsf{b}, x.M_\mathsf{l})$.
- Negation: not_x, a type line, and glueing, $\mathsf{notel}_x(M)$.
- Kan operations: coe, hcom.

Cubical Programming Language

Conventional functional programming constructs:

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

Unconventional functional programming constructs:

- Circle: $\mathbb{S}^1, \mathsf{base}, \mathsf{loop}_x, \mathbb{S}^1\text{-}\mathsf{elim}_{a.A}(M; M_{\mathsf{b}}, x.M_{\mathsf{l}})$.
- Negation: not_x, a type line, and glueing, $\mathsf{notel}_x(M)$.
- Kan operations: coe, hcom.

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

Kan Operations

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

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

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

Kan Operations

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

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

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

- Homogeneous: within type, not line, A.

Kan Operations

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

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

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

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

Kan Operations

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

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

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

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

Kan Operations

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

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

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

- Homogeneous: within type, not line, A.
- The start r and end r' dimensions.
- The cap M is the starting cube.
- The tubes $\overrightarrow{y.N_i^\varepsilon}$ with extent $\overrightarrow{r_i}$ in dimension $\overrightarrow{y_i}$.

Kan Operations

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

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

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

- Homogeneous: within type, not line, A.
- The start r and end r' dimensions.
- The cap M is the starting cube.
- The tubes $\overrightarrow{y.N_i^\varepsilon}$ 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

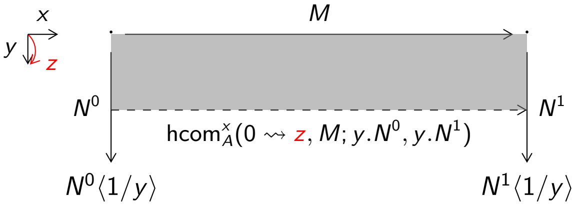
$$
\begin{tikzcd}
\cdot \arrow[rr, "M"] \arrow[rr, dashed, bend right, "{\mathsf{hcom}^x_A(0 \rightsquigarrow 0, M; y.N^0, y.N^1)}"'] \arrow[d, "N^0"'] & & \cdot \arrow[d, "N^1"] \\
N^0\langle 1/y\rangle & & N^1\langle 1/y\rangle
\end{tikzcd}
$$

Two-Dimensional Compositions

$$
\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
$$

$$
\begin{tikzcd}[column sep=huge, row sep=large]
\cdot \arrow[r, "M"] \arrow[d, "N^0"'] & \cdot \arrow[d, "N^1"] \\
N^0\langle 1/y\rangle \arrow[r, dashed, "\mathsf{hcom}^x_A(0 \rightsquigarrow 1, M; y.N^0, y.N^1)"'] & N^1\langle 1/y\rangle
\end{tikzcd}
$$

Two-Dimensional Compositions



Cubical Meaning Explanation

Explanation proceeds in stages:

- Define the canonical types and their elements at each dimension Ψ.
- 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 Ψ.
- 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.

Cubical Type Systems

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

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: $A_0 \approx^{\Psi} B_0$.
- Canonical elements of a canonical type: $M_0 \approx^{\Psi}_{A_0} N_0$.

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 $\approx^{\Psi}_{B_0}$.

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:

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 $A \longmapsto^* A_0$ and $B \longmapsto^* B_0$ and $A_0 \approx^{\Psi} B_0$.

Cubical Type Systems

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

- Canonical types: $A_0 \approx^{\Psi} B_0$.
- 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 $A \longmapsto^* A_0$ and $B \longmapsto^* B_0$ and $A_0 \approx^{\Psi} B_0$.
- $M \sim^{\Psi}_{A} N$ iff $M \longmapsto^* M_0$, $N \longmapsto^* N_0$, $A \longmapsto^* A_0$, and $M_0 \approx^{\Psi}_{A_0} N_0$.

Pre-Types: 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 $\psi_1 : \Psi_1 \to \Psi$ and $\psi_2 : \Psi_2 \to \Psi_1$.

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 $A\psi_1 \longmapsto^* A_1$ val, and $A_1\psi_2 \longmapsto^* A_2$ val, and $A\psi_2\psi_1 \longmapsto^* A_{12}$ val.

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

$$
\begin{tikzcd}
A \arrow[r, Rightarrow, "\psi_1"] \arrow[d, Rightarrow, "\psi_1\psi_2"'] & A_1 \arrow[d, Rightarrow, "\psi_2"] \\
A_{12} \arrow[r, phantom, "\approx^{\Psi_2}"] & A_2
\end{tikzcd}
$$

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

$$\begin{tikzcd} A \arrow[r, Rightarrow, "\psi_1"] \arrow[d, Rightarrow, "\psi_1\psi_2"'] & A_1 \arrow[d, Rightarrow, "\psi_2"] \\ A_{12} \approx^{\Psi_2} & A_2 \end{tikzcd}$$

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

Pre-Types and Types

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

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

Pre-Types and Types

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

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

A type is a Kan pre-type:

Pre-Types and Types

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

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

A type is a Kan pre-type:

- Supports coercion and composition.

Pre-Types and Types

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

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

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

$$\mathsf{coe}_{x.A\psi}^{r \rightsquigarrow r'}(M) \in A\psi\langle r'/x\rangle \; [\Psi'].$$

Kan Conditions for Coercion

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

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

then

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

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

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

then

$$\mathsf{coe}_{x.A\psi}^{r \rightsquigarrow r}(M) \doteq M \in A\psi\langle r/x\rangle\ [\Psi'].$$

Kan Conditions for Composition

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

- $M \in A\psi\ [\Psi']$,

Kan Conditions for Composition

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

- $M \in A\psi\ [\Psi']$,
- $N_i^{\varepsilon} \doteq N_j^{\varepsilon'} \in A\psi\ [\Psi', y \mid r_i = \varepsilon, r_j = \varepsilon']$ (all i, j, ε, and ε')

Kan Conditions for Composition

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

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

Kan Conditions for Composition

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

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

then

Kan Conditions for Composition

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

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

then

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

Kan Conditions for Composition

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

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

then

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

Kan Conditions for Composition

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

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

then

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

Kan Conditions for Composition

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

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

then

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

Constraints limit applicable substitutions; conditions can be vacuous.

Defining Booleans

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.

Defining Booleans

The Booleans are defined as a higher inductive type.

- Innocent of its status as a set.
- Certain hcom's are values.

Defining Booleans

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.

Defining Booleans

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 dynamics of the conditional accounts for

- true and false, as usual.
- hcom's that are values.

Boolean Dynamics

$$\frac{}{\mathsf{bool}\ \mathsf{val}}$$

$$\frac{\overrightarrow{r_i} = x_1, \ldots, x_{i-1}, \varepsilon, r_{i+1}, \ldots, r_n}{\mathsf{hcom}^{\overrightarrow{r_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto N_i^\varepsilon\langle r'/y\rangle}$$

$$\frac{r = r'}{\mathsf{hcom}^{x_1,\ldots,x_n}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon}) \longmapsto M}$$

$$\frac{}{\mathsf{true}\ \mathsf{val}} \qquad \frac{}{\mathsf{false}\ \mathsf{val}}$$

$$\frac{r \neq r'}{\mathsf{hcom}^{x_1,\ldots,x_n}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^\varepsilon})\ \mathsf{val}}$$

Boolean Dynamics

$$\frac{M \longmapsto M'}{\mathsf{if}_{a.A}(M; T, F) \longmapsto \mathsf{if}_{a.A}(M'; T, F)} \qquad \frac{}{\mathsf{if}_{a.A}(\mathsf{true}; T, F) \longmapsto T}$$

$$\frac{}{\mathsf{if}_{a.A}(\mathsf{false}; T, F) \longmapsto F}$$

$$\frac{r \neq r' \quad H = \mathsf{hcom}_{\mathsf{bool}}^{x_1,\ldots,x_n}(r \rightsquigarrow z, M; \overrightarrow{y.N_i^{\varepsilon}})}{\begin{array}{c}\mathsf{if}_{a.A}(\mathsf{hcom}_{\mathsf{bool}}^{x_1,\ldots,x_n}(r \rightsquigarrow r', M; \overrightarrow{y.N_i^{\varepsilon}}); T, F) \\ \longmapsto \\ \mathsf{com}_{z.A[H/a]}^{x_1,\ldots,x_n}(r \rightsquigarrow r', \mathsf{if}_{a.A}(M; T, F); \overrightarrow{y.\mathsf{if}_{a.A}(N_i^{\varepsilon}; T, F)})\end{array}}$$

$$\frac{}{\mathsf{coe}_{x.\mathsf{bool}}^{r \rightsquigarrow r'}(M) \longmapsto M}$$

Canonical Booleans

A CTS has booleans if bool $\approx^{\Psi}$ bool and $\approx^{\Psi}_{\mathrm{bool}}$ is least s.t.

Canonical Booleans

A CTS has booleans if bool $\approx^{\Psi}$ bool and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- true $\approx^{\Psi}_{\mathsf{bool}}$ true,

Canonical Booleans

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

Canonical Booleans

A CTS has booleans if $\mathsf{bool} \approx^{\Psi} \mathsf{bool}$ and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- $\mathsf{true} \approx^{\Psi}_{\mathsf{bool}} \mathsf{true}$,
- $\mathsf{false} \approx^{\Psi}_{\mathsf{bool}} \mathsf{false}$, and
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\mathsf{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when

Canonical Booleans

A CTS has booleans if bool $\approx^{\Psi}$ bool and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- true $\approx^{\Psi}_{\mathsf{bool}}$ true,
- false $\approx^{\Psi}_{\mathsf{bool}}$ false, and
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\mathsf{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when
  - $r \neq r'$,

Canonical Booleans

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
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\text{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\text{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when
  - $r \neq r'$,
  - $M \doteq O \in \text{bool}\ [\Psi]$,

Canonical Booleans

A CTS has booleans if $\mathsf{bool} \approx^{\Psi} \mathsf{bool}$ and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- $\mathsf{true} \approx^{\Psi}_{\mathsf{bool}} \mathsf{true}$,
- $\mathsf{false} \approx^{\Psi}_{\mathsf{bool}} \mathsf{false}$, and
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\mathsf{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when
  - $r \neq r'$,
  - $M \doteq O \in \mathsf{bool}\ [\Psi]$,
  - $N^{\varepsilon}_i \doteq N^{\varepsilon'}_j \in \mathsf{bool}\ [\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$,

Canonical Booleans

A CTS has booleans if $\mathsf{bool} \approx^{\Psi} \mathsf{bool}$ and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- $\mathsf{true} \approx^{\Psi}_{\mathsf{bool}} \mathsf{true}$,
- $\mathsf{false} \approx^{\Psi}_{\mathsf{bool}} \mathsf{false}$, and
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\mathsf{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when
  - $r \neq r'$,
  - $M \doteq O \in \mathsf{bool}\ [\Psi]$,
  - $N^{\varepsilon}_i \doteq N^{\varepsilon'}_j \in \mathsf{bool}\ [\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$,
  - $N^{\varepsilon}_i \doteq P^{\varepsilon}_i \in \mathsf{bool}\ [\Psi, y \mid x_i = \varepsilon]$ for all i, ε, and

Canonical Booleans

A CTS has booleans if $\mathsf{bool} \approx^{\Psi} \mathsf{bool}$ and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

- $\mathsf{true} \approx^{\Psi}_{\mathsf{bool}} \mathsf{true}$,
- $\mathsf{false} \approx^{\Psi}_{\mathsf{bool}} \mathsf{false}$, and
- $\mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', M; \overrightarrow{y.N^{\varepsilon}_i}) \approx^{\Psi,x}_{\mathsf{bool}} \mathsf{hcom}^{\overrightarrow{x_i}}_{\mathsf{bool}}(r \rightsquigarrow r', O; \overrightarrow{y.P^{\varepsilon}_i})$ when
  - $r \neq r'$,
  - $M \doteq O \in \mathsf{bool}\ [\Psi]$,
  - $N^{\varepsilon}_i \doteq N^{\varepsilon'}_j \in \mathsf{bool}\ [\Psi, y \mid x_i = \varepsilon, x_j = \varepsilon']$ for all $i, j, \varepsilon, \varepsilon'$,
  - $N^{\varepsilon}_i \doteq P^{\varepsilon}_i \in \mathsf{bool}\ [\Psi, y \mid x_i = \varepsilon]$ for all i, ε, and
  - $N^{\varepsilon}_i\langle r/y\rangle \doteq M \in \mathsf{bool}\ [\Psi \mid x_i = \varepsilon]$ for all i, ε.

Canonical Booleans

A CTS has booleans if $\mathsf{bool} \approx^{\Psi} \mathsf{bool}$ and $\approx^{\Psi}_{\mathsf{bool}}$ is least s.t.

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

Guarantees canonicity for closed points in bool: all evaluate to either true or false.

Not as a Type Line

Define not_x as a type line between bool and bool.

Not as a Type Line

Define not_x as a type line between bool and bool.

- Given by negation (swapping) as a (strict) equivalence.

Not as a Type Line

Define 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 not_x as a type line between bool and bool.

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

The term $\mathsf{notel}_x(M) \in \mathsf{not}_x\ [\Psi, x]$ is a use of gluing [CCHM16]:

$$
\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} & \cdot \arrow[r, "\mathsf{bool}"] \arrow[d, maps to, "\mathsf{not}(-)"'] & \cdot \arrow[d, maps to, "\mathsf{id}"] \\
{} & & \mathsf{bool} \arrow[r, dashed, "\mathsf{not}_x"'] & \mathsf{bool}
\end{tikzcd}
$$

Other Types Considered

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

Other Types Considered

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

- Same as LB14 and CCHM16, but not HoTT.

Other Types Considered

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

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

Other Types Considered

Identification type $\mathsf{Id}_{x.A}(M, N)$ 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.

Other Types Considered

Identification type $\mathsf{Id}_{x.A}(M, N)$ 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 $\mathsf{Id}_{x.A}(M, N)$ 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?

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.]

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.

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.

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.

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.

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.)

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).

References

Stuart F Allen, Mark Bickford, Robert L Constable, Richard Eaton, Christoph Kreitz, Lori Lorigo, and Evan Moran.
Innovations in computational type theory using Nuprl.
Journal of Applied Logic, 4(4):428–469, 2006.

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

Carlo Angiuli, Robert Harper, and Todd Wilson.
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.
In *19th International Conference on Types for Proofs and Programs (TYPES 2013)*, volume 26, pages 107–128, 2014.

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.