Computational (Higher) Type Theory

Robert Harper and Carlo Angiuli

Computer Science Department
Carnegie Mellon University

ACM PoPL Tutorial Session January 2018

Vladimir Voevodsky 1966-2017

Photo credit: Wikipedia

Acknowledgements

Thanks to many, including

- Collaborators: Evan Cavallo, Kuen-Bang Hou (Favonia), Daniel R. Licata, Jonathan Sterling, Todd Wilson.
- Colleagues: Steve Awodey, Marc Bezem, Guillaume Brunerie, Thierry Coquand, Simon Huber, Anders Mörtberg.
- Inspiration: Robert Constable, Per Martin-Löf, Dana Scott, Vladimir Voevodsky.

Supported by AFOSR MURI FA9550-15-1-0053.

References

Primary sources for these lectures:

- Carlo Angiuli and Robert Harper. "Meaning Explanations at Higher Dimension." *Indagationes Mathematicae* 29 (2018), pages 135–149. Special Issue: L.E.J. Brouwer after 50 years.
- Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. "Computational Higher Type Theory III: Univalent Universes and Exact Equality." `https://arxiv.org/abs/1712.01800`.
- Evan Cavallo and Robert Harper. "Computational Higher Type Theory IV: Inductive Types." `https://arxiv.org/abs/1801.01568`.

See also:

- Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical type theory: a constructive interpretation of the univalence axiom." To appear, 2018.
- Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. "Cartesian Cubical Type Theory." To appear, 2018.

Formal Type Theory

Martin-Löf; Coquand; HoTT

A formal type theory is inductively defined by rules:

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

Formal Type Theory

Martin-Löf; Coquand; HoTT

A 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

Martin-Löf; Coquand; HoTT

A 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.

Ought to admit a computational interpretation as programs.

Intensional Type Theory

Martin-Löf

The canonical formal dependent type theory: ITT.

- Inductive types: nat, bool, sums, well-founded trees.
- Dependent function and product types: $\Pi x{:}A.B$, $\Sigma x{:}A.B$.
- Identity type: $\mathrm{Id}_A(M, N)$.

Intensional Type Theory

Martin-Löf

The canonical formal dependent type theory: ITT.

- Inductive types: nat, bool, sums, well-founded trees.
- Dependent function and product types: $\Pi x{:}A.B$, $\Sigma x{:}A.B$.
- Identity type: $\mathsf{Id}_A(M, N)$.

Identity type is the least reflexive relation:

- Reflexivity: $\mathsf{refl}_A(M) : \mathsf{Id}_A(M, M)$.
- Induction: if $P : \mathsf{Id}_A(M, N)$ and $u{:}A \vdash Q : C[M, M, \mathsf{refl}_A(M)]$, then $\mathsf{J}(u.Q; P) : C[M, N, P]$.

Normalization: reduction of open terms.

- Variables are indeterminates, obey substitution.
- Canonicity via characterization of closed normal forms.

Computational Meaning of ITT

Martin-Löf

Normalization: reduction of open terms.

- Variables are indeterminates, obey substitution.
- Canonicity via characterization of closed normal forms.

Meaning explanations: evaluation of closed terms.

- Variables range over closed terms, obey functionality.
- Canonicity by definition of observable values.

Identity Type in ITT

Equational reasoning is handled by the identity type:

$$x : \mathsf{nat}, y : \mathsf{nat} \vdash P(x, y) : \mathsf{Id}_{\mathsf{nat}}(x + y, y + x)$$

The proof $P(x, y)$ is non-trivial: induction on x and y.

Identity Type in ITT

Equational reasoning is handled by the identity type:

$$x : \mathsf{nat}, y : \mathsf{nat} \vdash P(x, y) : \mathsf{Id}_{\mathsf{nat}}(x + y, y + x)$$

The proof $P(x, y)$ is non-trivial: induction on x and y.

Type families respect identity proofs:

$$x, y : \mathsf{nat} \vdash \mathsf{Vec}^{\dagger}(P(x, y)) : \mathsf{Id}_{\mathcal{U}}(\mathsf{Vec}(x + y), \mathsf{Vec}(y + x)).$$

Identity Type in ITT

Identity proofs in $\mathsf{Id}_{\mathcal{U}}(A, B)$ induce coercions:

$$a, b : \mathcal{U}, p : \mathsf{Id}_{\mathcal{U}}(a, b) \vdash \mathsf{coerce}(p) : a \to b$$

In particular, for any $M, N : \mathsf{nat}$,

$$\mathsf{coerce}(\mathsf{Vec}^{\dagger}(P(M, N))) : \mathsf{Vec}(M + N) \to \mathsf{Vec}(N + M)$$

Identity Type in ITT

Identity proofs in $\mathsf{Id}_{\mathcal{U}}(A, B)$ induce coercions:

$$a, b : \mathcal{U}, p : \mathsf{Id}_{\mathcal{U}}(a, b) \vdash \mathsf{coerce}(p) : a \to b$$

In particular, for any M, N : nat,

$$\mathsf{coerce}(\mathsf{Vec}^{\dagger}(P(M, N))) : \mathsf{Vec}(M + N) \to \mathsf{Vec}(N + M)$$

But for closed M and N these types are definitionally equal!

Thus, no coercion is needed at run-time!

Program extraction exploits irrelevance of identity proofs.

- Evaluate only closed terms of observable type.
- Erase uses of identity elimination.

Program extraction exploits irrelevance of identity proofs.

- Evaluate only closed terms of observable type.
- Erase uses of identity elimination.

Meaning explanation emphasizes extraction and execution.

- No transport operations to erase.
- Exact equality: $x, y : \mathsf{nat} \gg x + y \doteq y + x \in \mathsf{nat}$.

Homotopy Type Theory

Hofmann & Streicher; Awodey & Warren; Voevodsky

$\mathrm{Id}_A(M, N)$ may be considered as type of paths.

Homotopy Type Theory

Hofmann & Streicher; Awodey & Warren; Voevodsky

$\mathsf{Id}_A(M, N)$ may be considered as type of paths.

Univalence: if $E : \mathsf{Equiv}(A, B)$ is an equivalence, then

$$\mathsf{ua}(E) : \mathsf{Id}_{\mathcal{U}}(A, B).$$

Higher inductive types, such as the “circle”, $\mathbb{C}$:

$$\begin{aligned} &\mathsf{base} : \mathbb{C} \\ &\mathsf{loop} : \mathsf{Id}_{\mathbb{C}}(\mathsf{base}, \mathsf{base}). \end{aligned}$$

Coercions are no longer erasable!

$$\mathsf{coerce}(\mathsf{ua}(\ldots)) : \mathsf{nat} + \mathsf{nat} \to \mathsf{bool} \times \mathsf{nat}$$

(Even for closed terms.)

Homotopy Type Theory

Coercions are no longer erasable!

$$\mathsf{coerce}(\mathsf{ua}(\dots)) : \mathsf{nat} + \mathsf{nat} \to \mathsf{bool} \times \mathsf{nat}$$

(Even for closed terms.)

What is the computational content of HoTT?

$$\mathsf{coerce}(\mathsf{ua}(\dots)) \longmapsto \textbf{???}$$

Identity elimination does not eliminate identifications!

Higher Meaning Explanations

Judgmental account of higher structure of types:

- What is a path in a type?
- Define the action of a path.
- Ensure that paths can be composed.

Higher Meaning Explanations

Judgmental account of higher structure of types:

- What is a path in a type?
- Define the action of a path.
- Ensure that paths can be composed.

Identity type splits into two concepts:

- Exact equality: $M \doteq N \in A$.
- Path type: $\mathsf{Path}_{x.A}(M, N)$.

Computational Meaning Explanations

Martin-Löf; Constable; Allen

Start with a programming language:

- Programs are closed terms.
- Evaluation $M \Downarrow V$ to a canonical form aka value.

Computational Meaning Explanations

Martin-Löf; Constable; Allen

Start with a programming language:

- Programs are closed terms.
- Evaluation $M \Downarrow V$ to a canonical form aka value.

Types are programs that name specifications of programs.

- A type means $A \Downarrow V$ and V names a specification.
- if A type, then $M \doteq M' \in A$ means $M \Downarrow V$ and $M' \Downarrow V'$ and V and V' behave the same in the sense of A.

Computational Meaning Explanations

Martin-Löf; Constable; Allen

Start with a programming language:

- Programs are closed terms.
- Evaluation $M \Downarrow V$ to a canonical form aka value.

Types are programs that name specifications of programs.

- A type means $A \Downarrow V$ and V names a specification.
- if A type, then $M \doteq M' \in A$ means $M \Downarrow V$ and $M' \Downarrow V'$ and V and V' behave the same in the sense of A.

What matters is behavior, not form!

Computational Meaning Explanations

Variables are interpreted semantically.

- Range over closed terms satisfying a type.
- Respect equality at that type.

Computational Meaning Explanations

Variables are interpreted semantically.

- Range over closed terms satisfying a type.
- Respect equality at that type.

Functionality: $a : A \gg N \in B$ means

$$M \doteq M' \in A \text{ implies } N[M/a] \doteq N[M'/a] \in B[M/a].$$

Extensionality: $a : A \gg N \doteq N' \in B$ means

$$M \doteq M' \in A \text{ implies } N[M/a] \doteq N'[M'/a] \in B[M/a].$$

Computational Meaning Explanations

Proof theories are secondary, a matter of pragmatics.

- No privileged proof theory. (Down with C-H!).
- No requirement of decidability of judgments.

Computational Meaning Explanations

Proof theories are secondary, a matter of pragmatics.

- No privileged proof theory. (Down with C-H!).
- No requirement of decidability of judgments.

RedPRL proof theory is a refinement logic.

- Inspired by NuPRL.
- Emphasizes program extraction.

Computational Meaning Explanations

Proof theories are secondary, a matter of pragmatics.

- No privileged proof theory. (Down with C-H!).
- No requirement of decidability of judgments.

RedPRL proof theory is a refinement logic.

- Inspired by NuPRL.
- Emphasizes program extraction.

Inverts the conceptual order in ITT and related formalisms!

Computational Meaning Explanations

A specification is a symmetric, transitive relation on closed values.

Equal specifications must specify the same behavior,
i.e., be interchangeable as classifiers.

The construction of a type system ensures that specifications satisfy these conditions.

Booleans

Programs:

- bool, true, false are canonical.
- $\mathsf{if}(\mathsf{true}; P; Q) \longmapsto P$.
- $\mathsf{if}(\mathsf{false}; P; Q) \longmapsto Q$.
- if $M \longmapsto M'$ then $\mathsf{if}(M; P; Q) \longmapsto \mathsf{if}(M'; P; Q)$.

Booleans

Programs:

- bool, true, false are canonical.
- $\text{if}(\text{true}; P; Q) \longmapsto P$.
- $\text{if}(\text{false}; P; Q) \longmapsto Q$.
- if $M \longmapsto M'$ then $\text{if}(M; P; Q) \longmapsto \text{if}(M'; P; Q)$.

The type bool specifies that true and false are equal only to themselves.

bool is an inductive type.

Theorem (Dependent Elimination)

If $M \in \mathsf{bool}$ *and* $P \in A[\mathsf{true}/a]$ *and* $Q \in A[\mathsf{false}/a]$, *then* $\mathsf{if}(M; P; Q) \in A[M/a]$.

Theorem (Dependent Elimination)

If $M \in \mathsf{bool}$ *and* $P \in A[\mathsf{true}/a]$ *and* $Q \in A[\mathsf{false}/a]$*, then* $\mathsf{if}(M; P; Q) \in A[M/a]$.

Theorem (Behavioral Typing)

If $M \doteq \mathsf{true} \in \mathsf{bool}$ *and* $P \in A[\mathsf{true}/a]$*, then* $\mathsf{if}(M; P; Q) \in A[M/a]$.

Theorem (Dependent Elimination)

If $M \in \mathsf{bool}$ *and* $P \in A[\mathsf{true}/a]$ *and* $Q \in A[\mathsf{false}/a]$, *then* $\mathsf{if}(M; P; Q) \in A[M/a]$.

Theorem (Behavioral Typing)

If $M \doteq \mathsf{true} \in \mathsf{bool}$ *and* $P \in A[\mathsf{true}/a]$, *then* $\mathsf{if}(M; P; Q) \in A[M/a]$.

Theorem (Shannon Expansion)

If $a : \mathsf{bool} \gg M \in A$, *then*

$$a : \mathsf{bool} \gg M \doteq \mathsf{if}(a; M[\mathsf{true}/a]; M[\mathsf{false}/a]) \in A.$$

Functions

Programs:

- $(a{:}A) \to B$ and $\lambda a.M$ are canonical.
- $\mathrm{app}(\lambda a.P, N) \longmapsto P[N/a]$.
- if $M \longmapsto M'$, then $\mathrm{app}(M, N) \longmapsto \mathrm{app}(M', N)$.

Functions

Programs:

- $(a{:}A) \to B$ and $\lambda a.M$ are canonical.
- $\mathrm{app}(\lambda a.P, N) \longmapsto P[N/a]$.
- if $M \longmapsto M'$, then $\mathrm{app}(M, N) \longmapsto \mathrm{app}(M', N)$.

The value $\lambda a.M$ satisfies the spec. $(a{:}A) \to B$ iff

$$a : A \gg M \in B.$$

Functions

Programs:

- $(a{:}A) \to B$ and $\lambda a.M$ are canonical.
- $\mathsf{app}(\lambda a.P, N) \longmapsto P[N/a]$.
- if $M \longmapsto M'$, then $\mathsf{app}(M, N) \longmapsto \mathsf{app}(M', N)$.

The value $\lambda a.M$ satisfies the spec. $(a{:}A) \to B$ iff

$$a : A \gg M \in B.$$

Values $\lambda a.M$ and $\lambda a.M'$ are equal in $(a{:}A) \to B$ iff

$$a : A \gg M \doteq M' \in B \ [\Psi].$$

Theorem (Dependent Elim)

If $M \in (a{:}A) \to B$, *and* $N \in A$, *then* $\mathsf{app}(M, N) \in B[N/a]$.

Functions

Theorem (Dependent Elim)

If $M \in (a{:}A) \to B$, *and* $N \in A$, *then* $\mathsf{app}(M, N) \in B[N/a]$.

Theorem (β Equality)

If $\lambda a.P \in (a{:}A) \to B$ *and* $N \in A$, *then*

$$\mathsf{app}(\lambda a.P, N) \doteq P[N/a] \in B[N/a].$$

Functions

Theorem (Dependent Elim)
If $M \in (a{:}A) \to B$, *and* $N \in A$, *then* $\mathsf{app}(M, N) \in B[N/a]$.

Theorem (β Equality)
If $\lambda a.P \in (a{:}A) \to B$ *and* $N \in A$, *then*

$$\mathsf{app}(\lambda a.P, N) \doteq P[N/a] \in B[N/a].$$

Theorem (Extensionality)
If $a : A \gg \mathsf{app}(M, a) \doteq \mathsf{app}(N, a) \in B$, *then*

$$M \doteq N \in (a{:}A) \to B.$$

Exact Equality

Martin-Löf

Programs:

- $\mathsf{Eq}_A(M, N)$ and $\star$ are canonical.
- No elimination form needed!

The value $\star$ satisfies spec. $\mathsf{Eq}_A(M, N)$ iff $M \doteq N \in A$.

The value $\star$ is equal only to itself whenever it satisfies $\mathsf{Eq}_A(M, N)$.

Theorem

If $M \in A$, then $\star \in \mathsf{Eq}_A(M, M)$.

Theorem
If $M \in A$, *then* $\star \in \mathsf{Eq}_A(M, M)$.

Theorem
If $P \in \mathsf{Eq}_A(M, N)$, *then* $M \doteq N \in A$.

Demonstration

Please enjoy Carlo's demo of RedPRL!

Obligatory Cat Photo

Thanks to Tran Ma

Higher Meaning Explanations

HoTT encodes path structure in identification types:

$$A, \quad \mathsf{Id}_A(M, N), \quad \mathsf{Id}_{\mathsf{Id}_A(M,N)}(P, Q), \ldots$$

Higher Meaning Explanations

HoTT encodes path structure in identification types:

$$A, \quad \mathsf{Id}_A(M, N), \quad \mathsf{Id}_{\mathsf{Id}_A(M,N)}(P, Q), \ldots$$

Paths re-expressed using the interval $\mathbb{I} = [0, 1]$:

- Points: A.
- Lines btw points: $\mathbb{I} \rightsquigarrow A$,
- Squares, lines btw lines: $\mathbb{I} \rightsquigarrow (\mathbb{I} \rightsquigarrow A) \cong \mathbb{I}^2 \rightsquigarrow A$,
- Cubes, lines btw squares: $\mathbb{I}^3 \rightsquigarrow A$, ...
- n-cubes: $\mathbb{I}^n \rightsquigarrow A$

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

Cubical syntax:

- Dimensions $r := 0 \mid 1 \mid x$.
- Contexts $\Psi = x_1, \ldots, x_n$.
- Substitutions $\psi = \langle r_1/x_1, \ldots r_n/x_n \rangle : \Psi' \to \Psi$.
- Action on terms: $M\,\psi$

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

Cubical syntax:

- Dimensions $r := 0 \mid 1 \mid x$.
- Contexts $\Psi = x_1, \ldots, x_n$.
- Substitutions $\psi = \langle r_1/x_1, \ldots r_n/x_n \rangle : \Psi' \to \Psi$.
- Action on terms: $M\,\psi$

Cartesian cubes = substitutions are structural:

- Faces: $0/x$, $1/x$.
- Re-indexing: y/x.
- Weakening aka degeneracy: silent.
- Exchange aka symmetry: $y, x/x, y$.
- Contraction aka diagonal: $z, z/x, y$.

Cubical Programming Language

Substitutions act on cubes:

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

Substitutions act on cubes:

- Faces: $M\langle 0/x\rangle$, $M\langle 1/x\rangle$

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

Substitutions act on cubes:

- Faces: $M\langle 0/x\rangle$, $M\langle 1/x\rangle$
- Diagonals: $M\langle x/y\rangle$

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

Substitutions act on cubes:

- Faces: $M\langle 0/x\rangle$, $M\langle 1/x\rangle$
- Diagonals: $M\langle x/y\rangle$

$$\begin{tikzcd}[column sep=small]
{} \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge, row sep=huge]
{} \arrow[r, "M\langle 0/y\rangle"] \arrow[d, "M\langle 0/x\rangle"'] \arrow[dr, "M\langle x/y\rangle" description] & {} \arrow[d, "M\langle 1/x\rangle"] \\
{} \arrow[r, "M\langle 1/y\rangle"'] & {\begin{array}{l} M\langle 1/x\rangle\langle 1/y\rangle \\ \doteq M\langle 1/y\rangle\langle 1/x\rangle \\ \doteq M\langle x/y\rangle\langle 1/x\rangle \end{array}}
\end{tikzcd}$$

Cubical Programming Language

Any cube can be seen as a degenerate cube of higher dimension:

$$
\begin{tikzcd}
& x \\
y & \arrow[r] \arrow[d] & {} \\
& {}
\end{tikzcd}
\qquad
N\langle 0/x\rangle \xrightarrow{\quad N \quad} N\langle 1/x\rangle
$$

Cubical Programming Language

Any cube can be seen as a degenerate cube of higher dimension:

$$
\begin{tikzcd}
y \arrow[d] \arrow[r, "x"] & {} & N\langle 0/x \rangle \arrow[r, "N"] \arrow[d, equal, "N\langle 0/x \rangle"'] \arrow[dr, phantom, "N"] & N\langle 1/x \rangle \arrow[d, equal, "N\langle 1/x \rangle"] \\
{} & & N\langle 0/x \rangle \arrow[r, "N"'] & N\langle 1/x \rangle
\end{tikzcd}
$$

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

Evaluation: $M \Downarrow V\ [\Psi]$.

Evaluation: $M \Downarrow V\ [\Psi]$.

Conventional functional programming constructs:

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

Cubical Programming Language

Licata, Brunerie; Coquand, et al.

Evaluation: $M \Downarrow V\ [\Psi]$.

Conventional functional programming constructs:

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

Unconventional functional programming constructs:

- Circle: $\mathbb{C}, \mathsf{base}, \mathsf{loop}_x, \mathbb{C}\text{-}\mathsf{elim}_{a.A}(M; N, x.P)$.
- Kan operations: coe, hcom.

Cubical Programming Language

Evaluation is sensitive to dimensions:

$$\begin{aligned}
\mathsf{loop}_0 &\longmapsto \mathsf{base} \\
\mathsf{loop}_1 &\longmapsto \mathsf{base} \\
\mathbb{C}\text{-}\mathsf{elim}_{a.A}(\mathsf{base}; N, x.P) &\longmapsto N \\
\mathbb{C}\text{-}\mathsf{elim}_{a.A}(\mathsf{loop}_y; N, x.P) &\longmapsto P\langle y/x\rangle.
\end{aligned}$$

$$\mathsf{base} \doteq \mathsf{loop}_x\langle 0/x\rangle \xrightarrow{\quad \mathsf{loop}_x \quad} \mathsf{loop}_x\langle 1/x\rangle \doteq \mathsf{base}$$

Higher Meaning Explanations

If A type $[\Psi]$, then all faces of A evaluate to specifications:

- $A\psi \Downarrow V\ [\Psi']$ for all $\psi : \Psi' \to \Psi$, and
- Value V names a specification of values.

Higher Meaning Explanations

If A type $[\Psi]$, then all faces of A evaluate to specifications:

- $A\psi \Downarrow V$ $[\Psi']$ for all $\psi : \Psi' \to \Psi$, and
- Value V names a specification of values.

If $M \in A$ $[\Psi]$, then all faces of M satisfy the spec given by A.

That is, for every $\psi : \Psi' \to \Psi$,

- $M\psi \Downarrow V$, and
- V satisfies the specification given by $A\psi$.

Higher Meaning Explanations

If A type $[\Psi]$, then all faces of A evaluate to specifications:

- $A\,\psi \Downarrow V$ $[\Psi']$ for all $\psi : \Psi' \to \Psi$, and
- Value V names a specification of values.

If $M \in A$ $[\Psi]$, then all faces of M satisfy the spec given by A.

That is, for every $\psi : \Psi' \to \Psi$,

- $M\,\psi \Downarrow V$, and
- V satisfies the specification given by $A\,\psi$.

(Actually, we must define equal types and equal members.)

Cubical Specifications

Specifications are cubical symmetric, and transitive binary relations.

Cubical Specifications

Specifications are cubical symmetric, and transitive binary relations.

If V is a canonical type, then all of its faces must be types:

$$\text{for all } \psi : \Psi' \to \Psi,\ V\psi \text{ type } [\Psi'].$$

Cubical Specifications

Specifications are cubical symmetric, and transitive binary relations.

If V is a canonical type, then all of its faces must be types:

$$\text{for all } \psi : \Psi' \to \Psi,\ V\psi \text{ type } [\Psi'].$$

If W is canonical of type V, then its faces must be elements:

$$\text{for all } \psi : \Psi' \to \Psi,\ W\psi \in V\psi\ [\Psi'].$$

Coherence

An ambiguity arises for A type $[\Psi]$:

- $A\,\psi_1 \Downarrow V_1$ and $V_1\psi_2 \Downarrow V_2$.
- $A(\psi_1 \cdot \psi_2) \Downarrow V_{12}$.

Coherence

An ambiguity arises for A type $[\Psi]$:

- $A\,\psi_1 \Downarrow V_1$ and $V_1\psi_2 \Downarrow V_2$.
- $A(\psi_1 \cdot \psi_2) \Downarrow V_{12}$.

But are V_2 and V_{12} the *same* canonical type?

- Not necessarily the same program.
- But should have the same elements and equality.

Coherence demands that they determine the same specification.

Meaning of Variables

Term variables express functional dependence on closed values at all dimensions.

Thus $a : A \gg B$ type $[\Psi]$ means for all $\psi : \Psi' \to \Psi$,

$$\text{if } M \doteq N \in A\ [\Psi'],\ \text{then } B\psi[M/a] \doteq B\psi[N/a] \text{ type } [\Psi'].$$

Meaning of Variables

Term variables express functional dependence on closed values at all dimensions.

Thus $a : A \gg B$ type $[\Psi]$ means for all $\psi : \Psi' \to \Psi$,

$$\text{if } M \doteq N \in A\ [\Psi'],\ \text{then } B\psi[M/a] \doteq B\psi[N/a] \text{ type } [\Psi'].$$

In particular, type families transform lines into lines:

$$\text{if } \underbrace{M \in A\ [\Psi, x]}_{\text{line in } A},\ \text{then } \underbrace{B[M/a] \text{ type } [\Psi, x]}_{\text{line of types}}.$$

Pre- and Kan Types

Voevodsky (HTS)

These conditions define cubical pre-types

- From zero- to higher-dimensional types.
- Not sufficient for HoTT.

Pre- and Kan Types

Voevodsky (HTS)

These conditions define cubical pre-types

- From zero- to higher-dimensional types.
- Not sufficient for HoTT.

A full-fledged type must satisfy the Kan conditions:

- Type lines induce coercions between types.
- Paths must be closed under Kan composition.

Coercion along a Line of Types

Type lines A type $[\Psi, x]$ induce coercions:

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

Coercion along a Line of Types

Type lines A type $[\Psi, x]$ induce coercions:

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

Coercion along A type $[\Psi, x]$ is trivial when $r = r'$:

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

Coercion along a Line of Types

Type lines A type $[\Psi, x]$ induce coercions:

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

Coercion along A type $[\Psi, x]$ is trivial when $r = r'$:

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

Each type defines the meaning of coercion along lines!

Coercion along a Line of Types

Diagrammatically,

$$
\begin{tikzcd}
M \arrow[d, phantom, sloped, "\in"] & \\
A\langle 0/x\rangle \arrow[r, "A"'] & A\langle 1/x\rangle
\end{tikzcd}
$$

Coercion along a Line of Types

Diagrammatically,

$$
\begin{tikzcd}
M \arrow[r, dashed] \arrow[d, phantom, "\rotatebox{90}{$\in$}"] & \mathsf{coe}_{x.A}^{0 \rightsquigarrow 1}(M) \arrow[d, phantom, "\rotatebox{90}{$\in$}"] \\
A\langle 0/x \rangle \arrow[r, "A"'] & A\langle 1/x \rangle
\end{tikzcd}
$$

Diagrammatically,

$$\begin{tikzcd}
M \arrow[rr, "\mathsf{coe}_{x.A}^{0\rightsquigarrow x}(M)"] & & \mathsf{coe}_{x.A}^{0\rightsquigarrow 1}(M) \\
\rotatebox{90}{$\in$} & \rotatebox{90}{$\in$} & \rotatebox{90}{$\in$} \\
A\langle 0/x\rangle \arrow[rr, "A"'] & & A\langle 1/x\rangle
\end{tikzcd}$$

Diagrammatically,

$$\begin{tikzcd}
\mathsf{coe}_{x.A}^{0\rightsquigarrow 0}(M) \doteq M \arrow[rr, "\mathsf{coe}_{x.A}^{0\rightsquigarrow x}(M)"] \arrow[d, phantom, "\pitchfork"] & \arrow[d, phantom, "\pitchfork"] & \mathsf{coe}_{x.A}^{0\rightsquigarrow 1}(M) \arrow[d, phantom, "\pitchfork"] \\
A\langle 0/x\rangle \arrow[rr, "A"'] & & A\langle 1/x\rangle
\end{tikzcd}$$

Kan Composition

$$P\langle 0/x\rangle \xrightarrow{\quad P \quad} P\langle 1/x\rangle \qquad Q\langle 0/x\rangle \xrightarrow{\quad Q \quad} Q\langle 1/x\rangle$$

Paths in a type must compose.

- if $P \in A\ [\Psi, x]$, and $Q \in A\ [\Psi, x]$,

Kan Composition

$$P\langle 0/x\rangle \xrightarrow{\quad P \quad} P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \xrightarrow{\quad Q \quad} Q\langle 1/x\rangle$$

Paths in a type must compose.

- if $P \in A\ [\Psi, x]$, and $Q \in A\ [\Psi, x]$, and
- $P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \in A\ [\Psi]$,

Kan Composition

$$
\begin{tikzcd}
P\langle 0/x\rangle \arrow[r, "P"] \arrow[rr, bend right, "P \cdot Q"'] & P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \arrow[r, "Q"] & Q\langle 1/x\rangle
\end{tikzcd}
$$

Paths in a type must compose.

- if $P \in A\ [\Psi, x]$, and $Q \in A\ [\Psi, x]$, and
- $P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \in A\ [\Psi]$, then
- there exists $P \cdot Q \in A\ [\Psi, x]$,

Kan Composition

$$\begin{tikzcd} P\langle 0/x\rangle \arrow[r, "P"] \arrow[rr, bend right, "P \cdot Q"'] & P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \arrow[r, "Q"] & Q\langle 1/x\rangle \end{tikzcd}$$

Paths in a type must compose.

- if $P \in A\ [\Psi, x]$, and $Q \in A\ [\Psi, x]$, and
- $P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \in A\ [\Psi]$, then
- there exists $P \cdot Q \in A\ [\Psi, x]$,
- satisfying composition and identity laws up to higher paths.

Kan Composition

$$\begin{tikzcd} P\langle 0/x\rangle \arrow[r, "P"] \arrow[rr, bend right, "P \cdot Q"'] & P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \arrow[r, "Q"] & Q\langle 1/x\rangle \end{tikzcd}$$

Paths in a type must compose.

- if $P \in A\ [\Psi, x]$, and $Q \in A\ [\Psi, x]$, and
- $P\langle 1/x\rangle \doteq Q\langle 0/x\rangle \in A\ [\Psi]$, then
- there exists $P \cdot Q \in A\ [\Psi, x]$,
- satisfying composition and identity laws up to higher paths.

Miraculously, there is a simple way to capture the full meaning!

The HCom Diagram

Pictorially,

$$
\begin{tikzcd}
\cdot \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge, row sep=huge]
M_0 \doteq N_0 \arrow[rr, "M \doteq \mathsf{hcom}_A^{0 \rightsquigarrow 0}(M;\, \vec{T})"] \arrow[d, "N"'] & & M_1 \doteq P_0 \arrow[d, "P"] \\
N_1 \arrow[rr, dashed, "\mathsf{hcom}_A^{0 \rightsquigarrow 1}(M;\, \vec{T})"'] \arrow[urr, phantom, "\mathsf{hcom}_A^{0 \rightsquigarrow y}(M;\, \vec{T})"] & & P_1
\end{tikzcd}
$$

Symbolically,

$$\mathsf{hcom}_A^{0 \rightsquigarrow y}(\underbrace{M}_{\text{cap}}; \underbrace{x = 0 \hookrightarrow y.N, x = 1 \hookrightarrow y.P}_{\text{tube } \vec{T}}) \in A \; [\Psi, x, y].$$

Composition and Inversion from HCom

Concatenation and reversal are definable:

$$
\begin{tikzcd}
P_0 \arrow[r, "P"] \arrow[d, equal, "P_0"'] & P_1 \arrow[d, "Q"] \\
P_0 \arrow[r, dashed, "P \cdot Q"'] & Q_1
\end{tikzcd}
\qquad
\begin{tikzcd}
P_0 \arrow[r, equal, "P_0"] \arrow[d, "P"'] & P_0 \arrow[d, equal, "P_0"] \\
P_1 \arrow[r, dashed, "P^{-1}"'] & P_0
\end{tikzcd}
$$

Kan composition suffices to derive composition laws.

Strict Booleans

The type bool is defined such that for all M and Ψ,

$$M \in \mathsf{bool}\ [\Psi] \quad \text{iff} \quad M \Downarrow \mathsf{true} \text{ or } M \Downarrow \mathsf{false}.$$

Therefore, we can make bool Kan:

- $\mathsf{coe}^{r \rightsquigarrow r'}_{\_.\mathsf{bool}}(M) \longmapsto M$ for any M, r, r'.
- $\mathsf{hcom}^{r \rightsquigarrow r'}_{\mathsf{bool}}(M; \vec{T}) \longmapsto M$ for any M, $\vec{T}$, r, r'.

Strict Booleans

The type bool is defined such that for all M and Ψ,

$$M \in \mathsf{bool}\ [\Psi] \quad \text{iff} \quad M \Downarrow \mathsf{true} \text{ or } M \Downarrow \mathsf{false}.$$

Therefore, we can make bool Kan:

- $\mathsf{coe}^{r \rightsquigarrow r'}_{\_.\mathsf{bool}}(M) \longmapsto M$ for any M, r, r'.
- $\mathsf{hcom}^{r \rightsquigarrow r'}_{\mathsf{bool}}(M; \vec{T}) \longmapsto M$ for any M, $\vec{T}$, r, r'.

The properties of bool stated earlier carry over directly.

- Same proofs, using equality pre-type for equations.
- e.g., Shannon expansion.

Weak Booleans

Canonical:

- wbool, true, and false as before.
- $\mathsf{fcom}^{r \rightsquigarrow r'}(M; \vec{T})$, where $r \neq r'$.

Weak Booleans

Canonical:

- wbool, true, and false as before.
- $\mathsf{fcom}^{r \rightsquigarrow r'}(M; \vec{T})$, where $r \neq r'$.

Fcom = formal composition of booleans:

$$\begin{tikzcd} \cdot \arrow[r, "M"] \arrow[d, "N"'] & \cdot \arrow[d, "N'"] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd}$$

Weak Booleans

Conditional offloads composition to the motive at higher dims!

$$\mathrm{if}_{a.A}(\mathrm{true}; P; Q) \longmapsto P$$
$$\mathrm{if}_{a.A}(\mathrm{false}; P; Q) \longmapsto Q$$

$$\mathrm{if}_{a.A}\left(\begin{tikzcd} \cdot \arrow[r, "M"] \arrow[d, "N"'] & \cdot \arrow[d, "N'"] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd} ; P; Q\right) \longmapsto$$

$$\begin{tikzcd} \cdot \arrow[r, "\mathrm{if}_{a.A}(M; P; Q)"] \arrow[d, "\mathrm{if}_{a.A}(N; P; Q)"'] & \cdot \arrow[d, "\mathrm{if}_{a.A}(N'; P; Q)"] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd}$$

Result composition is heterogeneous.

The motive of the conditional on wbool must be Kan!

$$a : \mathsf{wbool} \gg A\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi].$$

Weak Booleans

The motive of the conditional on wbool must be Kan!

$$a : \mathsf{wbool} \gg A\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi].$$

Theorem (Dependent Elimination)

If $M \in \mathsf{bool}\ [\Psi]$ *and* $P \in A[\mathsf{true}/a]\ [\Psi]$ *and* $Q \in A[\mathsf{false}/a]\ [\Psi]$, *then*

$$\mathsf{if}_{a.A}(M; P; Q) \in A[M/a]\ [\Psi].$$

Weak Booleans

The motive of the conditional on wbool must be Kan!

$$a : \mathsf{wbool} \gg A\ \mathsf{type}_{\mathsf{Kan}}\ [\Psi].$$

Theorem (Dependent Elimination)

If $M \in \mathsf{bool}\ [\Psi]$ *and* $P \in A[\mathsf{true}/a]\ [\Psi]$ *and* $Q \in A[\mathsf{false}/a]\ [\Psi]$, *then*

$$\mathsf{if}_{a.A}(M; P; Q) \in A[M/a]\ [\Psi].$$

Looks unremarkable, but is not trivial because of higher dim's.

Circle

The circle $\mathbb{C}$ is like wbool.

$$\mathbb{C}\text{-elim}_{a.A}(\mathsf{base}; N, x.P) \longmapsto N$$

$$\mathbb{C}\text{-elim}_{a.A}(\mathsf{loop}_y; N, x.P) \longmapsto P\langle y/x\rangle$$

$$\mathbb{C}\text{-elim}_{a.A}\left(\begin{tikzcd} \cdot \arrow[r, "M'"] \arrow[d, "M"'] & \cdot \arrow[d, "M''"] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd} ; N, x.P\right) \longmapsto$$

$$\begin{tikzcd} \cdot \arrow[r, "\mathbb{C}\text{-elim}_{a.A}(M'; N, x.P)"] \arrow[d, "\mathbb{C}\text{-elim}_{a.A}(M; N, x.P)"'] & \cdot \arrow[d, "\mathbb{C}\text{-elim}_{a.A}(M''; N, x.P)"] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd}$$

Circle

The circle $\mathbb{C}$ is like wbool.

$$\mathbb{C}\text{-elim}_{a.A}(\text{base}; N, x.P) \longmapsto N$$

$$\mathbb{C}\text{-elim}_{a.A}(\text{loop}_y; N, x.P) \longmapsto P\langle y/x \rangle$$

$$\mathbb{C}\text{-elim}_{a.A}\left(M \begin{tikzcd} \cdot \arrow[r, "M'"] \arrow[d] & \cdot \arrow[d] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd} M'' ; N, x.P\right) \longmapsto$$

$$\mathbb{C}\text{-elim}_{a.A}(M; N, x.P) \begin{tikzcd} \cdot \arrow[r, "\mathbb{C}\text{-elim}_{a.A}(M'; N, x.P)"] \arrow[d] & \cdot \arrow[d] \\ \cdot \arrow[r, dashed] & \cdot \end{tikzcd} \mathbb{C}\text{-elim}_{a.A}(M''; N, x.P)$$

Iterations of loop defined using formal composition.

Functions

Abstraction and application as before:

- Canonical: $(a{:}A) \to B$, $\lambda a.M$.
- Non-canonical: $\mathsf{app}(M, N)$.
- Computation: $\mathsf{app}(\lambda a.P, N) \longmapsto P[N/a]$.

Functions

Abstraction and application as before:

- Canonical: $(a{:}A) \to B$, $\lambda a.M$.
- Non-canonical: $\mathsf{app}(M, N)$.
- Computation: $\mathsf{app}(\lambda a.P, N) \longmapsto P[N/a]$.

Coercion co- and contra-variantly:

$$\mathsf{coe}^{r\rightsquigarrow r'}_{x.(a:A)\to B}(M) \longmapsto \lambda a.\mathsf{coe}^{r\rightsquigarrow r'}_{x.B}(\mathsf{app}(M, \mathsf{coe}^{r'\rightsquigarrow r}_{x.A}(a))).$$

Functions

Abstraction and application as before:

- Canonical: $(a{:}A) \to B$, $\lambda a.M$.
- Non-canonical: $\mathsf{app}(M, N)$.
- Computation: $\mathsf{app}(\lambda a.P, N) \longmapsto P[N/a]$.

Coercion co- and contra-variantly:

$$\mathsf{coe}^{r \rightsquigarrow r'}_{x.(a:A)\to B}(M) \longmapsto \lambda a.\mathsf{coe}^{r \rightsquigarrow r'}_{x.B}(\mathsf{app}(M, \mathsf{coe}^{r' \rightsquigarrow r}_{x.A}(a))).$$

Kan composition by extensionality:

$$\mathsf{hcom}^{r \rightsquigarrow r'}_{(a:A)\to B}(M; \vec{T}) \longmapsto \lambda a.\mathsf{hcom}^{r \rightsquigarrow r'}_{B}(\mathsf{app}(M, a); \mathsf{app}(\vec{T}, a)).$$

Paths

The type $\mathsf{Path}_{x.A}(P_0, P_1)$ specifies paths in A with end points P_0 and P_1.

Dimension abstraction and application:

- $\mathsf{Path}_{x.A}(P_0, P_1)$, $\langle x \rangle M$ are canonical.
- $(\langle x \rangle M)@r \longmapsto M\langle r/x \rangle$.

Paths are Kan, provided that A is Kan.

Paths

Coercion: $\mathsf{coe}^{0\rightsquigarrow 1}_{y.\mathsf{Path}_{x.A}(P_0,P_1)}(M) \longmapsto$

$$\langle x\rangle \mathsf{com}^{0\rightsquigarrow 1}_{y.A}(M@x; x=0 \hookrightarrow y.P_0, x=1 \hookrightarrow y.P_1).$$

$$\begin{tikzcd}
{} \arrow[r, "x"] \arrow[d, "y"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}
P_0\langle 0/y\rangle \doteq M@0 \arrow[r, "M@x"] \arrow[d, "P_0"'] & M@1 \doteq P_1\langle 1/y\rangle \arrow[d, "P_1"] \\
P_0\langle 1/y\rangle \arrow[r, dashed] & P_1\langle 1/y\rangle
\end{tikzcd}$$

HoTT, Revisited

HoTT identity type splits into two concepts:

- Exact equality: extensional, evidence-free.
- Paths of arbitrary dimension.

HoTT, Revisited

HoTT identity type splits into two concepts:

- Exact equality: extensional, evidence-free.
- Paths of arbitrary dimension.

Both may be internalized:

- Equality pre-type, may or may not be Kan.
- Path type, always Kan.

HoTT, Revisited

HoTT identity type splits into two concepts:

- Exact equality: extensional, evidence-free.
- Paths of arbitrary dimension.

Both may be internalized:

- Equality pre-type, may or may not be Kan.
- Path type, always Kan.

Equality proofs are irrelevant and erasable.

Coercion and composition express the computational content of paths in each type.

HoTT, Revisited

Path type admits structure of identity type.

- Intro: $\mathsf{refl}_A(M) \in \mathsf{Path}_{\_.A}(M, M)$.
- Elim: $\mathsf{J}(u.Q; P)$ with $P \in \mathsf{Path}_{\_.A}(M, N)$.

Does not validate β law, because reflexivity is not special.

Jdentity type is definable as free Kan type on reflexivity:

- Validates β law for J.
- Elimination commutes with free Kan structure.

Admits computation: J is never "stuck."

But does not validate type-directed path laws!

HoTT, Revisited

Awodey; Cavallo

Jdentity type is definable as free Kan type on reflexivity:

- Validates β law for J.
- Elimination commutes with free Kan structure.

Admits computation: J is never "stuck."

But does not validate type-directed path laws!

It seems that we cannot have it both ways!

NOTATION	MEANING
$\Psi \mid \Gamma \Longrightarrow A \ \mathsf{true} \rightsquigarrow e$	There exists a term e such that if $\Gamma\ \mathit{ctx}\ [\Psi]$, then $\Gamma \gg A\ \mathit{type}_{\mathsf{pre}}\ [\Psi]$ and $\Gamma \gg e \in A\ [\Psi]$.
$\Psi \mid \Gamma \Longrightarrow A \doteq B\ \mathsf{type}_k$	If $\Gamma\ \mathit{ctx}\ [\Psi]$, then $\Gamma \gg A \doteq B\ \mathit{type}_k\ [\Psi]$.
$\Psi \mid \Gamma \Longrightarrow e\ \mathsf{synth} \rightsquigarrow A$	There exists a term A such that if $\Gamma\ \mathit{ctx}\ [\Psi]$, then $\Gamma \gg A\ \mathit{type}_{\mathsf{pre}}\ [\Psi]$ and $\Gamma \gg e \in A\ [\Psi]$.
$\Psi \mid \Gamma \Longrightarrow A \sqsubseteq B$	If $\Gamma\ \mathit{ctx}\ [\Psi]$, then $\Gamma \gg A\ \mathit{type}_{\mathsf{pre}}\ [\Psi]$ and $\Gamma \gg B\ \mathit{type}_{\mathsf{pre}}\ [\Psi]$, and $\Gamma, a : A \gg a \in B\ [\Psi]$.
$\Psi \mid \Gamma \Longrightarrow A \sqsubseteq \mathcal{U}^k_\omega$	If $\Gamma\ \mathit{ctx}\ [\Psi]$, then there exists some level i and kind $k' \leq k$ such that $\Gamma \gg A \doteq \mathcal{U}^{k'}_i\ \mathit{type}_{\mathsf{pre}}\ [\Psi]$.

Demonstration

Please enjoy Carlo's demonstration of RedPRL!

References I

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. Meaning explanations at higher dimension. *Indagationes Mathematicae*, 29:135–149, 2018. Virtual Special Issue – L.E.J. Brouwer after 50 years.

Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. Cartesian cubical type theory. (Unpublished manuscript), December 2017a.

Carlo Angiuli, Kuen-Bang Hou (Favonia), and Robert Harper. Computational higher type theory III: Univalent universes and exact equality. Preprint, December 2017b. URL `https://arxiv.org/abs/1712.01800`.

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.

Evan Cavallo and Robert Harper. Computational higher type theory IV: Inductive types. Preprint, January 2018. URL `https://arxiv.org/abs/1801.01568`.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical type theory: a constructive interpretation of the univalence axiom. to appear in the proceedings of TYPES 2015, 2015.

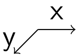
Simon Huber. Canonicity for cubical type theory. Preprint arXiv:1607.04156v1 [cs.LO], July 2016.

Jonathan Sterling, Kuen-Bang Hou (Favonia), Evan Cavallo, Carlo Angiuli, James Wilcox, Eugene Akentyev, David Christiansen, Daniel Gratzer, and Darin Morrison. RedPRL – the People's Refinement Logic. `http://www.redprl.org/`, 2017.

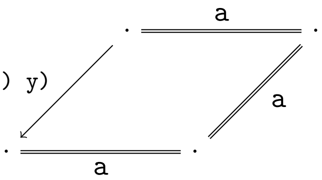
Vladimir Voevodsky. A simple type system with two identity types. Lecture notes, February 2013. URL `https://www.math.ias.edu/vladimir/sites/math.ias.edu.vladimir/files/HTS.pdf`.

InvRefl

```
(path [_] (path [_] ty a a) ($ PathInv ty a a (abs [_] a)) (abs [_] a))

abs x =>  abs y =>
```

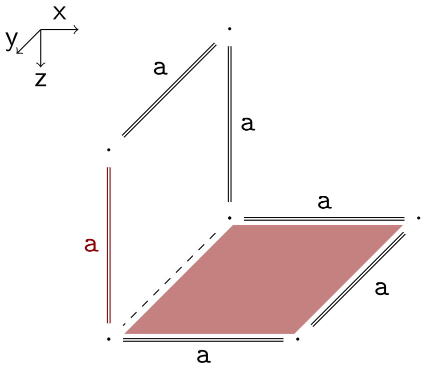



```
(@ ($ PathInv ty a a (abs [_] a)) y)
```



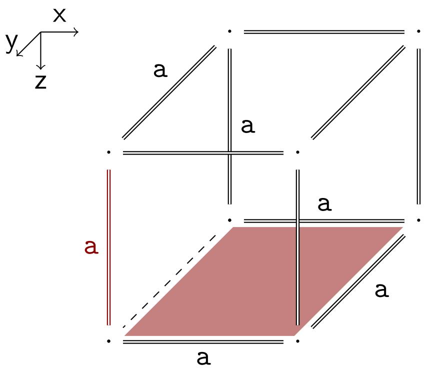
InvRefl

```
(path [_] (path [_] ty a a) ($ PathInv ty a a (abs [_] a)) (abs [_] a))

abs x =>  abs y =>
```



InvRefl

```
(path [_] (path [_] ty a a) ($ PathInv ty a a (abs [_] a)) (abs [_] a))

abs x =>  abs y =>
```



The Univalence Type

Favonia; CCHM

$$
\begin{tikzcd}
M \arrow[d, mapsto, "E"'] \arrow[dr, "\mathsf{Vin}_x(M, N)"] & \\
N_0 \arrow[r, "N"'] & N_1
\end{tikzcd}
$$

$$\in$$

$$
\begin{tikzcd}
A \arrow[d, "E"', "\wr"] \arrow[dr, "\mathsf{V}_x(A, B, E)"] & \\
B_0 \arrow[r, "B"'] & B_1
\end{tikzcd}
$$

The Univalence Type

Favonia; CCHM

$$
\begin{tikzcd}
M \arrow[r, "\mathsf{Vin}_x(M, N)"] \arrow[d, mapsto, "E"'] & N_1 \arrow[d, equal] \\
N_0 \arrow[r, "N"'] & N_1
\end{tikzcd}
$$

$$\in$$

$$
\begin{tikzcd}
A \arrow[r, "\mathsf{V}_x(A, B, E)"] \arrow[d, "E"', "\wr"] & B_1 \arrow[d, equal] \\
B_0 \arrow[r, "B"'] & B_1
\end{tikzcd}
$$

The Coe Diagram

Given $M \in A\ [\Psi, x]$:

$$\begin{tikzcd} M_0 \arrow[r, "M"] & M_1 \end{tikzcd} \quad \in \quad \begin{tikzcd} A_0 \arrow[r, "A"] & A_1 \end{tikzcd}$$

Then $\mathsf{coe}^{x \rightsquigarrow y}_{x.A_x}(M_x) \in A_y\ [\Psi, x, y]$:

$$\begin{tikzcd} \cdot \arrow[r, "x"] \arrow[d, "y"'] & \\ {} & \end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge, row sep=huge]
M_0 \arrow[r, "\mathsf{coe}^{x \rightsquigarrow 0}_{x.A_x}(M_x) \in A_0"] \arrow[d, "\mathsf{coe}^{0 \rightsquigarrow y}_{x.A_x}(M_0) \in A_y"'] \arrow[dr, phantom, "\mathsf{coe}^{x \rightsquigarrow y}_{x.A_x}(M_x)"] & \mathsf{coe}^{1 \rightsquigarrow 0}_{x.A_x}(M_1) \arrow[d, "\mathsf{coe}^{1 \rightsquigarrow y}_{x.A_x}(M_1) \in A_y"] \\
\mathsf{coe}^{0 \rightsquigarrow 1}_{x.A_x}(M_0) \arrow[r, "\mathsf{coe}^{x \rightsquigarrow 1}_{x.A_x}(M_x) \in A_1"'] & M_1
\end{tikzcd}$$

The Coe Diagram

Given $M \in A\ [\Psi, x]$:

$$\begin{tikzcd}[column sep=large]
M_0 \arrow[r, "M"] & M_1 & \in & A_0 \arrow[r, "A"] & A_1
\end{tikzcd}$$

Then $\mathrm{coe}_{x.A_x}^{x \rightsquigarrow y}(M_x) \in A_y\ [\Psi, x, y]$:

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

$$\begin{tikzcd}[column sep=huge, row sep=huge]
M_0 \arrow[r, "\mathrm{coe}_{x.A_x}^{x \rightsquigarrow 0}(M_x) \in A_0"] \arrow[d, "\mathrm{coe}_{x.A_x}^{0 \rightsquigarrow y}(M_0) \in A_y"'] \arrow[dr, "M" description] & \mathrm{coe}_{x.A_x}^{1 \rightsquigarrow 0}(M_1) \arrow[d, "\mathrm{coe}_{x.A_x}^{1 \rightsquigarrow y}(M_1) \in A_y"] \\
\mathrm{coe}_{x.A_x}^{0 \rightsquigarrow 1}(M_0) \arrow[r, "\mathrm{coe}_{x.A_x}^{x \rightsquigarrow 1}(M_x) \in A_1"'] & M_1
\end{tikzcd}$$

The Com Diagram

$$\mathsf{com}_{y.A}^{0\rightsquigarrow 1}(M; x = 0 \hookrightarrow y.N_0, x = 1 \hookrightarrow y.N_1) \in A\langle 1/y\rangle \ [\Psi, x]$$

$$\begin{tikzcd}
{} \arrow[r, "\mathsf{x}"] \arrow[d, "\mathsf{y}"'] & {} \\
{} &
\end{tikzcd}
\qquad
\begin{tikzcd}[column sep=huge, row sep=huge]
\cdot \arrow[r, "M \in A\langle 0/y\rangle"] \arrow[d, "A\langle 0/x\rangle \ni N_0"'] & \cdot \arrow[d, "N_1 \in A\langle 1/x\rangle"] \\
\cdot \arrow[r, dashed] & \cdot
\end{tikzcd}$$

The Com Diagram

$$\mathsf{com}_{y.A}^{0\rightsquigarrow 1}(M; x = 0 \hookrightarrow y.N_0, x = 1 \hookrightarrow y.N_1) \in A\langle 1/y\rangle \; [\Psi, x]$$

$$\begin{tikzcd}
 \arrow[r, "\mathrm{x}"] \arrow[d, "\mathrm{y}"'] & {} & & & \\
{} & & \cdot \arrow[rr, "M \in A\langle 0/y\rangle"] \arrow[dd, "A\langle 0/x\rangle \ni N_0"'] & & \cdot \arrow[dd, "N_1 \in A\langle 1/x\rangle"] \\
& & & & \\
& & \cdot \arrow[rr, dashed] & & \cdot \\
& \cdot \arrow[ur, "\mathsf{coe}_{y.A\langle 0/x\rangle}^{y\rightsquigarrow 1}(N_0)"] \arrow[rr, "\mathsf{coe}_{y.A}^{0\rightsquigarrow 1}(M)"'] & & \cdot \arrow[ur, "\mathsf{coe}_{y.A\langle 1/x\rangle}^{y\rightsquigarrow 1}(N_1)"'] &
\end{tikzcd}
\quad \in A\langle 1/y\rangle$$