Computational Higher Type Theory (CHiTT)

Robert Harper

Carlo Angiuli, Evan Cavallo, Favonia Mlatus
Anders Mörtberg, Jon Sterling

MURI Meeting March 23, 2018

Acknowledgements

Thanks to many, including

- Collaborators: Guillaume Brunerie, Dan Licata, Todd Wilson.
- Colleagues: Steve Awodey, Marc Bezem, Thierry Coquand, Simon Huber.
- Inspiration: Robert Constable, Per Martin-Löf, Dana Scott, Vladimir Voevodsky†.

Supported by AFOSR MURI FA9550-15-1-0053, Tristan Nguyen, PM.

Extension of Martin-Löf's Intensional Type Theory (ITT):

- Univalence: $\mathsf{idToEquiv} : \mathsf{Id}_{\mathcal{U}}(A, B) \to \mathsf{Equiv}(A, B)$ is an equivalence.
- Higher Inductive Types: Spheres, truncations, suspensions, ...

Extension of Martin-Löf's Intensional Type Theory (ITT):

- Univalence: idToEquiv : $\mathrm{Id}_{\mathcal{U}}(A, B) \to \mathrm{Equiv}(A, B)$ is an equivalence.
- Higher Inductive Types: Spheres, truncations, suspensions, ...

Rich mathematical content, but what is its computational content?

- $\mathsf{J}_{a,b,c.C}[a.Q](\mathrm{refl}_A(M)) \equiv Q[M/a]$.
- $\mathsf{J}_{a,b,c.C}[a.Q](\mathrm{ua}(A, B, E)) \equiv \mathbf{???}$.
- $\mathsf{J}_{a,b,c.C}[a.Q](\mathrm{loop}) \equiv \mathbf{???}$.

Basic conflict: type-dependent vs. type-independent identifications.

Two Central Ideas

Cubical accounts of higher-dimensional structure in type theory.

- Bezem, Coquand, Huber: face maps, degeneracies.
- Licata, Brunerie; Coquand: diagonals (Cartesian cubes).

Two Central Ideas

Cubical accounts of higher-dimensional structure in type theory.

- Bezem, Coquand, Huber: face maps, degeneracies.
- Licata, Brunerie; Coquand: diagonals (Cartesian cubes).

Uniform Kan condition.

- Computationally natural (literally and figuratively).
- Avoids difficulties with deciding degeneracy.

Two Central Ideas

Cubical accounts of higher-dimensional structure in type theory.

- Bezem, Coquand, Huber: face maps, degeneracies.
- Licata, Brunerie; Coquand: diagonals (Cartesian cubes).

Uniform Kan condition.

- Computationally natural (literally and figuratively).
- Avoids difficulties with deciding degeneracy.

These ideas figure into all current cubical type theories!

- CCHM: DeMorgan algebra of cubes.
- AFH, ABCFHL, AM: Cartesian cubes.

Computational Type Theory

Martin-Löf CMCP, Constable, et al. NuPRL:

> *A sufficiently powerful typed programming language is a setting for constructive mathematics . . .*

Computational Type Theory

Martin-Löf CMCP, Constable, et al. NuPRL:

> *A sufficiently powerful typed programming language is a setting for constructive mathematics . . .*
>
> *. . . and no programming language should be insufficiently powerful!*

Computational Type Theory

Martin-Löf CMCP, Constable, et al. NuPRL:

> *A sufficiently powerful typed programming language is a setting for constructive mathematics . . .*
>
> *. . . and no programming language should be insufficiently powerful!*

Here "constructive" means having computational meaning, rather than (just) enjoying axiomatic freedom.

Computational Type Theory

Martin-Löf CMCP, Constable, et al. NuPRL:

> *A sufficiently powerful typed programming language is a setting for constructive mathematics . . .*
>
> *. . . and no programming language should be insufficiently powerful!*

Here "constructive" means having computational meaning, rather than (just) enjoying axiomatic freedom.

Our goal is to account for higher-dimensional structure in a programming context.

- Supports proving and programming.
- Amenable to practical implementation.

Computational Type Theory

Types are specifications of program execution behavior.

- Computation is prior to typing.
- Types are certain programs.

Computational Type Theory

Types are specifications of program execution behavior.

- Computation is prior to typing.
- Types are certain programs.

Computation is defined by a *transition system*:

- A program is canonical: M val.
- A program may be simplified: $M \longmapsto M'$.

Computational Type Theory

Behavior is defined by (exact) equality of closed types and elements:

- $A \doteq B$ type: A and B behave equally as types (specify the same programs).
- $M \doteq N \in A$: M and N behave equally according to A.

(Types and members defined reflexively.)

Computational Type Theory

Behavior is defined by (exact) equality of closed types and elements:

- $A \doteq B$ type: A and B behave equally as types (specify the same programs).
- $M \doteq N \in A$: M and N behave equally according to A.

(Types and members defined reflexively.)

Open terms behave extensionally as functions of their free variables:

- $a : A \gg B \doteq B'$ type iff $M \doteq M' \in A$ implies $B[M/a] \doteq B'[M'/a]$ type.
- $a : A \gg N \doteq N' \in B$ iff $M \doteq M' \in A$ implies $N[M/a] \doteq N'[M'/a] \in B[M/a]$.

Computational Type Theory

Behavior is defined by (exact) equality of closed types and elements:

- $A \doteq B$ type: A and B behave equally as types (specify the same programs).
- $M \doteq N \in A$: M and N behave equally according to A.

(Types and members defined reflexively.)

Open terms behave extensionally as functions of their free variables:

- $a : A \gg B \doteq B'$ type iff $M \doteq M' \in A$ implies $B[M/a] \doteq B'[M'/a]$ type.
- $a : A \gg N \doteq N' \in B$ iff $M \doteq M' \in A$ implies $N[M/a] \doteq N'[M'/a] \in B[M/a]$.

A computational type system is a precisely defined mathematical object with respect to which types exactly satisfy their universal properties.

Proof Theory and Truth Theory

Equality judgments define truth conditions, not proof conditions.

- Judgments are synthetic, not analytic.
- Are not defined by axioms and rules.
- Do not express "type checking".

Proof Theory and Truth Theory

Equality judgments define truth conditions, not proof conditions.

- Judgments are synthetic, not analytic.
- Are not defined by axioms and rules.
- Do not express "type checking".

A proof theory defines an *analytic* approximation to the truth.

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

Proof Theory and Truth Theory

Equality judgments define truth conditions, not proof conditions.

- Judgments are synthetic, not analytic.
- Are not defined by axioms and rules.
- Do not express "type checking".

A proof theory defines an *analytic* approximation to the truth.

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

Fundamental theorem, aka extraction theorem, via type erasure:

If $\Gamma \vdash M : A$, *then* $|\Gamma| \gg |M| \in |A|$, *etc.*

Extending to Higher Dimensions

Following LB and C, use Cartesian cubical structure given by:

- Contexts $\Psi = x_1, \ldots, x_n$ with $n \geq 0$.
- Dimensions r, either 0, 1, or x.
- Substitutions $\psi : \Psi' \to \Psi$, including weakening, contraction, exchange.

Extending to Higher Dimensions

Following LB and C, use Cartesian cubical structure given by:

- Contexts $\Psi = x_1, \ldots, x_n$ with $n \geq 0$.
- Dimensions r, either 0, 1, or x.
- Substitutions $\psi : \Psi' \to \Psi$, including weakening, contraction, exchange.

Employ a cubical programming language for each dimension context, Ψ:

- Values: $M\ \mathsf{val}_\Psi$.
- Transitions: $M \longmapsto_\Psi M'$.

Extending to Higher Dimensions

Transitions are often dimension-sensitive:

- $\mathsf{seg}_x\langle 0/x\rangle = \mathsf{seg}_0 \longmapsto_\Psi 0$.
- $\mathsf{seg}_x\langle 1/x\rangle = \mathsf{seg}_1 \longmapsto_\Psi 1$.
- $\mathsf{hcom}_{\mathbb{C}}^{r \rightsquigarrow r}(M; \vec{T}) \longmapsto_\Psi M$.
- $\mathsf{hcom}_{\mathbb{C}}^{0 \rightsquigarrow 1}(M; \vec{T})\ \mathsf{val}_\Psi$.

Extending to Higher Dimensions

Transitions are often dimension-sensitive:

- $\mathsf{seg}_x\langle 0/x\rangle = \mathsf{seg}_0 \longmapsto_\Psi 0$.
- $\mathsf{seg}_x\langle 1/x\rangle = \mathsf{seg}_1 \longmapsto_\Psi 1$.
- $\mathsf{hcom}_{\mathbb{C}}^{r\rightsquigarrow r'}(M; \vec{T}) \longmapsto_\Psi M$.
- $\mathsf{hcom}_{\mathbb{C}}^{0\rightsquigarrow 1}(M; \vec{T})\ \mathsf{val}_\Psi$.

But is not necessarily stable under dimension substitution!

- Computation is untyped, so of course it's not.
- Coherence is a property of types and their members.

Cartesian Cubical Type System

The type and member judgments are ramified by dimension:

- $\Gamma \gg A \doteq B \text{ type}_{\kappa}\ [\Psi]$ (κ is a kind).
- $\Gamma \gg M \doteq N \in A\ [\Psi]$

Cartesian Cubical Type System

The type and member judgments are ramified by dimension:

- $\Gamma \gg A \doteq B \text{ type}_\kappa \ [\Psi]$ (κ is a kind).
- $\Gamma \gg M \doteq N \in A \ [\Psi]$

And enjoy the expected cubical structure:

- If $\Gamma \gg A \doteq B \text{ type}_\kappa \ [\Psi]$ and $\psi : \Psi' \to \Psi$, then $\Gamma\,\psi \gg A\,\psi \doteq B\,\psi \text{ type}_\kappa \ [\Psi']$.
- If $\Gamma \gg M \doteq N \in A \ [\Psi]$ and $\psi : \Psi' \to \Psi$, then $\Gamma\,\psi \gg M\,\psi \doteq N\,\psi \in A\,\psi \ [\Psi']$.

Cartesian Cubical Type System

The type and member judgments are ramified by dimension:

- $\Gamma \gg A \doteq B$ type_κ $[\Psi]$ (κ is a kind).
- $\Gamma \gg M \doteq N \in A$ $[\Psi]$

And enjoy the expected cubical structure:

- If $\Gamma \gg A \doteq B$ type_κ $[\Psi]$ and $\psi : \Psi' \to \Psi$, then $\Gamma\,\psi \gg A\,\psi \doteq B\,\psi$ type_κ $[\Psi']$.
- If $\Gamma \gg M \doteq N \in A$ $[\Psi]$ and $\psi : \Psi' \to \Psi$, then $\Gamma\,\psi \gg M\,\psi \doteq N\,\psi \in A\,\psi$ $[\Psi']$.

Coherence is demanded of well-formed types and elements!

- Roughly, evaluation commutes with dimension substitution = cubical aspects.
- E.g., computing a face, then computing another face is exactly the same as computing the "double" face.

Cartesian Cubical Type System

CHiTT is a two-level type theory in the style of VV's HTS.

- Equality is distinguished from identification.
- Equality is extensional.

Cartesian Cubical Type System

CHiTT is a two-level type theory in the style of VV's HTS.

- Equality is distinguished from identification.
- Equality is extensional.

Type constructors enjoy their full universal properties.

- E.g., $a : \mathsf{bool} \gg M \doteq \mathsf{if}(a; M[\mathsf{true}/a]; M[\mathsf{false}/a]) \in A$.
- $\mathrm{Eq}_A(M, N)$ internalizes exact equality $M \doteq N \in A$.

Cartesian Cubical Type System

CHiTT is a two-level type theory in the style of VV's HTS.

- Equality is distinguished from identification.
- Equality is extensional.

Type constructors enjoy their full universal properties.

- E.g., $a : \mathsf{bool} \gg M \doteq \mathsf{if}(a; M[\mathsf{true}/a]; M[\mathsf{false}/a]) \in A$.
- $\mathsf{Eq}_A(M, N)$ internalizes exact equality $M \doteq N \in A$.

Kan composition admits diagonal constraints.

- Necessary to support $r \rightsquigarrow r$ composition.
- Ensures coherence of univalence and composition in the multiverse.

Cartesian Cubical Type System

Universes are defined using induction-recursion, as in NuPRL.

- Univalence types $\mathsf{V}_x(A, B, E)$.
- Kan composition types $\mathsf{fcom}^{r \rightsquigarrow r'}(A; \vec{T})$, where $r \neq r'$.

Cartesian Cubical Type System

Universes are defined using induction-recursion, as in NuPRL.

- Univalence types $\mathsf{V}_x(A, B, E)$.
- Kan composition types $\mathsf{fcom}^{r \rightsquigarrow r'}(A; \vec{T})$, where $r \neq r'$.

A rich class of higher inductive types and families.

- An inductive type specification formalism.
- Jdentity types $\mathsf{Id}_A(M, N)$ as least reflexive family.

Cartesian Cubical Type System

Universes are defined using induction-recursion, as in NuPRL.

- Univalence types $\mathsf{V}_x(A, B, E)$.
- Kan composition types $\mathsf{fcom}^{r \rightsquigarrow r'}(A; \vec{T})$, where $r \neq r'$.

A rich class of higher inductive types and families.

- An inductive type specification formalism.
- Jdentity types $\mathsf{Id}_A(M, N)$ as least reflexive family.

Path types internalize paths at any dimension.

- Points of $\mathsf{Path}_{x.A}(M, N)$ are lines in A between $M \in A\langle 0/x \rangle$ and $N \in A\langle 1/x \rangle$.
- Supports J up to identification, but not computationally.

Proof Theories for CHiTT

There is no preferred proof theory for CHiTT, but many are compatible with it.

- Necessarily incomplete.
- Chosen according to ergonomics.

Proof Theories for CHiTT

There is no preferred proof theory for CHiTT, but many are compatible with it.

- Necessarily incomplete.
- Chosen according to ergonomics.

CHiTT interprets several different proof theories:

- HoTT: provides canonicity (cf Huber).
- RedPRL: engineered for convenience, changes daily.
- YaccTT: Yet Another Cartesian Cubical Type Theory.
- HTS, without resizing.

Summary

What CHiTT offers:

- A "sufficiently expressive" programming language, i.e. a computational foundation for Cartesian higher type theory.
- A constructive univalent "set theory" suitable for UniMath.
- Computational semantics for various proofs theories.
- A *new* Kan condition for Cartesian cubical types.

Summary

What CHiTT offers:

- A "sufficiently expressive" programming language, i.e. a computational foundation for Cartesian higher type theory.
- A constructive univalent "set theory" suitable for UniMath.
- Computational semantics for various proofs theories.
- A *new* Kan condition for Cartesian cubical types.

Some future directions:

- Relate to CCHM and extensions. VV's homotopy hypothesis?
- Implementation and experimentation. RedPRL is still preliminary.
- Foundation for specification of partial, imperative, concurrent, probabilistic programs?

Agenda

Today's talks:

- Carlo Angiuli: Semantics of CHiTT, esp. Kan conditions, univalence, composition.
- Anders Mörtberg: Inductive types and families in CCHM, and YaccTT.
- Evan Cavallo: Inductive types and families in CHiTT.
- Jon Sterling: RedPRL design and implementation, demo.
- Favonia Mlatus: Kinds of types, semi-simplicial types.
- Mike Shulman; Dan Licata and Mitchell Riley: Adjoint type theory, semantically and syntactically.

Primary References

- 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`.
- Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. "Cubical type theory: a constructive interpretation of the univalence axiom." 21st International Conference on Types for Proofs and Programs (TYPES 2015), 2018.
- Carlo Angiuli, Guillaume Brunerie, Thierry Coquand, Kuen-Bang Hou (Favonia), Robert Harper, and Daniel R. Licata. "Cartesian Cubical Type Theory." To appear, 2018.

Further References

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.

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

Daniel R. Licata and Guillaume Brunerie. A cubical type theory, November 2014. URL `http://dlicata.web.wesleyan.edu/pubs/lb14cubical/lb14cubes-oxford.pdf`. Talk at Oxford Homotopy Type Theory Workshop.

Daniel R. Licata and Robert Harper. Canonicity for 2-dimensional type theory. In *Proceedings of the 39th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '12, 2012.

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.

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

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

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.
- $\mathsf{app}(\lambda a.P, N) \longmapsto P[N/a]$.
- if $M \longmapsto M'$, then $\mathsf{app}(M, N) \longmapsto \mathsf{app}(M', N)$.

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.
- $\mathrm{app}(\lambda a.P, N) \longmapsto P[N/a]$.
- 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

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

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* $\mathsf{app}(M, N) \in B[N/a]$.

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 (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:

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

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

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

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

Exact Equality

Martin-Löf

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

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