Guarded Cubical Type Theory:
Path Equality for Guarded Recursion

Birkedal, Bizjak, Clouston
Grathwohl, Spitters and Vezzosi

Aarhus University
Bishop’s numerical language and type theory

Bishop: constructive mathematics as a language for numerical computations
State of the art: type theory based proof assistants
Big library (corn/math-classes) for verified exact analysis in Coq:
Reals, metric spaces, simple ODE solver which actually compute in type theory (with O’Connor, then ForMath)
Want better support/semantics for:
(co)inductive definitions, quotients, transport of structures, ...
GCTT is also a step in that direction
Also: need better algorithms (MAP16)
Verifying huge computational proofs in type theory is feasible
Two motivations for guarded cubical type theory:

- Path equality for guarded dependent types
  application in computer science
- Add guarded recursion to cubical type theory.
Univalent type theory

New foundation for (constructive) maths based on homotopy types
Extends set theoretic foundation
Analogy between setoids (≡ Bishop sets) and 0-types
Type of points and at most one proof that they are equal
Richman’s families of Bishop sets
Families of setoids correspond to maps $A/ = \rightarrow SET/ \simeq$ in the
Hofmann Streicher groupoid model
HS has univalence (isomorphic types may be identified)
Adding more universes leads to $\infty$-groupoids (≡ homotopy types)
These can be modeled by simplicial sets
sSet model of univalent type theory Voevodsky
Univalent type theory

sSet model is classical, no computation
Coquand: constructive cubical model of univalent type theory
Cubical type theory and type checker
Cubical type theory

Idea: equalities are paths, maps from abstract interval to the type
Abstract interval \( \mathbb{I} \) is modeled by a DeMorgan algebra
distributive lattice with involution sat DeMorgan laws
Involution: \( (1 - r) \) Paths between paths are squares, etc
Cubical type theory

\[
\begin{align*}
\Gamma, \Delta &::= \; () \mid \Gamma, x : A & \text{Contexts} \\
t, u, A, B &::= x \mid \lambda x : A.t \mid t u \mid (x : A) \to B & \Pi\text{-types} \\
&\quad | (t, u) \mid t.1 \mid t.2 \mid (x : A) \times B & \Sigma\text{-types} \\
&\quad | 0 \mid s t \mid \text{natrec } t u \mid N & \text{Naturals} \\
&\quad | U & \text{Universe} \\
\end{align*}
\]

Interval \( \mathbb{I} \) (context, not a type)

\[
\begin{align*}
r, s &::= 0 \mid 1 \mid i \mid 1 - r \mid r \land s \mid r \lor s. \\
\Gamma, \Delta &::= \cdots \mid \Gamma, i : \mathbb{I}. \\
\end{align*}
\]
Path types

$$
\frac{
\Gamma \vdash A \quad \Gamma \vdash t : A \quad \Gamma \vdash u : A
}{\Gamma \vdash \text{Path } A \ t \ u}
$$

$$
\frac{
\Gamma \vdash A \quad \Gamma, i : \Pi \vdash t : A
}{\Gamma \vdash \langle i \rangle \ t : \text{Path } A \ t[0/i] \ t[1/i]}
$$

$$
\frac{
\Gamma \vdash t : \text{Path } A \ u \ s \quad \Gamma \vdash r : \Pi
}{\Gamma \vdash t \ r : A}
$$

**Figure:** Typing rules for path types.
Funext for Path

Proof term for functional extensionality:

\[ \text{funext } f \ g \triangleq \lambda p. \langle i \rangle \lambda x. \ p \ x \ i \ : \]
\[ ((x : A) \to \text{Path } B \ (f \ x) \ (g \ x)) \to \text{Path } (A \to B) \ f \ g. \]
Face lattice

Free distributive lattice on the symbols \((i = 0)\) and \((i = 1)\) for all names \(i\), quotiented by the relation \((i = 0) \land (i = 1) = 0_F\).

\[
\phi, \psi ::= 0_F \mid 1_F \mid (i = 0) \mid (i = 1) \mid \phi \land \psi \mid \phi \lor \psi.
\]

Restriction of a context to a face:

\[
\Gamma, \Delta ::= \cdots \mid \Gamma, \phi.
\]

For example, \(\Gamma, \phi \vdash \psi_1 = \psi_2 : F\) is equivalent to
\(\Gamma \vdash \phi \land \psi_1 = \phi \land \psi_2 : F\)
\(\Gamma \vdash t : A[\phi \mapsto u]\) abbreviates \(\Gamma \vdash t : A\) and \(\Gamma, \phi \vdash t = u : A\)
Composition

\[ \Gamma \vdash \varphi : \mathbb{I} \]
\[ \Gamma, i : \mathbb{I} \vdash A \quad \Gamma, \varphi, i : \mathbb{I} \vdash u : A \quad \Gamma \vdash a_0 : A[0/i][\varphi \mapsto u[0/i]] \]
\[ \Gamma \vdash \text{comp}^i A[\varphi \mapsto u] a_0 : A[1/i][\varphi \mapsto u[1/i]] \]

Transport operation for Path types

\[ \text{trans}^i A a \triangleq \text{comp}^i A[0_{\mathbb{I}} \mapsto []] a : A[1/i] \]

where \( a \) has type \( A[0/i] \).

Example of the use of systems is a proof that Path is transitive; given \( p : \text{Path} A a b \) and \( q : \text{Path} A b c \) we can define

\[ \text{trans} p q \triangleq \langle i \rangle \text{comp}^i A [(i = 0) \mapsto a, (i = 1) \mapsto q j] (p_i) : \text{Path} A a c \]

This builds a path between the appropriate endpoints because:

\[ \text{comp}^i A[1_{\mathbb{I}} \mapsto a] (p0) = a \]
\[ \text{comp}^i A[1_{\mathbb{I}} \mapsto q j] (p1) = q1 = c. \]
Cubical type theory

Also: Glue, universe, ...
CTT is an extension of MLTT with functional extensionality and univalence.
(Precise: For strict $J$ one uses a modified equality $Id$ (Swan))
Guarded recursion

A way of defining infinite objects using self-reference
E.g. streams
New type former \( \Rightarrow \) (‘later’, data available tomorrow)

\[
\text{Str}_A = A \times \Rightarrow \text{Str}_A
\]

\( \text{fix} : (\Rightarrow A \rightarrow A) \rightarrow A \) for solving domain equations
Also used to model program logics (concurrency, …)
guarded recursion (GDTT) modeled in $\hat{\omega}$ (topos of trees).

$$(\therefore X)(n) = \begin{cases} \{\star\} & \text{if } n = 0 \\ X(m) & \text{if } n = m + 1 \end{cases}$$

GDTT is an extensional type theory.
Want computation, so intensional variant.
Need a computational interpretation for the proof that bisimular streams are equal. More generally:

$$(\therefore \text{Id } A \ t \ u \rightarrow \text{Id } (\therefore A) (\text{next } t) (\text{next } u)$$

Compare: funext

$$(x : A) \rightarrow \text{Id } B (fx) (gx) \rightarrow \text{Id } (A \rightarrow B) f g$$
Later

\[ \begin{align*}
\Gamma \\ \\
\vdash \cdot : \Gamma \rightarrow \cdot \\
\vdash \xi : \Gamma \rightarrow \Gamma' \\
\vdash t : \triangleright \xi . A
\end{align*} \]

\[ \vdash \xi [x \leftarrow t] : \Gamma \rightarrow \Gamma', x : A \]

**Figure:** Formation rules for delayed substitutions.

Do notation, applicative functor

\[ \begin{align*}
\Gamma, \Gamma' & \vdash A \\
\vdash \xi : \Gamma \rightarrow \Gamma' \\
\Gamma & \vdash \triangleright \xi . A \\
\Gamma, \Gamma' & \vdash A : U \\
\vdash \xi : \Gamma \rightarrow \Gamma' \\
\Gamma & \vdash \triangleright \xi . A : U \\
\Gamma, \Gamma' & \vdash t : A \\
\vdash \xi : \Gamma \rightarrow \Gamma' \\
\Gamma & \vdash \text{next} \ \xi \cdot t : \triangleright \xi . A
\end{align*} \]

**Figure:** Typing rules for later types.
\[\vdash \xi [x \leftarrow t] : \Gamma \rightarrow \Gamma', x : B \quad \Gamma, \Gamma' \vdash A\]

\[\Gamma \vdash \triangleright \xi [x \leftarrow t] . A \equiv \triangleright \xi . A\]

\[\vdash \xi [x \leftarrow t, y \leftarrow u] \xi' : \Gamma \rightarrow \Gamma', x : B, y : C, \Gamma'' \quad \Gamma, \Gamma' \vdash C \quad \Gamma, \Gamma', x : B, y : C, \Gamma'' \vdash A\]

\[\Gamma \vdash \triangleright \xi [x \leftarrow t, y \leftarrow u] \xi'. A \equiv \triangleright \xi [y \leftarrow u, x \leftarrow t] \xi'. A\]

\[\vdash \xi : \Gamma \rightarrow \Gamma' \quad \Gamma, \Gamma', x : B \vdash A \quad \Gamma, \Gamma' \vdash t : B\]

\[\Gamma \vdash \triangleright \xi [x \leftarrow \text{next } \xi. t] . A \equiv \triangleright \xi . A[t/x]\]

**Figure:** Type equality rules for later types
\[
\begin{align*}
\Gamma, \Gamma' \vdash u : A & \quad \Gamma \vdash \text{next } \xi [x \leftarrow t] . u = \text{next } \xi . u : \xi . A \\
\Gamma, \Gamma' \vdash C & \quad \Gamma \vdash \text{next } \xi [x \leftarrow t, y \leftarrow u] \xi' . v = \text{next } \xi [y \leftarrow u, x \leftarrow t] \xi' . v \vdash \xi [x \leftarrow t, y \leftarrow u] \xi'. A \\
\Gamma \vdash u : A & \quad \Gamma \vdash \text{next } \xi [x \leftarrow \text{next } \xi . t] . u = \text{next } \xi . u[t/x] : \xi . A[t/x] \\
\Gamma \vdash t : \xi . A & \quad \Gamma \vdash \text{next } \xi [x \leftarrow t] . x = t : \xi . A
\end{align*}
\]

Figure: Term equality rules for later types.
Example

Similar to funext, we now have in GCTT

\[ \lambda p. \langle i \rangle \text{next } \xi[p' \leftarrow p]. p' i : \]

\[ \Rightarrow \xi. \text{Path } A t u \rightarrow \text{Path } (\Rightarrow \xi. A)(\text{next } \xi. t)(\text{next } \xi. u). \]

This improves on the equality reflection that was needed before.
Cor: bisimilar streams are equal
Fixpoints

In GDTT: \( \text{fix } x. t = t[\text{next fix } x. t/x] \)
(breaks decidable type checking).
Instead, we have a delay fixed point (dfix).
A path from the fixed point \( \text{(dfix}^0 \text{)} \) to its unfolding \( \text{(dfix}^1 \text{)} \).

\[
\Gamma \vdash r : \Pi \quad \Gamma, x : \topA \vdash t : A \\
\quad \frac{} \quad \Gamma \vdash \text{dfix}^r x. t : \topA
\]

\[
\Gamma, x : \topA \vdash t : A \\
\quad \frac{} \quad \Gamma \vdash \text{dfix}^1 x. t = \text{next } t[\text{dfix}^0 x. t/x] : \topA
\]

Proposition (Unique guarded fixed points)

Any guarded fixed-point of \( f : \topA \to A \) is path equal to \( \text{fix } x. f x \).
Examples

- If \( f : A \to A \to B \) is commutative, then \( \text{zipWith } f : \text{Str}_A \to \text{Str}_A \to \text{Str}_B \) is commutative.

- Let

\[
\begin{align*}
\text{Rec}_A & \triangleq \text{fix } x.(\triangleright[x' \leftarrow x].x') \to A \\
\Delta & \triangleq \lambda x.f(\text{next}[x' \leftarrow x].((\text{unfold } x')x)) : \triangleright \text{Rec}_A \to A \\
Y & \triangleq \lambda f.\Delta(\text{next fold } \Delta) : (\triangleright A \to A) \to A
\end{align*}
\]

where fold and unfold are the transports along the path between \( \text{Rec}_A \) and \( \triangleright \text{Rec}_A \to A \).

\( Y \) is a guarded fixed-point combinator.
Semantics of GCTT

Intuitively, cubical model in the topos of trees. (iterated forcing)
Existing theory does not directly work due to strictness and universes. A more concrete construction. Semantics in $\square \times \omega$.

$\square$ is the opposite of the Kleisli category of the free De Morgan algebra monad on finite sets. (=Lawvere theory of De Morgan algebras).

More concretely, given a countably infinite set of names $i, j, k, \ldots$, $C$ has as objects finite sets of names $I, J$. A morphism $I \to J \in C$ is a function $J \to \text{DM}(I)$, where $\text{DM}(I)$ is the free De Morgan algebra with generators $I$. 
Semantics of GCTT

Theorem

A presheaf topos with a non-trivial \((0 \neq 1)\) internal DeMorgan algebra with the disjunction property \((a \vee b = 1 \vdash a = 1 \vee b = 1)\) models CTT (without universe and gluing).

In particular, \(\mathcal{C} \times \mathbb{C}\) for any category \(\mathbb{C}\) models CTT.

Hence \(\square \times \omega\) models CTT.

\(\triangleright\) can be defined explicitly.

\[\triangleright (X)(l, n) \begin{cases} \{ * \} & \text{if } n = 0 \\ X(l, m) & \text{if } n = m + 1 \end{cases}\]

Key observation: \(\triangleright\) preserves fibrancy.

Conclusion: \(\square \times \omega\) models GCTT.
Glue and universe

If the presheaf topos also has a fibrant universe and $\forall : \mathbb{F}^\Pi \to \mathbb{F}$, then we can model the full CTT.
In particular, $\mathcal{C} \times \mathcal{C}$ for any category $\mathcal{C}$ with an initial object can be used to give semantics to the entire cubical type theory.
Implementation

Prototype build on top of cubical
Hope to integrate into agda
Conclusions

- New type theory GCTT with a model in $\mathcal{C} \times \omega$
  Path equality for guarded dependent type theory
  (Application of HoTT to CS)
- Adding guarded recursion to cubical type theory
- Axiomatic treatment of the cubical model using the internal logic. ‘new’ class of models.

TODO:
canonicity of GCTT(from CTT)