Higher Inductive Types: The circle and friends, axiomatically

Peter Lumsdaine

Dalhousie University
Halifax, Nova Scotia

Foundational Methods in Computer Science
Kananaskis, June 2011
Dependent Type Theory (Martin-Löf, Calculus of Constructions, etc.): highly expressive constructive theory, potential foundation for maths.

Central concept: *terms of types*.

\[ \vdash \mathbb{N} \text{ type} \quad \text{Nat : Type} \]
\[ \vdash 0 : \mathbb{N} \quad 0 : \text{Nat} \]

(M-L notation) (pseudo-Coq syntax)

Both can be *dependent* on (typed) variables:

\[ n : \text{Nat} \vdash \mathbb{R}^n \text{ type} \]
\[ \text{Real}_\text{Vec} (n:\text{Nat}) : \text{Type} \]
DTT

Terms of dependent types:

\[
\begin{align*}
  n : \mathbb{N} \vdash 0_n : \mathbb{R}^n \\
  \vdash 0 : \prod_n \mathbb{R}^n \\

  \text{poly_zero} \ (n: \text{Nat}) : \text{Real_Vec} \ n \\
  \text{poly_zero} : \text{forall} \ (n: \text{Nat}) , \text{Real_Vec} \ n
\end{align*}
\]

Original intended interpretation: **Sets**. Types are sets; terms are elements of sets.

Dependent type over \( X \):

\[
X \xrightarrow{\gamma} \text{Sets} \quad \text{or} \quad \gamma = \sum_{i \in X} \gamma_i
\]

\[
X
\]

3 / 20
Logic within dependent type theory: Curry-Howard.

Euclid : forall (n: Nat), exists (p: Nat),
        (p > n) \ (isPrime p).

A predicate on X : Type is represented as a dependent type P : X -> Type.

(In classical set model, P(x) will be 1 or 0, depending on whether P holds at x.)
Homotopy Type Theory

Predicate representing equality/identity:

\[ x, y : A \vdash \text{Id}_A(x, y) : \text{Type} \]  
\[ \text{Id}(x : A, y : A) : \text{Type} \]

\[ \text{isPrime}(n : \text{Nat}) : \text{Type} \]

\[ := \neg (\text{Id} \ n \ 1) \ \land \]

\[ \forall d : \text{Nat}, (d \text{ divides} n) \rightarrow (\text{Id} \ d \ 1) \lor (\text{Id} \ d \ n). \]

Has clear, elegant axioms, and excellent computational behaviour. Can one prove it represents a proposition, i.e. any two terms \( p, q : \text{Id} \ x \ y \) are equal?
Homotopy Type Theory

Predicate representing equality/identity:

\[ x, y : A \vdash \text{Id}_A(x, y) \text{ type} \quad \text{Id} \ (x \ y : A) : \text{Type} \]

\[ \text{isPrime} \ (n : \text{Nat}) : \text{Type} \]

\[ := \sim(\text{Id} \ n \ 1) \ \text{\slash/} \]

\[ \quad \text{forall d : Nat, } (d \text{ divides n) } \rightarrow \]

\[ \quad \ (\text{Id} \ d \ 1) \ \text{\slash/} \ (\text{Id} \ d \ n). \]

Has clear, elegant axioms, and excellent computational behaviour. Can one prove it represents a proposition, i.e. any two terms \( p \ q : \text{Id} \ x \ y \) are equal?

"Problem". No! (Hofmann-Streicher groupoid model, 1995.)

Why is this a problem?
Homotopy Type Theory

Problem: a mismatch! Original conception: a theory of something like sets. Formulation largely motivated by computational behaviour, constructive philosophy. Types of the theory end up not behaving like familiar classical sets.
Homotopy Type Theory

Problem: a mismatch! Original conception: a theory of something like sets. Formulation largely motivated by computational behaviour, constructive philosophy. Types of the theory end up not behaving like familiar classical sets.

One solution: add more axioms — “equality reflection”, etc. Problem: destroys computational content, makes typechecking undecidable, etc.

Precise statements: models of the theory in $\text{Top}$, $\text{SSet}$, $n\text{-Gpd}$, etc. Conversely, higher categories, wfs's, etc. from theory (Garner, Gambino, van den Berg, etc.).
Homotopy Type Theory

Problem: a mismatch! Original conception: a theory of something like sets. Formulation largely motivated by computational behaviour, constructive philosophy. Types of the theory end up not behaving like familiar classical sets.

One solution: add more axioms — “equality reflection”, etc. Problem: destroys computational content, makes typechecking undecidable, etc.

Alternative: see types as being something more like spaces — topological spaces, (higher) groupoids, etc. Change our idea of what this is a theory of.

Precise statements: models of the theory in Top, SSet, $n$-Gpd, nice Quillen model categories... (Awodey, Warren, Garner, van en Berg, etc.); conversely, higher categories, wfs’s, etc. from theory (Garner, Gambino, van den Berg, PLL).
Homotopy Type Theory

Idea: work with dependent type theory as a theory of homotopy types.

\(\text{Id} \ x \ y\) not just proposition of “equality”, but *space of paths* from \(x\) to \(y\).

Notation: write \(x \sim \sim > x'\) for \(\text{Id} \ A \ x \ x'\).

Dep. type \(Y : X \to \text{Type}\) — a fibration

\[
\begin{array}{c}
Y \\
p \\
\downarrow \\
X \\
\end{array}
\]

Term \(f : \text{forall } x : X, (Y x)\) — a section \(f \left(\begin{array}{c} Y \\ X \end{array}\right) p\).
Programme (Voevodsky et al): develop homotopy theory axiomatically within this logic.

So far, enough to start making definitions: contractibility, loop spaces, equivalence...

But: how to start building interesting spaces? Circles, spheres, … ?
Inductive types

Main standard type-construction principle: *inductive types*.

Inductive Nat : Type where
  | zero : Nat
  | suc : Nat -> Nat.

“Let \( \text{Nat} \) be the type freely generated by an element \( \text{zero} : \text{Nat} \) and a map \( \text{suc} : \text{Nat} \to \text{Nat} \).”

From this specification, Coq automatically generates *induction principle* (aka *recursor, eliminator*) for \( \text{Nat} \):

\[
\text{forall} \ (P : \text{Nat} \to \text{Type}) \\
\quad \text{(d_zero : } P \ \text{zero)} \\
\quad \text{(d_suc : forall (n:}\text{Nat}) , \ P \ n \to \ P \ (\text{suc} \ n)), \\
\text{forall (n : } \text{Nat}) , \ P \ n.
\]
Higher Inductive Types

Extend this principle: allow constructors to produce paths.

Inductive Circle : Type where
  | base : Circle
  | loop : base ~ ~> base.

“Let Circle be the type freely generated by an element base : Circle and a path loop : base ~ ~> base.”

Can’t actually type this definition into Coq (yet). What should its induction principle be?
Type of non-dependent eliminator is clear:

forall (X : Type)
    (d_base : X)
    (d_loop : d_base ~> d_base),
Circle -> X

Not powerful enough to do much with. Need to be able to eliminate into dependent type. How about:

forall (P : Circle -> Type)
    (d_base : P base)
    (d_loop : d_base ~> d_base),
forall (x:Circle), P x.
Type of non-dependent eliminator is clear:

$$\forall (X : \text{Type}) \ (d\_base : X) \ (d\_loop : d\_base \rightsquigarrow d\_base), \ \text{Circle} \rightarrow X$$

Not powerful enough to do much with. Need to be able to eliminate into \textit{dependent} type. How about:

$$\forall (P : \text{Circle} \rightarrow \text{Type}) \ (d\_base : P \ base) \ (d\_loop : d\_base \rightsquigarrow d\_base), \ \forall (x : \text{Circle}), \ P \ x.$$
Interval

Digression: axiomatise the interval, as warmup.

Inductive Interval : Type where
| src : Interval
| tgt : Interval
| seg : src ~> tgt.

Induction principle?

Given fibration \( P : \text{Interval} \rightarrow \text{Type} \), how to produce section?

Need points \( d_{\text{src}} : (P \ \text{src}), d_{\text{tgt}} : (P \ \text{tgt}) \), and a path \( d_{\text{seg}} \) between them.
Interval

Digression: axiomatise the interval, as warmup.

Inductive Interval : Type where
  | src : Interval
  | tgt : Interval
  | seg : src ~> tgt.

Induction principle?

Given fibration \( P : \text{Interval} \to \text{Type} \), how to produce section?

Need points \( d_{\text{src}} : (P \text{ src}), d_{\text{tgt}} : (P \text{ tgt}) \), and a path \( d_{\text{seg}} \) between them.

Problem: \( d_{\text{src}} ~> d_{\text{tgt}} \) doesn’t typecheck — \( d_{\text{src}}, d_{\text{tgt}} \) have different types. How to get type for \( d_{\text{seg}} \)?
Answer: transport between fibers of a fibration, derivable in the type theory:

\[
\text{transport } \{X : \text{Type}\} \{P : X \rightarrow \text{Type}\}
\{x \ y : X\} \{u : x \rightsquigarrow y\} \{a : P \ x\}
: P \ y
\]

So, induction principle for interval:

\[
\forall (P : \text{Interval} \rightarrow \text{Type})
\quad (d_{\text{src}} : P \ src) \ (d_{\text{tgt}} : P \ tgt)
\quad (d_{\text{seg}} : (\text{transport} \ \text{seg} \ d_{\text{src}}) \rightsquigarrow d_{\text{tgt}}),
\forall (x : \text{Interval}), \ P \ x.
\]
In induction principle, the case for a constructor of path type should *lie over* that path.

Correct induction principle for the circle:

\[
\text{forall } (P : \text{Circle} \rightarrow \text{Type}) \quad (d\text{\textunderscore base} : P \text{ base}) \quad (d\text{\textunderscore loop} : (\text{transport loop } d\text{\textunderscore base}) \sim \rightarrow d\text{\textunderscore base}), \quad \text{forall } (x:\text{Circle}), \quad P \ x.
\]
Circle

Not a section.

Section!

transport loop $x$

base

loop
Consequences

What can we prove with these?

- Interval is contractible.
Consequences

What can we prove with these?

- Interval is contractible.
- Interval implies functional extensionality.
Consequences

What can we prove with these?

- Interval is contractible.
- Interval implies functional extensionality.
- Circle is contractible iff all path types are trivial (i.e. in a \textbf{Sets}-like model).
Consequences

What can we prove with these?

- Interval is contractible.
- Interval implies functional extensionality.
- Circle is contractible iff all path types are trivial (i.e. in a Sets-like model).
- \(\pi_1(S^1) \cong \mathbb{Z}\). Assuming Univalence (“equality between types is homotopy equivalence”), loop space of Circle is homotopy-equivalent to Int.
Models

Can interpret $\text{Circle}$ (and the other HIT’s below) in:

- **Set**: trivially, 0-truncated.
- **$\text{Gpd}$**: 1-truncated; but with a good enough univalent universe that the above theorem applies.
- **str-$n$-$\text{Gpd}$**, for $n \leq \omega$.

Hopefully also $\text{Sets}^{\Delta^{\text{op}}}$, $\text{Top}$?
More Higher Inductive Types

- Familiar spaces with good cell complex structures: higher spheres, tori, Klein bottle, …
- Maps between these: universal covers, Hopf fibration, …
- Mapping cylinders. From these, wfs’s as for a Quillen model structure.
- Truncations, homotopy groups: $\text{tr}_{-1} = \pi_{-1}$, $\text{tr}_0 = \pi_0$, $\text{tr}_1$, $\pi_1$, …
Tuncations

By using *proper recursion* (like `suc` for `Nat`), can construct *truncations* as higher inductive types:

```haskell
Inductive isInhab (X:Type) : Type where
  | incl : X -> isInhab X
  | contr : forall (y y' : isInhab X),
           y ~~> y'.
```

Gives the support of a type, aka $-1$-truncation $\text{tr}_{-1} = \pi_{-1}$, *homotopy-proposition reflection, bracket types* (Awodey, Bauer).

Gives an alternate “homotopy-proposition” interpretation of logic in the DTT, besides Curry-Howard. So may even have *classical* logic existing inside a completely constructive type theory!
Intrigued?

References, related reading, Coq files, and much more at:
http://homotopytypetheory.org