Towards an Implementation of Higher Observational Type Theory

Michael Shulman

University of San Diego

jww Thorsten Altenkirch, Ambrus Kaposi, and Elif Uskuplu

running HoTT @ NYU Abu Dhabi
20 April 2024

Outline

What is Higher Observational Type Theory?

H.O.T.T. is a third style of homotopy type theory, after Book HoTT and Cubical Type Theory.

- In Book HoTT, identity types are defined uniformly across all types as an inductive family.
- In Cubical Type Theory, identity types are defined uniformly across all types by mapping out of the interval.
- In Higher Observational Type Theory, identity types are defined observationally according to the base type.
  - $\mathrm{Id}_{A\times B}(\langle x_0, y_0\rangle, \langle x_1, y_1\rangle)$ is a product $\mathrm{Id}_A(x_0, x_1) \times \mathrm{Id}_B(y_0, y_1)$.
  - $\mathrm{Id}_{A\to B}(f_0, f_1)$ is $\cancel{(x : A) \to \mathrm{Id}_B(f_0\, x, f_1\, x)}$

    $(x_0\, x_1 : A)(x_2 : \mathrm{Id}_A(x_0, x_1)) \to \mathrm{Id}_B(f_0\, x_0, f_1\, x_1)$
  - $\mathrm{Id}_{\mathcal{U}}(A, B)$ is a type of equivalences $A \simeq B$.

HOTT has natural semantics in semicartesian (BCH) cubical sets.

The primitives of HOTT

1. Any type A has an identity type $\mathsf{Id}_A(x_0, x_1)$, which computes* based on the structure of A.
2. Any term $M : A$ has a reflexivity term $\mathsf{refl}_M : \mathsf{Id}_A(M, M)$, which computes based on the structure of M.
   - $\mathsf{refl}_{\langle a,b\rangle} = \langle \mathsf{refl}_a, \mathsf{refl}_b \rangle$ and $\mathsf{refl}_{\mathsf{fst}\,u} = \mathsf{fst}\,\mathsf{refl}_u$, etc.
   - $\mathsf{refl}_{\lambda x.M} = \lambda x_0\, x_1\, x_2.\, \mathsf{ap}_{x.M}(x_0, x_1, x_2)$, etc.
3. Any open term $x : A \vdash M : B\, x$ has an $\mathsf{ap}_{x.M}(a_0, a_1, a_2)$, for $a_2 : \mathsf{Id}_A(a_0, a_1)$, which computes based on M.
   - $\mathsf{ap}_{x.\langle M,N\rangle}(a_0, a_1, a_2) = \big\langle \mathsf{ap}_{x.M}(a_0, a_1, a_2), \mathsf{ap}_{x.N}(a_0, a_1, a_2)\big\rangle$
   - $\mathsf{ap}_{x.\lambda y.M}(a_0, a_1, a_2) = \lambda y_0\, y_1\, y_2.\, \mathsf{ap}_{(x,y).M}(a_0, a_1, a_2, y_0, y_1, y_2)$
   - $\mathsf{ap}_{x.M\,N}(a_0,a_1,a_2) =$ $$\mathsf{ap}_{x.M}(a_0,a_1,a_2)\; N[x \mapsto a_0]\; N[x \mapsto a_1]\; \mathsf{ap}_{x.N}(a_0,a_1,a_2)$$

   (This is what requires our definition of $\mathsf{Id}_{A\to B}$.)
4. Any square $a_{22} : \mathsf{Id}^{a_{02},a_{12}}_{\mathsf{Id}_A}(a_{20}, a_{21})$ has a symmetry $\mathsf{sym}(a_{22}) : \mathsf{Id}^{a_{20},a_{21}}_{\mathsf{Id}_A}(a_{02}, a_{12})$, which computes based on a_{22}.

From parametric type theory to HOTT

Cubical Type Theory can be obtained by defining a fibrancy predicate in a non-univalent substrate theory (Orton–Pitts).

We intend to obtain HOTT similarly. The rule

$$\mathsf{Id}_{A \to B}(f_0, f_1) \quad \text{is} \quad (x_0\, x_1 : A)(x_2 : \mathsf{Id}_A(x_0, x_1)) \to \mathsf{Id}_B(f_0\, x_0, f_1\, x_1)$$

suggests that the substrate should be internal binary parametricity, where Id is a "bridge type". This satisfies all the same rules as the identity type in HOTT except

- $\mathsf{Id}_{\mathcal{U}}(A, B)$ is a type of correspondences $A \to B \to \mathcal{U}$.

What we want

What we want

1. A proof assistant implementing HOTT!

For that we need...

2. A typechecking algorithm

For that we need (as for any dependent type theory)...

3. An equality-testing algorithm

And for that we need (more or less)...

4. A normalization algorithm (computing with open terms).

Roughly speaking, we test equality by normalizing both terms and comparing normal forms.

What we have

To be presented today

1. A normalization algorithm for a version of "Parametric OTT".
2. An implementation of this algorithm in OCaml, along with a typechecker for a prototype proof assistant called Narya.

NOT being presented today

A proof that this algorithm is correct!

However:

- The algorithm aligns with general principles of NbE.
- The implementation is very strongly typed, so it serves as a partial formalization of correctness.
- Narya has been tested on many examples and seems to work.

Outline

Higher-dimensional structure

The higher structure of HOTT is generated by low-dimensional primitives like "refl" and "sym". But many different such composites produce the same operation.

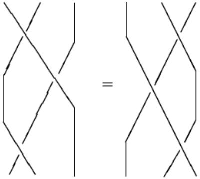

Image credit: John Baez

$$\mathrm{sym}(\mathrm{ap}_{\mathrm{sym}}(\mathrm{sym}(x_{222}))) \equiv \mathrm{ap}_{\mathrm{sym}}(\mathrm{sym}(\mathrm{ap}_{\mathrm{sym}}(x_{222})))$$

A normalization algorithm must implement such equalities.

Our choice

Represent higher dimensions directly internally, evaluating each composite of refl and sym to a cubical operator in canonical form.

The user can still restrict themselves to refl and sym.

Σ-types vs records

The identity type of a Σ-type is “defined to be” another Σ-type:

$$\mathrm{Id}_{\Sigma(x:A).B(x)}(u,v) \;\approx\; \Sigma(p:\mathrm{Id}_A(\pi_1 u,\pi_1 v)).\mathrm{Id}_B^p(\pi_2 u,\pi_2 v)$$

In a proof assistant, Σ-types are just a particular record type:

```
def Σ (A : Type) (B : A → Type) : Type ≔ sig (
  fst : A,
  snd : B fst,
)
```

In general, the identity type of any record type should be another record type, but it can't be an instance of the same record type. And similarly for inductive and coinductive types.

(Non-)computation with types

Our choice

Refrain from computing definitionally with any identity types.

For example `Id (Σ A B) u v` is not definitionally equal to

```
Σ (Id A (u .fst) (v .fst))
    (p ↦ Id B (u .fst) (v .fst) p (u .snd) (v .snd))
```

but instead behaves like a record type defined as

```
sig (
  fst : Id A (u .fst) (v .fst),
  snd : Id B (u .fst) (v .fst) fst (u .snd) (v .snd),
)
```

They are definitionally isomorphic, and their fields and constructors have the same names, so we can usually pretend they are the same. Inductive, coinductive, and even function types are similar.

Outline

Old-style normalization

Old view of normalization

1. Formulate reduction rules such as $(\lambda x.M)\,N \rightsquigarrow M[x \mapsto N]$
2. Prove that applying these reductions to any term eventually leads to a normal form, a term that cannot be further reduced.

However, this is not very efficient. For example:

$$\begin{aligned}(\lambda x.\lambda y.M)\,N\,P &\rightsquigarrow \big((\lambda y.M)[x \mapsto N]\big)\,P\\ &\equiv \big(\lambda y.M[x \mapsto N]\big)\,P\\ &\rightsquigarrow \big(M[x \mapsto N]\big)[y \mapsto P]\end{aligned}$$

We have to traverse the term M (which could be large) twice: once to substitute N for x, then again to substitute P for y.

(Also worry about variable capture, or incrementing De Bruijn indices, etc.)

Towards NbE

First idea

Don't actually compute $(\lambda y.M)[x \mapsto N]$, but keep it as a closure. Then, when it is applied to a further argument P, compute the simultaneous substitution $M[x \mapsto N, y \mapsto P]$.

However, if it never is applied to a further argument, we do have to actually compute it as $\lambda y.\big(M[x \mapsto N]\big)$ to get a normal form.

To track this, and ensure that closures never appear in normal forms, we use two different kinds of terms:

- terms do not contain closures, and use De Bruijn indices.
- values contain closures, and use De Bruijn levels.

(Use of levels/indices eliminates variable capture and index increments.)

Normalization by evaluation

Normalization has two steps:

1. evaluation of a term M into a value, using an environment that assigns a value to every free (index) variable in M.
2. readback of a value into a normalized term.

In particular:

- There is no "substitution" operation: evaluation does it all.
- When readback finds a closure $(\lambda y.M)[x \mapsto N]$, it restarts evaluation with y bound to a variable, $M[x \mapsto N, y \mapsto y]$, then reads back the result and re-wraps it in λy.
- Readback can be type-directed and perform η-expansion.
- If we define the type of values to contain no redexes, we can guarantee statically that the result is a normal form.
- There's a close connection to mathematical proofs by categorical gluing along a restricted Yoneda embedding.

Outline

Matching under binders

In ordinary NbE, matching happens during evaluation.

Example

To evaluate the term "if M then N else P", we first evaluate M to a value and inspect the result. If it is "true", we proceed to evaluate N; if it is "false", we proceed to evaluate P.

However, this style doesn't play well with matching under binders.

Example

To evaluate "$\mathsf{ap}_{x.M}(p_0, p_1, p_2)$", we have to inspect M to implement rules like for pairs:

$$\mathsf{ap}_{x.\langle M,N\rangle}(p_0, p_1, p_2) \equiv \left\langle \mathsf{ap}_{x.M}(p_0, p_1, p_2), \mathsf{ap}_{x.N}(p_0, p_1, p_2) \right\rangle$$

But evaluating $x.M$ produces a closure, not actually computing the body M to anything we can match against!

ap is a form of substitution

"ap" is a lot like substitution:

1. They are never* normal forms: they always reduce away, computing on both introduction and elimination forms.
2. The user doesn't need direct access to them. For "ap", it suffices to use "refl" on a function.

$$\mathsf{ap}_{x.M}(p_0, p_1, p_2) \equiv \mathsf{refl}_{\lambda x.M}\ p_0\ p_1\ p_2$$

3. Their computation rules are similar:

$$\langle M, N\rangle[x \mapsto P] \equiv \big\langle M[x \mapsto P], N[x \mapsto P]\big\rangle$$

Thus, we replace "ap" by a higher-dimensional substitution, which in NbE becomes higher-dimensional evaluation.

Higher-dimensional environments

Definition

An *n-dimensional environment* associates to each (index) variable an n-dimensional cube of values.

$$
\begin{array}{ll}
n = 0 & a : A \\[1em]
n = 1 & a_0 : A,\ a_1 : A,\ a_2 : \mathrm{Id}_A(a_0, a_1) \\[1em]
n = 2 & a_{00} : A,\ a_{01} : A,\ a_{02} : \mathrm{Id}_A(a_{00}, a_{01}), \\
& \quad a_{10} : A,\ a_{11} : A,\ a_{12} : \mathrm{Id}_A(a_{10}, a_{11}), \\
& a_{20} : \mathrm{Id}_A(a_{00}, a_{10}),\ a_{21} : \mathrm{Id}_A(a_{01}, a_{11}), \\
& \qquad\qquad\qquad a_{22} : \mathrm{Id}_{\mathrm{Id}_A}^{a_{02}, a_{12}}(a_{20}, a_{21})
\end{array}
$$

Faces and evaluation

For any k-dimensional face ϕ of an n-dimensional cube, an n-dimensional environment θ has a k-dimensional face environment $\theta * \phi$. E.g. the faces of the 1-dimensional

$$\left[\begin{array}{c} x \mapsto \Big(a_0 : A,\ a_1 : A,\ a_2 : \mathsf{Id}_A(a_0, a_1)\Big), \\ y \mapsto \Big(b_0 : B,\ b_1 : B,\ b_2 : \mathsf{Id}_B(b_0, b_1)\Big) \end{array}\right]$$

are the 0-dimensional $[x \mapsto a_0,\ y \mapsto b_0]$ and $[x \mapsto a_1,\ y \mapsto b_1]$.

Evaluating a term M in an n-dimensional environment θ produces an n-dimensional value $M[\theta]$, whose boundary consists of $M[\theta * \phi]$ for the faces ϕ of n. For example, if $\langle x, y\rangle : A \times B$, then

$$\langle x, y\rangle \Big[x \mapsto (a_0, a_1, a_2), y \mapsto (b_0, b_1, b_2)\Big] \equiv \langle a_2, b_2\rangle$$

which lies in $\mathsf{Id}_{A\times B}(\langle a_0, b_0\rangle, \langle a_1, b_1\rangle)$.

ap is higher evaluation

Now instead of $\mathsf{ap}_{x.M}(a_0, a_1, a_2)$ we have

$$M[x \mapsto (a_0, a_1, a_2)].$$

In particular, the computation rule for reflexivity of an abstraction, which "starts" higher substitution, is

$$\mathsf{refl}_{\lambda x.M} \equiv \lambda x_0\, x_1\, x_2.\, M[x \mapsto (x_0, x_1, x_2)].$$

In NbE, this should be an evaluation rule in some environment θ. But if θ starts out 0-dimensional, we need to evaluate M in a 1-dimensional environment that we can extend by (x_0, x_1, x_2).

$$\mathsf{refl}_{\lambda x.M}[\theta] \equiv \lambda x_0\, x_1\, x_2.\, M[?, x \mapsto (x_0, x_1, x_2)]$$

We need an operation of "degenerate environments".

ap is higher evaluation

Now instead of $\mathsf{ap}_{x.M}(a_0, a_1, a_2)$ we have

$$M[x \mapsto (a_0, a_1, a_2)].$$

In particular, the computation rule for reflexivity of an abstraction, which "starts" higher substitution, is

$$\mathsf{refl}_{\lambda x.M} \equiv \lambda x_0\, x_1\, x_2.\, M[x \mapsto (x_0, x_1, x_2)].$$

In NbE, this should be an evaluation rule in some environment θ. But if θ starts out 0-dimensional, we need to evaluate M in a 1-dimensional environment that we can extend by (x_0, x_1, x_2).

$$\mathsf{refl}_{\lambda x.M}[\theta] \equiv \lambda x_0\, x_1\, x_2.\, M[\mathsf{refl}_\theta, x \mapsto (x_0, x_1, x_2)]$$

We need an operation of "degenerate environments".

Degeneracies

Any m-dimensional degeneracy δ of an n-dimensional cube maps an n-dimensional object M to an m-dimensional one $M\langle\delta\rangle$. E.g.

$$\mathsf{refl}\, M \equiv M\langle\rho\rangle \qquad \mathsf{sym}\, M \equiv M\langle\sigma\rangle$$

Like substitution/evaluation, $M\langle\delta\rangle$ is defined by traversing M.
But unlike evaluation, both M and $M\langle\delta\rangle$ are values.
This is necessary to evaluate degeneracies:

$$(\mathsf{refl}\, x)[x \mapsto M] \equiv M\langle\rho\rangle$$

where M, being in an environment, is a value.

(NB: For afficionados of modal type theory, $\theta * \phi$ and $M\langle\delta\rangle$ may remind you of locks and keys.)

Degenerate environments

An m-dimensional degeneracy δ of an n-dimensional cube also maps any n-dimensional environment θ to a degenerate environment $\theta * \delta$. For instance, $[x \mapsto a, y \mapsto b] * \rho$ (reflexivity) is

$$\left[\begin{array}{c} x \mapsto \Big(a : A,\ a : A,\ \mathsf{refl}_a : \mathsf{Id}_A(a, a)\Big), \\ y \mapsto \Big(b : B,\ b : B,\ \mathsf{refl}_b : \mathsf{Id}_B(b, b)\Big) \end{array}\right]$$

This is how we evaluate degeneracies in general:

$$(M\langle\delta\rangle)[\theta] \equiv M[\theta * \delta].$$

And act on closures by degeneracies:

$$\big((\lambda y.M)[\theta]\big)\langle\delta\rangle \equiv (\lambda y.M)[\theta * \delta]$$

In particular, the actual evaluation of reflexivity of an abstraction is

$$\big((\lambda x.M)\langle\delta\rangle\big)[\theta] \equiv (\lambda x.M)[\theta * \delta]$$

which is, of course, a closure and doesn't go under the λ until applied or read back.

Some categorical remarks

In combination, environments are acted on by arbitrary morphisms in the BCH cube category (composites of faces and degeneracies).

$$\theta * (\phi \circ \delta) = (\theta * \phi) * \delta$$

In an algebraic presentation, substitutions ($\sim$ environments) are indexed by a dimension:

$$\theta : \Gamma \xrightarrow{n} \Delta$$

and are acted on by morphisms in the cube category:

$$\frac{\theta : \Gamma \xrightarrow{n} \Delta \qquad \psi : m \to n}{\theta * \psi : \Gamma \xrightarrow{m} \Delta}$$

Thus, we have a cubical set of substitutions from Γ to Δ. That is,

The category of contexts is enriched over cubical sets.

We thus expect an enriched version of categorical gluing to appear in a formal proof of normalization.

Higher-dimensional NbE

With these modifications. . .

. . . and a lot of omitted work and details. . .

. . . we get a normalization by evaluation algorithm.

$$\begin{tikzcd} \text{terms} \arrow[r, "\text{eval}"] & \text{values} \arrow[loop above, "\text{degeneracies}"] \arrow[r, "\text{readback}"] & \text{normals} \end{tikzcd}$$

Using this for equality-checking, we then implement a typechecker.

`https://github.com/mikeshulman/narya`