The Box of Delights
(Cubical Observational Type Theory)

James Chapman, Fredrik Nordvall Forsberg, Conor McBride

January 14, 2018

1 What is Cubical Type Theory?

In ordinary type theory, the judgments characterize the construction of a type, or of a value in a type: a single thing. In cubical type theory, the judgments characterize the simultaneous construction of continuously varying things at all points in some n-dimension cubical space (i.e., a point, a line, a squares, a cubes, a tesseract,...). The notion of 'propositional equality' is supplanted by the notion of *path* between two given points, or more generally, the interior enclosed by a given surface. There is no reason to expect such interiors to be devoid of information.

2 Bidirectional Religion

The judgments of the system are n-place predicates, and we further characterize those places as having one of three *modes*. They may be

inputs being information we already trust in some way that we must specify;

subjects being syntactic objects, trust in which we seek to establish;

outputs being information we generate in the process of establishing that trust, which must be trustworthy in some way that we must specify.

Every syntactic category has a small step computation relation:

$$\boxed{s \rightsquigarrow t}$$

Every judgment form is declared with its subjects in braces, inputs before and outputs after,

$$\boxed{\overline{\imath}\ \{\overline{s}\}\ \overline{o}}$$

The braces are not part of the syntax of the judgment. Judgments are closed under *pre*computation of inputs and *post*computation of outputs.

$$\frac{\overline{\imath} \rightsquigarrow \overline{\imath}' \quad \overline{\imath}'\ \overline{s}\ \overline{o}}{\overline{\imath}\ \overline{s}\ \overline{o}} \qquad \frac{\overline{\imath}\ \overline{s}\ \overline{o} \quad \overline{o} \rightsquigarrow \overline{o}'}{\overline{\imath}\ \overline{s}\ \overline{o}'}$$

We are not in the habit of writing contexts in rules, only context *extensions*. In a premise, we may write $\Delta \vdash J$ to mean that Δ is added to the local end of the context (i.e., on the right). We must ensure that Δ comprises context entries which are incrementally well formed over the current context. A context entry assigns some properties to some fresh variable. We may write a premise $\dashv x \ldots$ to look up the entry for x in the context.

3 Points

$$\boxed{\text{POINT } \{p\}}$$

$$\frac{}{\text{POINT } 0} \qquad \frac{}{\text{POINT } 1} \qquad \frac{\dashv i}{\text{POINT } i} \qquad \frac{\text{POINT } p \quad \text{POINT } p_0 \quad \text{POINT } p_1}{\text{POINT } p(p_0\text{-}p_1)}$$

We call 0 and 1 the *endpoints*.

Point variables, also known as *dimensions*, are bound in the context. The context entry for a dimension carries no additional information. The context thus *orders* the dimensions, right to left, from local to global.

The computational mechanism for points is *affine rescaling*, satisfying

$$0(p_0\text{-}p_1) = p_0 \qquad 1(p_0\text{-}p_1) = p_1 \qquad p'(p\text{-}p) = p$$

We also call this operation 'multiplexing' or 'conditional expression'.

The normal form of a point expression over a vector of dimensions is its *reduced ordered binary decision diagram*. That is,

- a normal point over no dimensions is an endpoint: either 0 or 1;
- a normal point over dimensions $\bar{\imath}, j$ is either a normal point over $\bar{\imath}$ or $j(p_0\text{-}p_1)$ where p_0 and p_1 are *distinct* normal points over $\bar{\imath}$ (and we call j the *pivot* of the point).

The naïve normalization algorithm for points over n-dimensions is simply to evaluate them for all 2^n assignments of endpoints to dimensions: for a given $p(\bar{\imath}, j)$, normalize and compare $p(\bar{\imath}, 0)$ and $p(\bar{\imath}, 1)$ to determine whether j is relevant. However, it is straightforward to compute affine rescaling for normal points symbolically: the most local pivot of the three inputs is the candidate for the pivot of the output. To compute $p(p_0\text{-}p_1)$ for normal points, either

- we see that p is endpoint i and deliver p_i accordingly, or
- we see that p has a pivot, in which case the three points have a most local pivot, i, so we can normalize $p^{i=j}(p_0^{i=j}\text{-}p_1^{i=j})$ for the cases $i = 0$ and $i = 1$, where $(i(p_0\text{-}p_1))^{i=j} = p_j$ and otherwise $p^{i=j} = p$; if these coincide, they are the normal form, otherwise they form the range for pivot i.

4 (No) Types (Yet): Basic Setup

Usages of things are subject to type *synthesis*. The type of types is $*$ (and we know we are purchasing simplicity with inconsistency, but our focus is elsewhere). The context entry for a value variable looks like $x : S$ where S is the type of x, and must thus be checkably a type, i.e., $* \ni S$.

$$\boxed{\{e\} \in S} \qquad * \ni S$$

$$\frac{\dashv x : S}{x \in S} \qquad \frac{* \ni T \quad T \ni t}{t : T \in T}$$

The type annotation syntax $t : T$ denotes the usage of construction t at type T. The presence of the type allows t to be used.

Constructions of things are subject to type *checking*.

$$* \ni T \qquad \boxed{T \ni \{t\}}$$

$$\frac{}{* \ni *} \qquad \frac{e \in S \quad * \ni S = T}{T \ni \underline{e}}$$

We consider $*$ to be a *canonical* construction. The syntax $\underline{e}$ denotes a non-canonical construction, given by a usage.

A construction which is not being used should no longer be labelled by its type.

$$\boxed{t \rightsquigarrow t'}$$
$$\underline{t : T} \rightsquigarrow t$$

We define judgmental equality for usages, resynthesizing their common type.

$$e_i \in S_i \quad \boxed{e_0 = e_1\{\} \in S} \quad * \ni S \quad * \ni S = S_i$$

$$\frac{\dashv x : S}{x = x \in S} \qquad \frac{e_0 = e_1 \in S}{(\underline{e_0} : T) = e_1 \in S} \qquad \frac{e_0 = e_1 \in S}{e_0 = (\underline{e_1} : T) \in S}$$

We define judgmental equality for constructions with their type given in advance.

$$* \ni T \quad T \ni t_i \quad \boxed{T \ni t_0 = t_1\{\}}$$

$$\frac{}{* \ni * = *} \qquad \frac{e_0 = e_1 \in S}{\underline{E} \ni \underline{e_0} = \underline{e_1}}$$

We shall insist upon the *progress* property: every checkable construction must either be canonical, computable, or *stuck* at a free value variable. $\underline{x}$ is stuck at x. The presence of free variables thus offers us an excuse for not computing to canonical form. By contrast, the presence of free dimensions offers us no such escape from responsibility, but rather *increases* our obligations: we must produce a continuously varying canonical form if we cannot blame a free variable for getting us stuck.

5 Functions, Pairing and One

Let us have some *things* in our type theory. Functions inhabit dependent function types, are constructed by abstraction over values, and are used by application.

$$\frac{* \ni S \quad x : S \vdash * \ni T(x)}{* \ni \Pi x : S.\, T(x)} \qquad \frac{x : S \vdash T(x) \ni t(x)}{\Pi x : S.\, T(x) \ni \lambda x.\, t(x)} \qquad \frac{f \in \Pi x : S.\, T(x) \quad S \ni s}{f\ s \in T(s : S)}$$

Computation is by β-reduction,

$$(\lambda x.\, t(x) : \Pi x : S.\, T(x))\ s \ \rightsquigarrow\ t(s : S) : T(s : S)$$

but we also impose an η-law

$$\frac{x : S \vdash T(x) \ni (t : \_)\ \underline{x} = (t' : \_)\ \underline{x}}{\Pi x : S.\, T \ni t = t'}$$

where each $\_$ is the only type it can be—$\Pi x : S.\, T$.

Meanwhile, pairs inhabit dependent pair types, are constructed by pairing, and are used by projection, which we write postfix.

$$\frac{* \ni S \quad x : S \vdash * \ni T(x)}{* \ni \Sigma x : S.\, T(x)} \qquad \frac{S \ni s \quad T(s : S) \ni t}{\Sigma x : S.\, T(x) \ni s, t} \qquad \frac{e \in \Sigma x : S.\, T(x)}{e\ \pi_0 \in S} \qquad \frac{e \in \Sigma x : S.\, T(x)}{e\ \pi_1 \in T(e\ \pi_0)}$$

Again, we have β-laws

$$(s, t : \Sigma x : S.\, T(x))\ \pi_0 \rightsquigarrow s : S \qquad (s, t : \Sigma x : S.\, T(x))\ \pi_1 \rightsquigarrow t : T(s : S)$$

and a componentwise η-law.

$$\frac{S \ni (t\!:\!\_)\ \pi_0 = (t'\!:\!\_)\ \pi_0 \qquad T((t\!:\!\_)\ \pi_0) \ni (t\!:\!\_)\ \pi_1 = (t'\!:\!\_)\ \pi_1}{\Sigma x\!:\!S.\, T(x) \ni t = t'}$$

The unit type, 'One', is straightforward.

$$\frac{}{* \ni 1} \qquad \frac{}{1 \ni \star} \qquad \frac{}{1 \ni t = t'}$$

6 Surfaces and Interiors

We may form a type of *paths* between two given values as follows:

$$\frac{i \vdash * \ni T(i) \quad T(0) \ni t_0 \quad T(1) \ni t_1}{* \ni [i|T(i)]t_0\text{-}t_1}$$

The path type specifies the *typespace* for a line of constructions, and the values on its *surface*: the surface of a line consists of the points at either end.

A path may be formed by abstracting a construction over a dimension: it gives the *interior* of the line of constructions as a valid construction at each type in the typespace. Moreover, the interior must *connect* with the specified surface.

$$\frac{i \vdash T(i) \ni t(i) \quad T(0) \ni t(0) = t_0 \quad T(1) \ni t(1) = t_1}{[i|T(i)]t_0\text{-}t_1 \ni \langle i \rangle t(i)}$$

We may project from a path at a point.

$$\frac{e \in [i|T(i)]t_0\text{-}t_1 \quad \text{POINT } p}{e\ p \in T(p)} \qquad (\langle i \rangle t(i) : [i|T(i)]t_0\text{-}t_1)\ p \rightsquigarrow t(p) : T(p)$$

Moreover, projection at a surface point (i.e., either *endpoint*) computes, without any need to inspect the interior.

$$e \in [i|T(i)]t_0\text{-}t_1 \Rightarrow e\ 0 \rightsquigarrow t_0 : T(0) \qquad e \in [i|T(i)]t_0\text{-}t_1 \Rightarrow e\ 1 \rightsquigarrow t_1 : T(1)$$

Judgmental equality for path types and projection are both structural. For paths, we impose an η-law.

$$\frac{i \vdash T(i) \ni (t\!:\![i|T(i)]t_0\text{-}t_1)\ i = (t'\!:\![i|T(i)]t_0\text{-}t_1)\ i}{[i|T(i)]t_0\text{-}t_1 \ni t = t'}$$

We adopt the convention that at any binding site for a dimension, we may add an affine rescaling.

$$\langle i\ p_0\text{-}p_1 \rangle t(i) = \langle i \rangle t(i(p_0\text{-}p_1))$$

This allows us to zoom in on any segment of a path:

$$\langle i\ p_0\text{-}p_1 \rangle \underline{e\ i}$$

is that segment of path e that runs p_0-p_1.

By iterating the path type construction, we may give types to n-dimensional interiors with an n-dimensional typespace and a surface comprising $2n$ faces, each with $n-1$ dimensions, connecting to each other at $2n(n-1)$ edges. Note that when

n is one, we get that the surface consists of two 0-dimensional constructions which need not connect at all, exactly as above.

If we have a vector of dimensions $\bar{\imath}$, a typespace is a suitably parametrised type $T(\bar{\imath})$, and a surface is a vector $\bar{t}(\bar{\imath})$ of $2n$ faces,

$$t_0^0(\bar{\imath}_0)\text{-}t_1^0(\bar{\imath}_0), \ldots, t_0^{n-1}(\bar{\imath}_{n-1})\text{-}t_1^{n-1}(\bar{\imath}_{n-1})$$

where $\bar{\imath}_k$ denotes the dimension vector with the kth dimension deleted. Of course, these faces must connect. We introduce derived judgment forms for surfaces, allowing us to manipulate them more conveniently. Firstly, we have a judgment for typechecking a surface, which must necessarily check connections. Secondly, we have a judgment for checking that an interior connects with its surface.

$$\bar{\imath} \vdash * \ni T(\bar{\imath}) \qquad \boxed{\nabla\bar{\imath} \vdash T(\bar{\imath}) \ni \bar{t}(\bar{\imath})}$$

$$\frac{}{\nabla \vdash T \ni}$$

$$\frac{\bar{\imath} \vdash T(0, \bar{\imath}) \ni t_0(\bar{\imath}) \quad \bar{\imath} \vdash T(1, \bar{\imath}) \ni t_1(\bar{\imath}) \quad i \vdash \nabla\bar{\imath} \vdash T(i, \bar{\imath},) \ni \bar{t}(i, \bar{\imath}) \quad \nabla\bar{\imath} \vdash T(0, \bar{\imath}) \ni t_0(\bar{\imath}) = \bar{t}(0, \bar{\imath}) \quad \nabla\bar{\imath} \vdash T(1, \bar{\imath}) \ni t_1(\bar{\imath}) = \bar{t}(1, \bar{\imath})}{\nabla i, \bar{\imath} \vdash T(i, \bar{\imath}) \ni t_0(\bar{\imath})\text{-}t_1(\bar{\imath}), \bar{t}(i, \bar{\imath})}$$

For no dimensions, the surface has no faces. Otherwise, we have a pair of end-faces $t_0(\bar{\imath})$ and $t_1(\bar{\imath})$ for the endpoints of the first dimension, then a tube made of side-faces $\bar{t}(i, \bar{\imath})$, which you can just as well see as an *outline* (the surface of a face) parametrized over the first dimension. The surface of a cube, for example, has two solid end-squares and four side-faces which are also solid squares, but can also be seen as a hollow square moving from one end-square to the other. The end-faces must connect with the ends of the tube.

$$\bar{\imath} \vdash * \ni T(\bar{\imath}) \quad \bar{\imath} \vdash T(\bar{\imath}) \ni t(\bar{\imath}) \quad \nabla\bar{\imath} \vdash T(\bar{\imath}) \ni \bar{t}(\bar{\imath}) \qquad \boxed{\nabla\bar{\imath} \vdash T(\bar{\imath}) \ni t(\bar{\imath}) = \bar{t}(\bar{\imath})}$$

$$\frac{}{\nabla \vdash \bar{\imath} \vdash T(\bar{\imath}) \ni t(\bar{\imath}) =}$$

$$\frac{\bar{\imath} \vdash T(0, \bar{\imath}) \ni t(0, \bar{\imath}) = t_0(\bar{\imath}) \quad \bar{\imath} \vdash T(1, \bar{\imath}) \ni t(1, \bar{\imath}) = t_1(\bar{\imath}) \quad \nabla i, \bar{\imath} \vdash T(i, \bar{\imath}) \ni t(i, \bar{\imath}) = \bar{t}(i, \bar{\imath})}{\nabla i, \bar{\imath} \vdash T(i, \bar{\imath}) \ni t(i, \bar{\imath}) = t_0(\bar{\imath})\text{-}t_1(\bar{\imath}), \bar{t}(i, \bar{\imath})}$$

We may write $\nabla\bar{\imath} \vdash T(\bar{\imath}) \ni \bar{t}(\bar{\imath})$, for the top-level judgment where the accumulated list of 'visited' dimensions is empty and we are talking about the entire surface.

Now, we may formulate the derivable notion of n-dimensional *interior* type.

$$\frac{\bar{\imath} \vdash * \ni T(\bar{\imath}) \quad \nabla\bar{\imath} \vdash T(\bar{\imath}) \ni \bar{t}(\bar{\imath})}{* \ni [\bar{\imath}|T(\bar{\imath})]\bar{t}(\bar{\imath})} \qquad \frac{\bar{\imath} \vdash T(\bar{\imath}) \ni t(\bar{\imath}) \quad \nabla\bar{\imath} \vdash T(\bar{\imath}) \ni t(\bar{\imath}) = \bar{t}(\bar{\imath})}{[\bar{\imath}|T(\bar{\imath})]\bar{t}(\bar{\imath}) \ni \langle\bar{\imath}\rangle t(\bar{\imath})}$$

Of course, $\langle\bar{\imath}\rangle t(\bar{\imath})$ is exactly iterated path construction, and $[\bar{\imath}|T(\bar{\imath})]\bar{t}(\bar{\imath})$ is the iterated path type.

$$[|T] = T \qquad [i, \bar{\imath}|T(i, \bar{\imath})]t_0(\bar{\imath})\text{-}t_1(\bar{\imath}), \bar{t}(i, \bar{\imath}) = [i|[\bar{\imath}|T(i, \bar{\imath})]\bar{t}(i, \bar{\imath})]\langle\bar{\imath}\rangle t_0(\bar{\imath})\text{-}\langle\bar{\imath}\rangle t_1(\bar{\imath})$$

Crucially, we may just as well see this iteration inside-out as outside-in. It is also the case that an n-dimensional interior whose typespace consists of path types *is* an $(n+1)$-dimensional interior.

$$[\bar{\imath}|[j|T(\bar{\imath}, j)]t_0(\bar{\imath}, j)\text{-}t_1(\bar{\imath}, j)]\langle j\rangle\bar{t}(\bar{\imath}, j) = [\bar{\imath}, j|T(\bar{\imath}, j)]\bar{t}(\bar{\imath}, j), t_0(\bar{\imath}, j)\text{-}t_1(\bar{\imath}, j)$$

7 Transport of Delight: Gathering and Scattering

We may send a value across a line of types.

$$\frac{i \vdash * \ni T(i) \quad T(0) \ni t}{t\ [i|T(i)\rangle \in T(1)}$$

Affine rescaling allows us to extrude the value across the line

$$t\ [j\ 0\text{-}i|T(j)\rangle \in T(i)$$

assuming that transportation supports the property of *regularity*: when the line of types is degenerate, the value is unchanged.

$$t\ [\_|T\rangle \rightsquigarrow t : T$$

Regularity is enough to ensure that

$$[i|T(i)]t\text{-}(t\ [i|T(i)\rangle) \ni \langle i\rangle t\ [j\ 0\text{-}i|T(j)\rangle$$

noting that when $i = 0$, the transportation becomes degenerate in typespace $T(0)$, so that regularity establishes the connection with t at the 0 end.

Lines are, however, a source of confusion. What is the $(n+1)$-dimensional generalization? Along any 'square' tube, we may transmit an interior for one end (the base) to an interior for the other.

$$\frac{\begin{array}{c} i, \overline{j} \vdash * \ni T(i, \overline{j}) \\ \overline{j} \vdash T(0, \overline{j}) \ni t(\overline{j}) \quad i \vdash \nabla \overline{j} \vdash T(i, \overline{j}) \ni \overline{t}(i, \overline{j}) \quad \nabla \overline{j} \vdash T(0, \overline{j}) \ni t(\overline{j}) = \overline{t}(0, \overline{j}) \\ \text{POINT } \overline{p} \end{array}}{t(\overline{j}), \overline{t}(i, \overline{j})\ [i, \overline{j}|T(i, \overline{j})\rangle\ \overline{p} \in T(1, \overline{p})}$$

What is a square tube with a base? It's an open *box*, which we can write as a single solid face $t(\overline{j})$, followed by a vector of opposing faces $\overline{t}(i, \overline{j})$ which you can just as well see as an outline parametrised by the dimension of travel. We access the interior we are constructing at the other end (the lid of the box) pointwise, so we give a vector of points to identify our destination. For a given typespace, transportation amounts to *gathering* a box of values to a point on its lid. Generalized regularity amounts to a box with degenerate volume and sides collapsing to its base.

$$t(\overline{j}), \overline{t}(\_, \overline{j})\ [\_, \overline{j}|T(\_, \overline{j})\rangle\ \overline{p} \rightsquigarrow t(\overline{p}) : T(\overline{p})$$

Fortunately for us, $(n+1)$-dimensional transportation is exactly transportation over a line of n-dimensional interior types.

$$t(\overline{j}), \overline{t}(i, \overline{j})\ [i, \overline{j}|T(i, \overline{j})\rangle\ \overline{p} = (\langle \overline{j}\rangle t(\overline{j}))\ [i|[\overline{j}|T(i, \overline{j})]\overline{t}(i, \overline{j})\rangle\ \overline{p}$$

Or rather, the other way around. it is the dimensionally general version which admits the compositional definition. We have just discovered how to gather a box of paths—move up a dimension, consider the endpoints of paths as part of the box and work pointwise.

To achieve canonical form, we must explain how to gather a box of canonical values in a canonical type. For ease of notation, we may write the whole box as one vector, comprising the base and the sides. It makes sense to match vectors with constructor patterns, for if any value in the vector is canonical, the connections ensure that the rest are also canonical with the same head.

For types, we may start with

$$\overline{*}\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow *\qquad \overline{1}\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow 1$$

although these are, in any case, a consequence of regularity.

Let us try to gather a box of function types. Working compositionally, we can make a start. Gathering the box of domains is no problem, but we then need to *construct* the box of codomains.

$$\Pi x : \overline{S}(i, \overline{\jmath}).\, \overline{T}(i, \overline{\jmath}, x)\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow \Pi x : (\overline{S}(i, \overline{\jmath})\ [i, \overline{\jmath}|*\rangle\ \overline{p}).\, (\overline{T}(i, \overline{\jmath}, ?)\ [i, \overline{\jmath}|*\rangle\ \overline{p}) : *$$

The bound variable x inhabits a particular type on the lid of domains, but we need to instantiate codomain types that come from the box. We need to *scatter* x from a lid point to the box, which we can do by transportation. First, we construct the entire domain typespace by filling the box,

$$S(i', \overline{\jmath}') = \overline{S}(i, \overline{\jmath})\ [i\ 0\text{-}i', \overline{\jmath}|*\rangle\ \overline{\jmath}'$$

then we construct the scattering of x,

$$\hat{x}(i', \overline{\jmath}') = x\ [k|S(k(1, \overline{\jmath}\text{-}\, i', \overline{\jmath}'))\rangle : S(i', \overline{\jmath}')$$

and now we can build our box of codomains, so

$$\Pi x : \overline{S}(i, \overline{\jmath}).\, \overline{T}(i, \overline{\jmath}, x)\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow \Pi x : (\overline{S}(i, \overline{\jmath})\ [i, \overline{\jmath}|*\rangle\ \overline{p}).\, (\overline{T}(i, \overline{\jmath}, \hat{x}(i, \overline{\jmath}))\ [i, \overline{\jmath}|*\rangle\ \overline{p}) : *$$

In general, for a box $\overline{t}(i, \overline{\jmath})$ we may fill it to make $t(i, \overline{\jmath})$ by gathering. For some bound variable in a lid type $x : S(1, \overline{p})$, we may scatter it to any $\hat{x}(i, \overline{\jmath})$ in the typespace and hence to the entire box. And that is how it will work for all of the canonical forms: we gather covariantly and scatter contravariantly.

$$\begin{aligned}&\lambda x.\, \overline{t}(i, \overline{\jmath}, x)\ [i, \overline{\jmath}|\Pi x : S(i, \overline{\jmath}).\, T(i, \overline{\jmath}, x)\rangle\ \overline{p} \rightsquigarrow\\ &\quad \lambda x.\, \overline{t}(\overline{\jmath}, \hat{x}(i, \overline{\jmath}))\ [i, \overline{\jmath}|T(i, \overline{\jmath}, \hat{x}(i, \overline{\jmath}))\rangle\ \overline{p} : \Pi x : S(1, \overline{p}).\, T(1, \overline{p}, x)\end{aligned}$$

For pairs, we have

$$\Sigma x : \overline{S}(i, \overline{\jmath}).\, \overline{T}(i, \overline{\jmath}, x)\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow \Sigma x : (\overline{S}(i, \overline{\jmath})\ [i, \overline{\jmath}|*\rangle\ \overline{p}).\, (\overline{T}(i, \overline{\jmath}, \hat{x}(i, \overline{\jmath}))\ [i, \overline{\jmath}|*\rangle\ \overline{p}) : *$$

and

$$\begin{aligned}&\overline{s}(i, \overline{\jmath}), \overline{t}(i, \overline{\jmath})\ [i, \overline{\jmath}|\Sigma x : S(i, \overline{\jmath}).\, T(i, \overline{\jmath}, x)\rangle\ \overline{p} \rightsquigarrow\\ &\quad s(1, \overline{p}), \overline{t}(i\ \overline{\jmath})\ [i, \overline{\jmath}|T(i, \overline{\jmath}, \hat{x}(i, \overline{\jmath}))\rangle\ \overline{p} : \Sigma x : S(1, \overline{p}).\, T(1, \overline{p}, x)\end{aligned}$$

where $s(p, \overline{p}) = \overline{s}(i\ \overline{\jmath})\ [i\ 0\text{-}p, \overline{\jmath}|S(i, \overline{\jmath})\rangle\ \overline{p}$, is the usual gathering.

For the unit type, we have

$$\star\ [i, \overline{\jmath}|1\rangle\ \overline{p} \rightsquigarrow \star$$

We have already said how to transport paths pointwise, by moving the endpoints into a box with one more dimension, but we must transport path *types*.

$$[k|\overline{T}(i, \overline{\jmath}, k)]\overline{t}_0(i, \overline{\jmath}, k)\text{-}\overline{t}_1(i, \overline{\jmath}, k)\ [i, \overline{\jmath}|*\rangle\ \overline{p} \rightsquigarrow [k|T(1, \overline{p}, k)]t_0(1, \overline{p}, k)\text{-}t_1(1, \overline{p}, k) : *$$

where $T(p, \overline{p}, k) = \overline{T}(i, \overline{\jmath}, k)\ [i\ 0\text{-}p, \overline{\jmath}|*\rangle\ \overline{p}$ is again the usual gathering, pointwise in the k dimension, and we unzip our vector of opposing faces as an opposing pair of vectors, then gather each vector.

We have thus delivered gathering for all canonical boxes in all canonical typespaces, with the result that every normal form in a context with exactly n dimensions is a canonical n-dimensional form.

8 Observational Type Theory

What we have done here is exactly to reconstruct the observational type theory (OTT) of Altenkirch, McBride and Swierstra in a proof-relevant, cubical style. In the proof-*irrelevant* setting, equations between equality proofs did not matter, so the higher-dimensional structure was not apparent. Moreover, working in only one dimension creates a misleading symmetry between *gathering* and *scattering*, because a one-dimensional box is exactly a point. OTT demanded a *coercion* operation (one-dimensional transportation), and a *coherence* proof (a transported value is (heterogeneously) equal to the original): proof irrelevance allows coherence to be axiomatic, as consistent *propositional* assumptions are no barrier to computation. Here, however, affine rescaling is exactly what recovers coherence from coercion: we can transport a value along none, some, or all of any journey.

We obtain functional extensionality as a matter of course, a function yielding paths becomes a path between functions just by flipping the value input and the dimension of travel. Given two functions which are pointwise equal

$$f_i : \Pi x : S.\, T(x) \qquad q : \Pi x : S.\, [\_|T(x)]\, f_0\ x \text{ - } f_1\ x$$

construct

$$[\_|\Pi x : S.\, T(x)]\, f_0\text{-}f_1 \ni \langle i \rangle q\ i$$

The contractability of the singleton is also straightforward. Given

$$t : T \qquad s : \Sigma x : T.\, [\_|T]\, t\text{-}x$$

construct

$$[\_|\Sigma x : T.\, [\_|T]\, t\text{-}x](t, \langle \_ \rangle t)\text{-}s \ni \langle i \rangle (s\ \pi_1\ i, \langle j\ 0\text{-}i \rangle s\ \pi_1\ j)$$

and note that when $s = (t, \langle \_ \rangle t)$, this computes to $\langle \_ \rangle (t, \langle \_ \rangle t)$. As a consequence, the traditional 'J' rule for the inductively defined equality is recoverable (transportation between types indexed by singletons). Moreover regularity ensures that the computational behaviour of J survives intact.

9 Turn One Way, Towards Proof-Irrelevance

The computational behaviour of transportation exploited the uniformity of boxes of canonical forms. This uniformity is inevitable, given no way to break it: we can only form paths which are *parametric*. We could double-down on this and seek to restore proof-irrelevance by identifying all interiors with the same surface and, in effect using uniformity to infer the interior from the surface when we have at least one dimension (hence a nontrivial surface).

10 Turn the Other Way, Towards Univalence

Heading in exactly the opposite direction, rather than exploiting parametricity, we could deliberately break it. Univalence allows us to define a path by *case analysis* on a point, so long as the two branches are isomorphic. Such a case analysis should be regarded as a canonical object in higher dimensions. Indeed, we might try to find some sort of representation as a reduced ordered binary decision diagram, augmented with isomorphisms. The corresponding higher-dimensional values simply record the entry point of a value in one of the isotopes. Transportation of values will then need to work case-by-case, with any non-parametricity in the typespace or the box leading to non-parametricity in the output. Somehow, a degenerate typespace and tube might manage not to preserve exactly the non-parametricity in the base.

11 Conclusion

We have a great deal more to think about.