Polarized Cubical Types

Abstract

...

1. Introduction

In a computational world in which the laws of physics are embraced and resources are carefully maintained (e.g., quantum computing [Abramsky and Coecke 2004; Nielsen and Chuang 2000]), programs must be reversible. Although this is apparently a limiting idea, it turns out that conventional computation can be viewed as a special case of such resource-preserving reversible programs. This thesis has been explored for many years from different perspectives [Bennett 2003, 2010, 1973; Fredkin and Toffoli 1982; Landauer 1961, 1996; Toffoli 1980]. We build on the work of James and Sabry [2012] which expresses this thesis in a type theoretic computational framework, expressing computation via type isomorphisms.

Make sure we introduce the abbreviation HoTT in the introduction [The Univalent Foundations Program 2013].

2. Computing with Type Isomorphisms

The main syntactic vehicle for the developments in this paper is a simple language called Π whose only computations are isomorphisms between finite types. The set of types τ includes the empty type 0, the unit type 1, and conventional sum and product types. The values of these types are the conventional ones: $()$ of type 1, $\mathsf{inl}\ v$ and $\mathsf{inr}\ v$ for injections into sum types, and (v_1, v_2) for product types:

$$\begin{array}{llcl}
(\textit{Types}) & \tau & ::= & 0 \mid 1 \mid \tau_1 + \tau_2 \mid \tau_1 * \tau_2 \\
(\textit{Values}) & v & ::= & () \mid \mathsf{inl}\ v \mid \mathsf{inr}\ v \mid (v_1, v_2) \\
(\textit{Combinator types}) & & & \tau_1 \leftrightarrow \tau_2 \\
(\textit{Combinators}) & c & ::= & [\textit{see Table 1}]
\end{array}$$

The interesting syntactic category of Π is that of *combinators* which are witnesses for type isomorphisms $\tau_1 \leftrightarrow \tau_2$. They consist of base combinators (on the left side of Table 1) and compositions (on the right side of the same table). Each line of the table on the left introduces a pair of dual constants[1] that witness the type isomorphism in the middle. This set of isomorphisms is known to be complete [Fiore 2004; Fiore et al. 2006] and the language is universal for hardware combinational circuits [James and Sabry 2012]. The trace operator provides a bounded iteration facility which adds no expressiveness in the current context but will be needed in Sec. 3.[2]

As simple illustrative examples of "programming" in Π, here are three useful combinators that we define here for future reference:

$$\begin{array}{ll}
\mathit{assoc}_1 : & \tau_1 + (\tau_2 + \tau_3) \leftrightarrow (\tau_2 + \tau_1) + \tau_3 \\
\mathit{assoc}_1 = & \mathit{assocl}_+ \fatsemi (\mathit{swap}_+ \oplus \mathit{id}) \\
\\
\mathit{assoc}_2 : & (\tau_1 + \tau_2) + \tau_3 \leftrightarrow (\tau_2 + \tau_3) + \tau_1 \\
\mathit{assoc}_2 = & (\mathit{swap}_+ \oplus \mathit{id}) \fatsemi \mathit{assocr}_+ \fatsemi (\mathit{id} \oplus \mathit{swap}_+) \fatsemi \mathit{assocl}_+ \\
\\
\mathit{assoc}_3 : & (\tau_1 + \tau_2) + \tau_3 \leftrightarrow \tau_1 + (\tau_3 + \tau_2) \\
\mathit{assoc}_3 = & \mathit{assocr}_+ \fatsemi (\mathit{id} \oplus \mathit{swap}_+)
\end{array}$$

From the perspective of category theory, the language Π models what is called a traced *symmetric bimonoidal category* or a *commutative rig category*. These are categories with two binary operations $\oplus$ and $\otimes$ satisfying the axioms of a rig (i.e., a ring without negative elements also known as a semiring) up to coherent isomorphisms. And indeed the types of the Π-combinators are precisely the semiring axioms. A formal way of saying this is that Π is the *categorification* [Baez and Dolan 1998] of the natural numbers. A simple (slightly degenerate) example of such categories is the category of finite sets and permutations in which we interpret every Π-type as a finite set, the values as elements in these finite sets, and the combinators as permutations. Another common example of such categories is the category of finite dimensional vector spaces and linear maps over any field. Note that in this interpretation, the Π-type 0 maps to the 0-dimensional vector space which is *not* empty. Its unique element, the zero vector — which is present in every vector space — acts like a "bottom" everywhere-undefined element and hence the type behaves like the unit of addition and the annihilator of multiplication as desired.

3. The Int Construction

Our immediate technical goal is to explore an extension of Π with a notion of higher-order functions. In the context of monoidal categories, it is known that a notion of higher-order functions emerges from having an additional degree of *symmetry*. In particular, both the **Int** construction of Joyal, Street, and Verity [1996] and the closely related $\mathcal{G}$ construction of linear logic [Abramsky 1996] construct higher-order *linear* functions by considering a new category built on top of a given base traced monoidal category. The objects of the new category are of the form $(\tau_1 - \tau_2)$ where τ_1 and τ_2 are objects in the base category. Intuitively, the component τ_1 is viewed as a conventional type whose elements represent values flowing, as usual, from producers to consumers. The component τ_2 is viewed as a *negative type* whose elements represent demands for values or equivalently values flowing backwards. Under this interpretation,

[1] where swap_+ and swap_* are self-dual.

[2] If recursive types are added, the trace operator provides unbounded iteration and the language becomes Turing complete [Bowman et al. 2011; James and Sabry 2012]. We will not be concerned with recursive types in this paper.

[Copyright notice will appear here once 'preprint' option is removed.]

$$\frac{}{\vdash id : \tau \leftrightarrow \tau} \qquad \frac{\vdash c : \tau_1 \leftrightarrow \tau_2}{\vdash sym\ c : \tau_2 \leftrightarrow \tau_1}$$

$$\begin{array}{rrcll}
identl_+ : & 0 + \tau & \leftrightarrow & \tau & : identr_+ \\
swap_+ : & \tau_1 + \tau_2 & \leftrightarrow & \tau_2 + \tau_1 & : swap_+ \\
assocl_+ : & \tau_1 + (\tau_2 + \tau_3) & \leftrightarrow & (\tau_1 + \tau_2) + \tau_3 & : assocr_+ \\
identl_* : & 1 * \tau & \leftrightarrow & \tau & : identr_* \\
swap_* : & \tau_1 * \tau_2 & \leftrightarrow & \tau_2 * \tau_1 & : swap_* \\
assocl_* : & \tau_1 * (\tau_2 * \tau_3) & \leftrightarrow & (\tau_1 * \tau_2) * \tau_3 & : assocr_* \\
dist_0 : & 0 * \tau & \leftrightarrow & 0 & : factor_0 \\
dist : & (\tau_1 + \tau_2) * \tau_3 & \leftrightarrow & (\tau_1 * \tau_3) + (\tau_2 * \tau_3) & : factor
\end{array}$$

$$\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_2 \leftrightarrow \tau_3}{\vdash c_1 \fatsemi c_2 : \tau_1 \leftrightarrow \tau_3}$$

$$\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_3 \leftrightarrow \tau_4}{\vdash c_1 \oplus c_2 : \tau_1 + \tau_3 \leftrightarrow \tau_2 + \tau_4}$$

$$\frac{\vdash c_1 : \tau_1 \leftrightarrow \tau_2 \quad \vdash c_2 : \tau_3 \leftrightarrow \tau_4}{\vdash c_1 \otimes c_2 : \tau_1 * \tau_3 \leftrightarrow \tau_2 * \tau_4}$$

$$\frac{\vdash c : \tau + \tau_1 \leftrightarrow \tau + \tau_2}{\vdash trace\ c : \tau_1 \leftrightarrow \tau_2}$$

Table 1. Π-combinators [James and Sabry 2012]

and as we explain below, a function is nothing but an object that converts a demand for an argument into production of a result.

We begin our formal development by extending Π with a new universe of types $\mathbb{T}$ that consists of composite types $(\tau_1 - \tau_2)$:

$$(\textit{1d types}) \quad \mathbb{T} \quad ::= \quad (\tau_1 - \tau_2)$$

In anticipation of future developments, we will refer to the original types τ as 0-dimensional (0d) types and to the new types $\mathbb{T}$ as 1-dimensional (1d) types. It turns out that, except for one case discussed below, the 1d level is a "lifted" instance of Π with its own notions of empty, unit, sum, and product types, and its corresponding notion of isomorphisms on these 1d types.

Our next step is to define lifted versions of the 0d types:

$$\begin{array}{rcl}
\mathbb{0} & \triangleq & (0 - 0) \\
\mathbb{1} & \triangleq & (1 - 0) \\
(\tau_1 - \tau_2) \boxplus (\tau_3 - \tau_4) & \triangleq & (\tau_1 + \tau_3) - (\tau_2 + \tau_4) \\
(\tau_1 - \tau_2) \boxtimes (\tau_3 - \tau_4) & \triangleq & ((\tau_1 * \tau_3) + (\tau_2 * \tau_4)) - \\
& & ((\tau_1 * \tau_4) + (\tau_2 * \tau_3))
\end{array}$$

Building on the idea that Π is a categorification of the natural numbers and following a long tradition that relates type isomorphisms and arithmetic identities [Di Cosmo 2005], one is tempted to think that the **Int** construction (as its name suggests) produces a categorification of the integers. Based on this hypothesis, the definitions above can be intuitively understood as arithmetic identities. The same arithmetic intuition explains the lifting of isomorphisms to 1d types:

$$(\tau_1 - \tau_2) \Leftrightarrow (\tau_3 - \tau_4) \quad \triangleq \quad (\tau_1 + \tau_4) \leftrightarrow (\tau_2 + \tau_3)$$

In other words, an isomorphism between 1d types is really an isomorphism between "re-arranged" 0d types where the negative input τ_2 is viewed as an output and the negative output τ_4 is viewed as an input. Using these ideas, it is now a fairly standard exercise to define the lifted versions of most of the combinators in Table 1.[3] There are however a few interesting cases whose appreciation is essential for the remainder of the paper that we discuss below.

[3] See Krishnaswami's [2012] excellent blog post implementing this construction in OCaml.

Easy Lifting. Many of the 0d combinators lift easily to the 1d level. For example:

$$\begin{array}{rcl}
id & : & \mathbb{T} \Leftrightarrow \mathbb{T} \\
& : & (\tau_1 - \tau_2) \Leftrightarrow (\tau_1 - \tau_2) \\
& \triangleq & (\tau_1 + \tau_2) \leftrightarrow (\tau_2 + \tau_1) \\
id & = & swap_+ \\
\\
identl_+ & : & \mathbb{0} \boxplus \mathbb{T} \Leftrightarrow \mathbb{T} \\
& = & assocr_+ \fatsemi (id \oplus swap_+) \fatsemi assocl_+
\end{array}$$

Composition using $trace$.

$$\begin{array}{rl}
(\fatsemi) : & (\mathbb{T}_1 \Leftrightarrow \mathbb{T}_2) \rightarrow (\mathbb{T}_2 \Leftrightarrow \mathbb{T}_3) \rightarrow (\mathbb{T}_1 \Leftrightarrow \mathbb{T}_3) \\
f \fatsemi g = & trace\ (assoc_1 \fatsemi (f \oplus id) \fatsemi assoc_2 \fatsemi (g \oplus id) \fatsemi assoc_3)
\end{array}$$

New combinators $curry$ ***and*** $uncurry$ ***for higher-order functions.***

$$\begin{array}{rcl}
\boxminus(\tau_1 - \tau_2) & \triangleq & \tau_2 - \tau_1 \\
(\tau_1 - \tau_2) \multimap (\tau_3 - \tau_4) & \triangleq & \boxminus(\tau_1 - \tau_2) \boxplus (\tau_3 - \tau_4) \\
& \triangleq & (\tau_2 + \tau_3) - (\tau_1 + \tau_4)
\end{array}$$

$$\begin{array}{rcl}
flip & : & (\mathbb{T}_1 \Leftrightarrow \mathbb{T}_2) \rightarrow (\boxminus\mathbb{T}_2 \Leftrightarrow \boxminus\mathbb{T}_1) \\
flip\ f & = & swap_+ \fatsemi f \fatsemi swap_+ \\
\\
curry & : & ((\mathbb{T}_1 \boxplus \mathbb{T}_2) \Leftrightarrow \mathbb{T}_3) \rightarrow (\mathbb{T}_1 \Leftrightarrow (\mathbb{T}_2 \multimap \mathbb{T}_3)) \\
curry\ f & = & assocl_+ \fatsemi f \fatsemi assocr_+ \\
\\
uncurry & : & (\mathbb{T}_1 \Leftrightarrow (\mathbb{T}_2 \multimap \mathbb{T}_3)) \rightarrow ((\mathbb{T}_1 \boxplus \mathbb{T}_2) \Leftrightarrow \mathbb{T}_3) \\
uncurry\ f & = & assocr_+ \fatsemi f \fatsemi assocl_+
\end{array}$$

The "phony" multiplication that is not a functor. The definition for the product of 1d types used above is:

$$\begin{array}{l}
(\tau_1 - \tau_2) \boxtimes (\tau_3 - \tau_4) = \\
((\tau_1 * \tau_3) + (\tau_2 * \tau_4)) - ((\tau_1 * \tau_4) + (\tau_2 * \tau_3))
\end{array}$$

That definition is "obvious" in some sense as it matches the usual understanding of types as modeling arithmetic. Using it, it is possible to lift all the 0d combinators involving products *except* the functor:

$$(\otimes) : (\mathbb{T}_1 \Leftrightarrow \mathbb{T}_2) \rightarrow (\mathbb{T}_3 \Leftrightarrow \mathbb{T}_4) \rightarrow ((\mathbb{T}_1 \boxtimes \mathbb{T}_3) \Leftrightarrow (\mathbb{T}_2 \boxtimes \mathbb{T}_4))$$

After a few failed attempts, we suspected that this definition of multiplication is not functorial which would mean that the **Int** construction provides a limited notion of higher-order functions at the expense of losing the multiplicative structure at higher-levels.

This observation is less well-known that it should be. Further investigation reveals that this observation is intimately related to a well-known problem in algebraic topology that was identified thirty years ago as the "phony" multiplication [Thomason 1980] in a special class categories related to ours. This problem was recently solved [Baas et al. 2012] using a technique whose fundamental ingredient is to add more dimensions. We exploit this idea in the remainder of the paper.

4. Cubes

As hinted at in the previous section, one can think of the **Int** construction as generalizing conventional 0d types to 1d types indexed by a positive or negative polarity. We will generalize this idea further in steps. The first step is to extend from 1d to arbitrary dimensions without consideration for polarities at this point. The extension is somewhat tedious but the idea is fundamentally simple as everything is defined pointwise.

4.1 Syntax

The set of types is now indexed by a dimension n:

$$\begin{array}{rcl} \tau, \mathbb{T}^0 & ::= & 0 \mid 1 \mid \tau_1 + \tau_2 \mid \tau_1 * \tau_2 \\ \mathbb{T}^{n+1} & ::= & \boxed{\mathbb{T}_1^n \mid \mathbb{T}_2^n} \end{array}$$

At dimension 0, we have the usual first-order types τ representing "points." At higher dimensions, a type is a pair of subspaces $\boxed{\mathbb{T}_1 \mid \mathbb{T}_2}$, each of a lower dimension. Like in the case for the **Int** construction, a 1d cube, $\boxed{\tau_1 \mid \tau_2}$, intuitively corresponds to the difference $\tau_1 - \tau_2$ of the two types. The type can be visualized as a "line." A 2d cube, $\boxed{(\boxed{\tau_1 \mid \tau_2}) \mid (\boxed{\tau_3 \mid \tau_4})}$, intuitively corresponds to the iterated difference of the types $(\tau_1 - \tau_2) - (\tau_3 - \tau_4)$ where the successive "colors" from the outermost box encode the signs. The type can be visualized as a "square" connecting the two lines corresponding to $(\tau_1 - \tau_2)$ and $(\tau_3 - \tau_4)$. (See Fig. 1 which is further explained after we discuss multiplication below.)

Even though the type constants 0 and 1 and the sums and products operations are only defined at dimension 0, cubes of all dimensions inherit this structure. We have constants $\mathbb{0}^n$ and $\mathbb{1}^n$ at every dimension and we also have families of sum $\boxplus^n$ and product $\boxtimes^m_n$ operations on higher dimensional cubes. The sum operation $\boxplus^n$ takes two n-dimensional cubes and produces another n-dimensional cube. Note that for the case of 1d types, the definition coincides with the one used in the **Int** construction. The product operation $\boxtimes^m_n$ takes two cubes of dimensions m and n respectively and as confirmed in Prop. 4.1 below and illustrated with a small example in Fig. 1, produces a cube of dimension $m + n$.

$$\begin{array}{rcl} \mathbb{0}^0 & = & 0 \\ \mathbb{0}^{n+1} & = & \boxed{\mathbb{0}^n \mid \mathbb{0}^n} \\ & & \\ \mathbb{1}^0 & = & 1 \\ \mathbb{1}^{n+1} & = & \boxed{\mathbb{1}^n \mid \mathbb{0}^n} \\ & & \\ \tau_1 \boxplus^0 \tau_2 & = & \tau_1 + \tau_2 \\ (\boxed{\mathbb{T}_1^n \mid \mathbb{T}_2^n}) \boxplus^{n+1} (\boxed{\mathbb{T}_3^n \mid \mathbb{T}_4^n}) & = & \boxed{\mathbb{T}_1^n \boxplus^n \mathbb{T}_3^n \mid \mathbb{T}_2^n \boxplus^n \mathbb{T}_4^n} \\ & & \\ \tau_1 \boxtimes^0_0 \tau_2 & = & \tau_1 * \tau_2 \\ \tau \boxtimes^0_{n+1} (\boxed{\mathbb{T}_1^n \mid \mathbb{T}_2^n}) & = & \boxed{\tau \boxtimes^0_n \mathbb{T}_1 \mid \tau \boxtimes^0_n \mathbb{T}_2} \\ (\boxed{\mathbb{T}_1^m \mid \mathbb{T}_2^m}) \boxtimes^{m+1}_n \mathbb{T}^n & = & \boxed{\mathbb{T}_1^m \boxtimes^m_n \mathbb{T}^n \mid \mathbb{T}_2^m \boxtimes^m_n \mathbb{T}^n} \end{array}$$

Proposition 4.1 (Dimensions)**.** *The dimension of $\mathbb{T}^m \boxtimes^m_n \mathbb{T}^n$ is $m + n$.*

Proof. By a simple induction on m. □

4.2 Isomorphisms

The main point of the generalization to arbitrary dimensions beyond the **Int** construction is that all higher-dimensions would have the full computational power of Π. In other words, all type isomorphisms (including the ones involving products) should lift to the higher dimensions. The lifting is surprisingly simple: everything is defined pointwise. As an example, there is an isomorphism between $\boxed{(\boxed{\tau_1 \mid \tau_2}) \mid (\boxed{\tau_3 \mid \tau_4})}$ and $\boxed{(\boxed{\tau_1' \mid \tau_2'}) \mid (\boxed{\tau_3' \mid \tau_4'})}$ if there are 0-dimensional Π-isomorphisms between each of the corresponding τ_i and τ_i'.

Table 2 replicates all the Π-combinators for higher-dimensional cubical types with a small modification: all the combinators and isomorphisms are indexed by the appropriate dimensions. In addition, we add explicit combinators to mediate between all the representations of the empty at different dimensions (and similarly for the unit type). Note that in general it is possible to have isomorphisms between types of different dimensions.

It is possible to embed any type $\mathbb{T}^m$ into several equivalent types of a higher dimension $n > m$ by padding it with empty types in the appropriate subspaces. For example, consider the 1d type $\boxed{\tau_1 \mid \tau_2}$ and the following embedding in 2d types:

$$\begin{array}{cl} & \boxed{\tau_1 \mid \tau_2} \\ \mathit{identr}_+^{0,1} & \mathbb{1}^0 \boxtimes^0_1 \boxed{\tau_1 \mid \tau_2} \\ (\mathit{raise1}^0 \otimes^{0,1,1,1} \mathit{id}^1) & \mathbb{1}^1 \boxtimes^1_1 \boxed{\tau_1 \mid \tau_2} \\ = & \boxed{(\boxed{1 * \tau_1 \mid 1 * \tau_2}) \mid (\boxed{0 * \tau_1 \mid 0 * \tau_2})} \\ & \boxed{(\boxed{\tau_1 \mid \tau_2}) \mid (\boxed{0 \mid 0})} \end{array}$$

If we simply introduce a swap after the second step, the resulting embedding is:

$$\boxed{(\boxed{\tau_1 \mid 0}) \mid (\boxed{\tau_2 \mid 0})}$$

More generally it is possible to embed a type of dimension n in $n + 1$ different ways into a type of dimension $n + 1$.

formalize that n-dim comb
are families of pi-comb

formalize embedding on
n-dim types into
n+1-dim types; I think we
can get all of the embeddings
into the various faces by
use T x 1 or 1 x T
recursively at each level

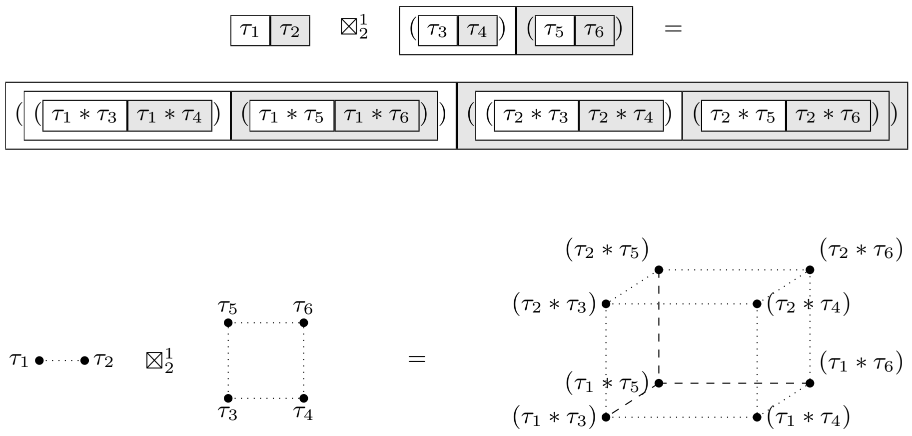

Figure 1. Example of multiplication of two cubical types.

$$
\begin{array}{rrcll}
\mathit{identl}^n_+ : & \mathbb{0}^n \boxplus^n \mathbb{T}^n & \Leftrightarrow & \mathbb{T}^n & : \mathit{identr}^n_+ \\
\mathit{swap}^n_+ : & \mathbb{T}^n_1 \boxplus^n \mathbb{T}^n_2 & \Leftrightarrow & \mathbb{T}^n_2 \boxplus^n \mathbb{T}^n_1 & : \mathit{swap}^n_+ \\
\mathit{assocl}^n_+ : & \mathbb{T}^n_1 \boxplus^n (\mathbb{T}^n_2 \boxplus^n \mathbb{T}^n_3) & \Leftrightarrow & (\mathbb{T}^n_1 \boxplus^n \mathbb{T}^n_2) \boxplus^n \mathbb{T}^n_3 & : \mathit{assocr}^n_+ \\
\mathit{identl}^{m,n}_* : & \mathbb{1}^m \boxtimes^m_n \mathbb{T}^n & \Leftrightarrow & \mathbb{T}^n & : \mathit{identr}^{m,n}_* \\
\mathit{swap}^{m,n}_* : & \mathbb{T}^m_1 \boxtimes^m_n \mathbb{T}^n_2 & \Leftrightarrow & \mathbb{T}^n_2 \boxtimes^n_m \mathbb{T}^m_1 & : \mathit{swap}^{m,n}_* \\
\mathit{assocl}^{m,n,k}_* : & \mathbb{T}^m_1 \boxtimes^m_{n+k} (\mathbb{T}^n_2 \boxtimes^n_k \mathbb{T}^k_3) & \Leftrightarrow & (\mathbb{T}^m_1 \boxtimes^m_n \mathbb{T}^n_2) \boxtimes^{m+n}_k \mathbb{T}^k_3 & : \mathit{assocr}^{m,n,k}_* \\
\mathit{dist}^{m,n}_0 : & \mathbb{0}^m \boxtimes^m_n \mathbb{T}^n & \Leftrightarrow & \mathbb{0}^m & : \mathit{factor}^{m,n}_0 \\
\mathit{dist}^{m,n} : & (\mathbb{T}^m_1 \boxplus^m \mathbb{T}^m_2) \boxtimes^m_n \mathbb{T}^n_3 & \Leftrightarrow & (\mathbb{T}^m_1 \boxtimes^m_n \mathbb{T}^n_3) \boxplus^{m+n} (\mathbb{T}^m_2 \boxtimes^m_n \mathbb{T}^n_3) & : \mathit{factor}^{m,n} \\
\mathit{raise0}^n : & \mathbb{0}^n & \Leftrightarrow & \mathbb{0}^{n+1} & : \mathit{lower0}^n \\
\mathit{raise1}^n : & \mathbb{1}^n & \Leftrightarrow & \mathbb{1}^{n+1} & : \mathit{lower1}^n
\end{array}
$$

$$
\frac{}{\vdash \mathit{id}^n : \mathbb{T}^n \Leftrightarrow \mathbb{T}^n}
\qquad
\frac{\vdash c : \mathbb{T}^m_1 \Leftrightarrow \mathbb{T}^n_2}{\vdash \mathit{sym}^{n,m}\ c : \mathbb{T}^n_2 \Leftrightarrow \mathbb{T}^m_1}
$$

$$
\frac{\vdash c_1 : \mathbb{T}^m_1 \Leftrightarrow \mathbb{T}^n_2 \quad \vdash c_2 : \mathbb{T}^n_2 \Leftrightarrow \mathbb{T}^k_3}{\vdash c_1 \fatsemi^{m,n,k} c_2 : \mathbb{T}^m_1 \Leftrightarrow \mathbb{T}^k_3}
$$

$$
\frac{\vdash c_1 : \mathbb{T}^m_1 \Leftrightarrow \mathbb{T}^n_2 \quad \vdash c_2 : \mathbb{T}^m_3 \Leftrightarrow \mathbb{T}^n_4}{\vdash c_1 \oplus^{m,n} c_2 : \mathbb{T}^m_1 \boxplus^m \mathbb{T}^m_3 \Leftrightarrow \mathbb{T}^n_2 \boxplus^n \mathbb{T}^n_4}
$$

$$
\frac{\vdash c_1 : \mathbb{T}^m_1 \Leftrightarrow \mathbb{T}^n_2 \quad \vdash c_2 : \mathbb{T}^k_3 \Leftrightarrow \mathbb{T}^p_4}{\vdash c_1 \otimes^{m,n,k,p} c_2 : \mathbb{T}^m_1 \boxtimes^m_k \mathbb{T}^k_3 \Leftrightarrow \mathbb{T}^n_2 \boxtimes^n_p \mathbb{T}^p_4}
$$

Table 2. Combinators on cubical types

4.3 Operational Semantics

Because everything is defined pointwise, the new combinators are just families of base combinators. As an example, consider:

$$
\begin{array}{rl}
\mathit{swap}^2_+ : & ((\tau_1 \; \tau_2) \; (\tau_3 \; \tau_4)) \boxplus^2 ((\tau'_1 \; \tau'_2) \; (\tau'_3 \; \tau'_4)) \\
\Leftrightarrow & ((\tau'_1 \; \tau'_2) \; (\tau'_3 \; \tau'_4)) \boxplus^2 ((\tau_1 \; \tau_2) \; (\tau_3 \; \tau_4)) \\
 & \\
: & ((\tau_1 + \tau'_1 \;\; \tau_2 + \tau'_2) \; (\tau_3 + \tau'_3 \;\; \tau_4 + \tau'_4)) \\
\Leftrightarrow & ((\tau'_1 + \tau_1 \;\; \tau'_2 + \tau_2) \; (\tau'_3 + \tau_3 \;\; \tau'_4 + \tau_4)) \\
 & \\
= & ((\mathit{swap}_+ \;\; \mathit{swap}_+) \; (\mathit{swap}_+ \;\; \mathit{swap}_+))
\end{array}
$$

In other words, the definition of swap_+ at a 2d dimensional type reduces to a family of 0-dimensional swap_+ Π-combinators at each of the vertices of the 2d cube.

```
Implement op. sem. in Agda.
Basically a version of pi indexed
by dimensions!

Check and implement all the
various isos: embeddings of
n-dim into n+1-dim etc.

need to make sure there are
no other isos implied by
the development in secs 2
and 3; and that we don't have
```

```
any isos that are not justified
by the math
```

5. Polarities

```
need to understand and then add
the t-t=0 equations

finally need to move on to
sec. 4 and beyond
and understand what they
add in terms of isos
```

6. Related Work and Context

A ton of stuff here.

7. Conclusion

References

S. Abramsky. Retracing some paths in process algebra. In *CONCUR '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1996.

S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. In *LICS*, 2004.

N. Baas, B. Dundas, B. Richter, and J. Rognes. Ring completion of rig categories. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 2013(674):43–80, Mar. 2012.

J. C. Baez and J. Dolan. Categorification. In Higher Category Theory, Contemp. Math. 230, 1998, pp. 1-36., 1998.

C. Bennett. Notes on Landauer's principle, reversible computation, and Maxwell's Demon. *Studies In History and Philosophy of Science Part B: Studies In History and Philosophy of Modern Physics*, 34(3):501–510, 2003.

C. Bennett. Notes on the history of reversible computation. *IBM Journal of Research and Development*, 32(1):16–23, 2010.

C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17: 525–532, November 1973.

W. J. Bowman, R. P. James, and A. Sabry. Dagger Traced Symmetric Monoidal Categories and Reversible Programming. In *RC*, 2011.

R. Di Cosmo. A short survey of isomorphisms of types. *Mathematical. Structures in Comp. Sci.*, 15(5):825–838, Oct. 2005. ISSN 0960-1295. doi: 10.1017/S0960129505004871. URL `http://dx.doi.org/10.1017/S0960129505004871`.

M. Fiore. Isomorphisms of generic recursive polynomial types. In *POPL*, pages 77–88. ACM, 2004.

M. P. Fiore, R. Di Cosmo, and V. Balat. Remarks on isomorphisms in typed calculi with empty and sum types. *Annals of Pure and Applied Logic*, 141(1-2):35–50, 2006.

E. Fredkin and T. Toffoli. Conservative logic. *International Journal of Theoretical Physics*, 21(3):219–253, 1982.

R. P. James and A. Sabry. Information effects. In *POPL*, pages 73–84. ACM, 2012.

A. Joyal, R. Street, and D. Verity. Traced monoidal categories. In *Mathematical Proceedings of the Cambridge Philosophical Society*. Cambridge Univ Press, 1996.

N. Krishnaswami. The geometry of interaction, as an OCaml program. `http://semantic-domain.blogspot.com/2012/11/in-this-post-ill-show-how-to-turn.html`, 2012.

R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183–191, July 1961.

R. Landauer. The physical nature of information. *Physics Letters A*, 1996.

M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. `http://homotopytypetheory.org/book`, Institute for Advanced Study, 2013.

R. Thomason. Beware the phony multiplication on Quillen's $\mathcal{A}^{-1}\mathcal{A}$. *Proc. Amer. Math. Soc.*, 80(4):569–573, 1980.

T. Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, pages 632–644. Springer-Verlag, 1980.