# 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 $\Pi$ whose only computations are isomorphisms between finite types. The set of types $\tau$ 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:

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

The interesting syntactic category of $\Pi$ 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 $\mathit{trace}$ 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 $\Pi$, 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 $\Pi$ 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 $\Pi$-combinators are precisely the semiring axioms. A formal way of saying this is that $\Pi$ 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 $\Pi$-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 $\Pi$-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 $\Pi$ 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 $\tau_1$ and $\tau_2$ are objects in the base category. Intuitively, the component $\tau_1$ is viewed as a conventional type whose elements represent values flowing, as usual, from producers to consumers. The component $\tau_2$ is viewed as a *negative type* whose elements represent demands for values or equivalently values flowing backwards. Under this interpretation,

[1] where $\mathit{swap}_+$ and $\mathit{swap}_*$ 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.

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

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

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

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

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

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

**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 $\tau$ 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 $\tau_2$ is viewed as an output and the negative output $\tau_4$ 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*** $trace$***.***

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

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

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

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

***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:

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

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 (0-dimensional) types to a higher-dimension. The types in the higher-dimension are indexed by a polarity which specifies their position in a 1-dimensional space. Generalizing this idea further, we view types as being indexed by a dimension $n$ and a position in the corresponding $n$-dimensional space.

### 4.1 Indexing

We will refer to two kinds of $n$-dimensional cubical diagrams with $2^n$ and $3^n$ vertices respectively. These diagrams will be indexed by the categories $\mathcal{P}\mathbf{n}$ and $\mathcal{Q}\mathbf{n}$ defined below. The category $\mathcal{P}\mathbf{n}$ has subsets of $\mathbf{n} = \{1, \ldots, n\}$ as objects and set inclusions as morphisms. The category $\mathcal{Q}\mathbf{n}$ has subsets of $\{\pm 1, \ldots, \pm n\}$ in which no number appears with both positive and negative polarities as objects and set inclusions as morphisms. Both $\mathcal{P}\mathbf{0}$ and $\mathcal{Q}\mathbf{0}$ consist of the category with one object, the empty set. The category $\mathcal{Q}\mathbf{1}$ is depicted below:

$$\{-1\} \longleftarrow \emptyset \longrightarrow \{1\}$$

It consists of 3 objects. The category $\mathcal{P}\mathbf{1}$ consists of the right part of the diagram. The category $\mathcal{Q}\mathbf{2}$ is depicted below:

$$\begin{tikzcd}
\{-1,2\} & \{2\} \arrow[l] \arrow[r] & \{1,2\} \\
\{-1\} \arrow[u] \arrow[d] & \emptyset \arrow[l] \arrow[r] \arrow[u] \arrow[d] & \{1\} \arrow[u] \arrow[d] \\
\{-1,-2\} & \{-2\} \arrow[l] \arrow[r] & \{1,-2\}
\end{tikzcd}$$

The category $\mathcal{P}\mathbf{2}$ is the upper right square.

Our types will be indexed by objects from the categories $\mathcal{Q}\mathbf{n}$, i.e., by pairs $(\mathbf{n}, S)$ specifying the category $\mathcal{Q}\mathbf{n}$ and the selected object in it. Examples of such indexing objects are $(\mathbf{0}, \emptyset)$, $(\mathbf{1}, \{-1\})$ which is the object $\{-1\}$ in $\mathcal{Q}\mathbf{1}$, $(\mathbf{2}, \{-1\})$ which is also the object $\{-1\}$ but in $\mathcal{Q}\mathbf{2}$, and $(\mathbf{2}, \{-1,-2\})$ which is the object $\{-1,-2\}$ in $\mathcal{Q}\mathbf{2}$. By the Grothendieck construction [Barr and Wells 1995, Ch.12], these indexing objects form a category with morphisms from $(\mathbf{n}, S)$ to $(\mathbf{m}, T)$ if there is an injection between the finite sets $\mathbf{n}$ and $\mathbf{m}$ that maps $S$ to a subset of $T$. For example, there are two ways of embedding $\mathcal{Q}\mathbf{1}$ into $\mathcal{Q}\mathbf{2}$ determined by how one chooses to inject the set $\{1\}$ into the set $\{1,2\}$. One can choose to send the element 1 in the first set to either the element 1 or the element 2 in the second set. The first map embeds $\mathcal{Q}\mathbf{1}$ into the top row of $\mathcal{Q}\mathbf{2}$ while the second map embeds $\mathcal{Q}\mathbf{1}$ into the right column of $\mathcal{Q}\mathbf{2}$. Each of these possibilities gives a morphism between $(\mathbf{1}, \{-1\})$ and $(\mathbf{2}, \{-1,2\})$.

### 4.2 The Cube Construction

Having defined our indexing objects, we now proceed to define the generalized $n$-dimensional types. We do this in two stages, first for the special case of indexing objects in $\mathcal{P}\mathbf{n}$ and then for general indexing objects in $\mathcal{Q}\mathbf{n}$.

Given $(\mathbf{n}, S)$ for the special case of $S \in \mathcal{P}\mathbf{n}$, we construct a $n$-dimensional extension of $\Pi$ denoted $\Pi\mathbf{n}(S)$. The types in $\Pi\mathbf{n}(S)$ are as many copies as there are subsets of $S$ of the types in $\Pi$ labeled by the subsets of $S$. For example, a type in $\Pi\mathbf{2}(\emptyset)$ is just a plain $\Pi$ type (technically indexed by $\emptyset$); a type in $\Pi\mathbf{2}(\{1\})$ is a collection of two $\Pi$-types indexed by $\emptyset$ and $\{1\}$; a type in $\Pi\mathbf{2}(\{2\})$ is also a collection of two $\Pi$-types but indexed by $\emptyset$ and $\{2\}$; and finally a type in $\Pi\mathbf{2}(\{1,2\})$ is a collection of four $\Pi$-types indexed by $\emptyset$, $\{1\}$, $\{2\}$, and $\{1,2\}$. In general, for a given $\mathbf{n}$, there are $2^n$ versions of $\Pi\mathbf{n}$ viewed as being spread out over the corners of an $n$-dimensional cube. The various copies of $\Pi\mathbf{n}(S)$ can be embedded in one another following the morphisms in the indexing category. Thus the two $\Pi$-types in $\Pi\mathbf{2}(\{1\})$ embed in the four $\Pi$-types in $\Pi\mathbf{2}(\{1,2\})$ as follows. We get two copies of the type indexed by $\emptyset$; one indexed by $\emptyset$ and one indexed by $\{2\}$; and we get two copies of the type indexed by $\{1\}$; one indexed by $\{1\}$ and one indexed by $\{1,2\}$.

$$\begin{tikzcd}
\begin{array}{l}\Pi\mathbf{2}(\{2\}) = \\ \Pi_\emptyset \times \Pi_{\{2\}}\end{array} \arrow[r] & \begin{array}{l}\Pi\mathbf{2}(\{1,2\}) = \Pi_\emptyset \times \\ \Pi_{\{1\}} \times \Pi_{\{2\}} \times \Pi_{\{1,2\}}\end{array} \\
\begin{array}{l}\Pi\mathbf{2}(\emptyset) = \\ \Pi_\emptyset\end{array} \arrow[r] \arrow[u] & \begin{array}{l}\Pi\mathbf{2}(\{1\}) = \\ \Pi_\emptyset \times \Pi_{\{1\}}\end{array} \arrow[u]
\end{tikzcd}$$

Formally the set of types in $\Pi\mathbf{n}(S)$ is:

$$\begin{array}{rcll}
\tau\mathbf{n}(S) & ::= & \bigotimes_n \{\tau_s \mid s \text{ subset of } S\} & \\
& \mid & \tau\mathbf{n}(S) + \tau\mathbf{n}(S) & \\
& \mid & \tau\mathbf{i}(S) * \tau\mathbf{j}(S) & \text{where } n = i + j \\
& \mid & \mathit{embed}(\mathbf{m} \mapsto \mathbf{n})\tau\mathbf{m}(T) &
\end{array}$$

indexing by qn
show functors on the pi-n types and then isos

We present the syntax of generalized $n$-dimensional types and isomorphisms between them.

### 4.3 Negative and Cubical Types

Our types $\mathbb{T}$ are "cubes" defined as follows:

$$
\begin{array}{rcl}
\tau & ::= & 0 \mid 1 \mid \tau_1 + \tau_1 \mid \tau_1 * \tau_2 \\
\mathbb{T} & ::= & \tau \mid \boxed{\mathbb{T}_1 \,\big|\, \mathbb{T}_2} \mid \mathbb{T}_1 \boxplus \mathbb{T}_2 \mid \mathbb{T}_1 \boxtimes \mathbb{T}_2 \mid \boxminus \mathbb{T}
\end{array}
$$

The *dimension* of a type is defined as follows:

$$
\begin{array}{rcl}
\mathsf{dim}(\cdot) & :: & \tau \rightarrow \mathbb{N} \\
\mathsf{dim}(0) & = & 0 \\
\mathsf{dim}(1) & = & 0 \\
\mathsf{dim}(\tau_1 + \tau_2) & = & \max(\mathsf{dim}(\tau_1), \mathsf{dim}(\tau_2)) \\
\mathsf{dim}(\tau_1 * \tau_2) & = & \mathsf{dim}(\tau_1) + \mathsf{dim}(\tau_2) \\
\mathsf{dim}(-\tau) & = & \max(1, \mathsf{dim}(\tau))
\end{array}
$$

The base types have dimension 0. If negative types are not used, all dimensions remain at 0. If negative types are used but no products of negative types appear anywhere, the dimension is raised to 1. This is the situation with the **Int** or $\mathcal{G}$ construction. Once negative and product types are freely used, the dimension can increase without bounds.

This point is made precise in the following denotation of types which maps a type of dimension $n$ to an $n$-dimensional cube. We represent such a cube syntactically as a binary tree of maximum depth $n$ with nodes of the form $\boxed{\mathbb{T}_1 \,\big|\, \mathbb{T}_2}$. In such a node, $\mathbb{T}_1$ is the positive subspace and $\mathbb{T}_2$ (shaded in gray) is the negative subspace along the first dimension. Each of these subspaces is itself a cube of a lower dimension. The 0-dimensional cubes are plain sets representing the denotation of conventional first-order types. We use $S$ to denote the denotations of these plain types. A 1-dimensional cube, $\boxed{S_1 \,\big|\, S_2}$, intuitively corresponds to the difference $\tau_1 - \tau_2$ of the two types whose denotations are $S_1$ and $S_2$ respectively. The type can be visualized as a "line" with polarized endpoints connecting the two points $S_1$ and $S_2$. A full 2-dimensional cube, $\boxed{(\boxed{S_1 \,\big|\, S_2}) \,\big|\, (\boxed{S_3 \,\big|\, S_4})}$, intuitively corresponds to the iterated difference of the appropriate types $(\tau_1 - \tau_2) - (\tau_3 - \tau_4)$ where the successive "colors" from the outermost box encode the sign. The type can be visualized as a "square" with polarized corners 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.)

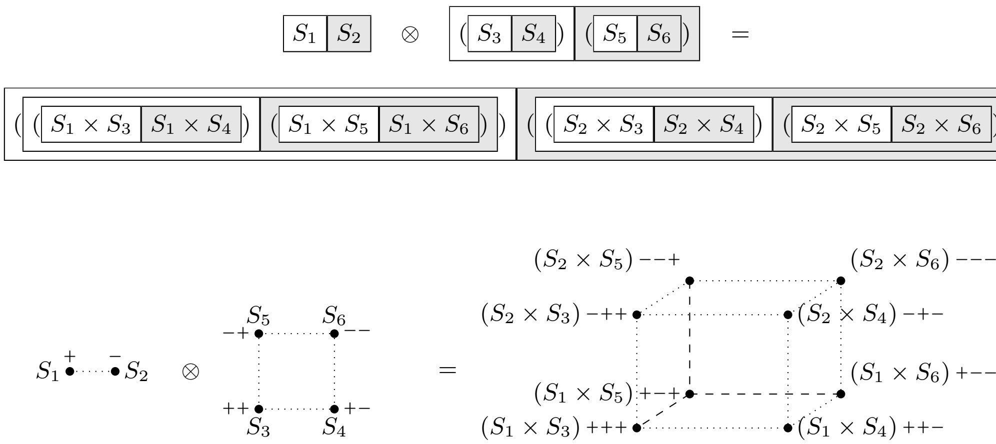

**Figure 1.** Example of multiplication of two cubical types.

Formally, the denotation of types discussed so far is as follows:

$$\begin{array}{rcl} [\![0]\!] & = & \emptyset \\ [\![1]\!] & = & \{\star\} \\ [\![\tau_1 + \tau_2]\!] & = & [\![\tau_1]\!] \oplus [\![\tau_2]\!] \\ [\![\tau_1 * \tau_2]\!] & = & [\![\tau_1]\!] \otimes [\![\tau_2]\!] \\ [\![-\tau]\!] & = & \ominus [\![\tau]\!] \end{array}$$

where:

$$\begin{array}{rcl} S_1 \oplus S_2 & = & S_1 \uplus S_2 \\ S \oplus (\boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2}) & = & \boxed{S \oplus \mathbb{T}_1}\boxed{\mathbb{T}_2} \\ (\boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2}) \oplus S & = & \boxed{\mathbb{T}_1 \oplus S}\boxed{\mathbb{T}_2} \\ (\boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2}) \oplus (\boxed{\mathbb{T}_3}\boxed{\mathbb{T}_4}) & = & \boxed{\mathbb{T}_1 \oplus \mathbb{T}_3}\boxed{\mathbb{T}_2 \oplus \mathbb{T}_4} \end{array}$$

$$\begin{array}{rcl} S_1 \otimes S_2 & = & S_1 \times S_2 \\ S \otimes (\boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2}) & = & \boxed{S \otimes \mathbb{T}_1}\boxed{S \otimes \mathbb{T}_2} \\ (\boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2}) \otimes \mathbb{T} & = & \boxed{\mathbb{T}_1 \otimes \mathbb{T}}\boxed{\mathbb{T}_2 \otimes \mathbb{T}} \end{array}$$

$$\begin{array}{rcl} \ominus S & = & \boxed{\phantom{S}}\boxed{S} \\ \ominus \boxed{\mathbb{T}_1}\boxed{\mathbb{T}_2} & = & \boxed{\ominus \mathbb{T}_2}\boxed{\ominus \mathbb{T}_1} \end{array}$$

The type 0 maps to the empty set. The type 1 maps to a singleton set. The sum of 0-dimensional types is the disjoint union as usual. For cubes of higher dimensions, the subspaces are recursively added. Note that the sum of 1-dimensional types reduces to the sum used in the **Int** construction. The definition of negation is natural: it recursively swaps the positive and negative subspaces. The product of 0-dimensional types is the cartesian product of sets. For cubes of higher-dimensions $n$ and $m$, the result is of dimension $(n + m)$. The example in Fig. 1 illustrates the idea using the product of 1-dimensional cube (i.e., a line) with a 2-dimensional cube (i.e., a square). The result is a 3-dimensional cube as illustrated.

## 5. Related Work and Context

A ton of stuff here.

## 6. Conclusion

## References


S. Abramsky. Retracing some paths in process algebra. In U. Montanari and V. Sassone, editors, *CONCUR '96: Concurrency Theory*, volume 1119 of *Lecture Notes in Computer Science*, pages 1–17. Springer Berlin Heidelberg, 1996. ISBN 978-3-540-61604-7. doi: 10.1007/3-540-61604-7_44. URL `http://dx.doi.org/10.1007/3-540-61604-7_44`.

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.

M. Barr and C. Wells. *Category theory for computing science (2. ed.)*. Prentice Hall international series in computer science. Prentice Hall, 1995. ISBN 978-0-13-323809-9.

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.