Representing Modular Tensor Categories: A Computer Algebra System for Topological Quantum Computing

Daniel Adam Roberts

New College

Computing Laboratory

University of Oxford

A dissertation for the degree of

Master of Science by research

Trinity 2011

Representing Modular Tensor Categories:

A Computer Algebra System for Topological Quantum Computing

Daniel A. Roberts

New College

M.Sc. by research

Trinity 2011

Abstract

Topological quantum computation (TQC) is a new fault-tolerant approach to quantum information, where computations are carried out by braiding particles called anyons. Anyons are quasiparticles that exist in $2 + 1$ dimensions and are neither bosons or fermions. Modular tensor categories capture the structure of anyon systems and thus serve as models for topological quantum computation. However, program of using category theory to find higher-level structures and protocols has yet to be applied to TQC due to the difficulty of working with modular tensor categories. This difficulty could be greatly mitigated by the development of a computer algebra system to represent such categories. Thus, a computer algebra system for representing modular tensor categories within the symmetric monoidal 2-category **2Vect** was developed.

A general representation for **2Vect** is described. This involves extending basic linear algebra operations to handle matrices of zero-dimension and 2-matrices (matrices of matrices). Then, representations for the 0-cells, 1-cells, and 2-cells of **2Vect** are found. Due to the chosen representation, various structural isomorphisms are required to ensure equations expected of 1-categories hold in the 2-categories setting. These structural isomorphisms are explicitly constructed.

In order to identify special categories, such as modular tensor categories, a diagrammatic language of monoidal categories is extended for 2-categories. This language is used to construct the axioms for these special categories within **2Vect**. Finally, a way of implementing these axioms within **2Vect** in the computer algebra system is described. This system will now be used for the investigation of higher-level structures within modular tensor categories to better understand the structure of topological quantum computing.

Contents

1 Introduction

Topological Quantum Computation (TQC) is an alternative program for quantum computation that has recently started to gain traction [18, 20, 24, 28, 31, 34, 30]. Opposing the industry-standard *local* two-level system qubit model [29], TQC proposes—as the name suggests—an entirely *topological* computation scheme. To perform computations, quasiparticles, called *nonabelian anyons*, are *braided* together in $2+1$ dimensions. The proposed *quantum computer* is naturally fault-tolerant—qubits do not require complex encodings or error correcting schemes—they are naturally resilient to the errors that plague the traditional model [28].

Category theory is an abstract branch of mathematics that generalizes the approach of set theory [3]. The program of category theory is to identify and unify similar structures across many diverse fields of mathematics and science, such as logic, physics, topology and computation [5]—and sometimes even outside the realm of traditional mathematics and science, such as linguistics [17]. A favored haunt of category theory is the quantum realm, where certain types of categories, monoidal categories, are perfect structures for representing compound systems. Recently, category theory has been applied to the foundations of *quantum mechanics* as a replacement for the von Neumann Hilbert space formalism [2]—to put quantum mechanics on proper axiomatic footing. Through its use of a high level of abstraction, category theory brings the fundamental structure of these fields into focus, providing, at worst, a nice new formalism in which to view and study branches of mathematics, and, at best, insight into the *deep connections* within math, philosophy, reality, and everything—essentially the 42-ness of it all [4].

In additional to foundational issues, category theory has been applied to *quantum information* as a means of developing a high-level language for *quantum computation* [1]. Not unlike the difference between mere bit-flipping and Python or Java, the categorical approach to quantum computation and information hopes to bring the field out of the proverbial complex-number-qubit-(well, not just flipping anymore but, more technically)-unitary-operating gutter [12, 13]. Coupled with this new formalism, a *diagrammatic language* developed by Joyal and Street [21] has been applied by Abramsky and Coecke [1] to simplify calculations and elucidate not only the *how* but also the *why* of quantum information theory. Concepts key to the quantum world, such as *no-cloning* and *teleportation*, are rendered intuitive, if not obvious, in the diagrammatic language.

Kitaev, who originally proposed TQC [24], also attempted to give it a categorical foundation [25]. A special type of category, the *modular tensor category* (MTC), has been shown to have the exact structure needed to model the kinematics and dynamics of anyons [31]. Thus, MTCs provide a model for TQC.

Unfortunately, MTCs are not simple categories[1] and are encumbered with structures and properties. It takes many pages of matrices to even test whether a proposed set of structures form a modular tensor category. Therefore, it is useful to develop a computer algebra representation of modular tensor categories. This representation would aid in identifying and finding such categories as well as allow the user to compute within them. Since MTCs model topological quantum computation, this would also allow the user to compute things about anyons or the underlying *topological quantum field theory* (TQFT) the category represents—essentially to aid in the exploration of TQC.

This computer model will be developed within the symmetric monoidal 2-category, **2Vect**. 2-categories are further generalizations in category theory and may be thought of as *categories of categories* [26]. In this paradigm, modular tensor categories are structures to be found *within* **2Vect**. In fact, the structure of **2Vect** is general enough to represent many different types of categories, not just MTCs. As such, this model will provide an explicit representation of **2Vect** and thus serve to elucidate its structures, which carries an interest in its own right. These structures must be explicitly constructed, and on the way, it was determined that certain equations of 1-categories only hold up to isomorphism in **2Vect**.

Additionally, the diagrammatic language was extended to symmetric monoidal 2-categories, where a special type of two-dimensional bracketing is necessary to keep equations "type safe." This allows us to easily write equations of the MTC in **2Vect**, and ensures (from a non-categorical viewpoint) that no "implicit change of basis" will disrupt our calculations.

Along the way, we will make discoveries about the structure of **2Vect** (as mentioned) and the different equivalent classes of MTCs that live inside it. In the end, we hope to use such a computer algebra representation to work with modular tensor categories and anyon models at the same high-level quantum protocols now enjoy due to category theory.

[1] Technically, MTCs are semisimple. However, here we really use *simple* to mean "of low complexity" or "not elaborate."

2 Theory

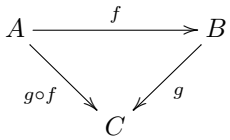
2.1 Modular Tensor Categories

The following provides a brief introduction to modular tensor categories. A *modular tensor category* is a twisted braided monoidal category that is semisimple, modular, and rigid. The braiding, the twist, and monoidality are all different *structures* that must be defined on top of the base category. Modularity, rigidity, and semi-simplicity are *properties* that a given category must satisfy. This presentation of categories, monoidal categories, and braided monoidal categories is adapted from Baez and Stay [5], the discussion of rigidity and the twist follows the approach of Turaev [36], and our approach to modularity follows Müger [27].

A category consists of a set of objects and a set of *arrows* or *morphisms* that go between the objects. We write arrows with their source and destination objects. For instance, $f : A \to B$ specifies a morphism, f, goes from object A to object B. This can be represented diagrammatically as

$$A \xrightarrow{f} B$$

Furthermore, a category has an associative binary operation that allows morphisms to be composed. If we have $g : B \to C$, in addition to f, then we also have $g \circ f : A \to C$. This is diagrammatically represented as



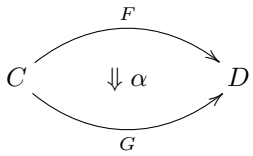
Finally, all objects have an identity morphism, for instance, $id_B : B \to B$, such that $g \circ id_B = g$ and $id_B \circ f = f$.

A category is, in many ways, an abstract generalization of a set. For the category of sets, **Set**, the objects would be the sets and arrows would be functions between the sets. However, category theory shifts the focus from the objects (or possible elements "within") to the arrows. The main focus of category theory is structures and the various operations that preserve structure.

We can also define maps between categories, called *functors*. For each object and morphism in the source category a functor provides a map to an object and morphism (respectively) in the

destination category. If a functor F maps from category C to category D, we write $F : C \to D$. Furthermore, if A and B are objects in C, and $f : A \to B$ is a morphism is C, then FA and FB are objects in D, and $Ff : FA \to FB$ is a morphism in D. Finally, functors preserve the composition of morphisms, so that $F(g \circ f) = Fg \circ Ff$. In the category of categories, **Cat**, the morphisms are functors. Thus, as morphisms they must satisfy the associativity and identity properties as defined above for arrows.

Furthermore, we can define *natural transformations* as maps between functors that go between the same categories. If we have functors $F : C \to D$ and $G : C \to D$, then we can have a natural transformation from F to G, $\alpha : F \Rightarrow G$. This can be represented diagrammatically as



where C and D are categories—or equivalently seen as objects in the category of categories. Furthermore, for every object in C, the natural transformation provides a morphism. For an object A in C, we have the morphism $\alpha_A : FA \to GA$. Furthermore, for any morphism $f : A \to B$ in C, we have $\alpha_B \circ Ff = Gf \circ \alpha_A$ as a true equation in category D. This equation can be represented as a diagram in D

$$\begin{array}{ccc} FA & \xrightarrow{Ff} & FB \\ {\scriptstyle \alpha_A}\downarrow & & \downarrow{\scriptstyle \alpha_B} \\ GA & \xrightarrow[Gf]{} & GB \end{array}$$

along with the statement that the diagram *commutes*—meaning that all paths through the diagram are equal. A equation (or diagram) such as this is known as a coherence condition. An invertible natural transformation is known as a *natural isomorphism*.

Examples of all these structures will be given in the forthcoming definitions, and furthermore, these concepts will be explained in greater detail within the structure of 2-categories in Section 2.2. Nevertheless, the essentials of category theory should be familiar to the reader—if not, the reader may consult Abramsky and Tzevelekos [3] for an introduction.

2.1.1 Monoidal Categories

A *monoidal category* (sometimes referred to as a *tensor category*) is a category equipped with a tensor product functor, usually written as $\otimes$, a unit object, denoted u, and three natural isomorphisms, the associator, the right unitor, and the left unitor, denoted α, ρ, and λ, respectively.

The product functor $\otimes$ is a *bifunctor*—it takes two objects, A and B from the underlying category and returns a combined object, $A \otimes B$. This monoidal combination can be thought of as *A and B*.

The components of the natural isomorphisms, α, ρ, and λ are

$$\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$$

$$\rho_A : A \otimes u \to A$$

$$\lambda_A : u \otimes A \to A$$

The associator mediates between the different ways of bracketing three objects, and the unitors single out the unit object—effectively making it the identity under $\otimes$. Finally, all of these structures must satisfy two equations or axioms.

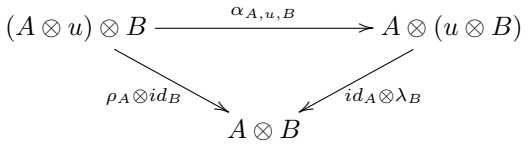
The *pentagon equation*—so aptly named due to the shape of its representative diagram—is a condition stating that the two ways to rebracket $((A \otimes B) \otimes C) \otimes D$ to $A \otimes (B \otimes (C \otimes D))$, built up from different combinations of α, must be equal. The pentagon equation is given by the statement that the following diagram

$$\begin{array}{ccc} & ((A \otimes B) \otimes C) \otimes D & \\ {\scriptstyle \alpha_{A,B,C} \otimes id_D} \swarrow & & \searrow {\scriptstyle \alpha_{A \otimes B,C,D}} \\ (A \otimes (B \otimes C)) \otimes D & & (A \otimes B) \otimes (C \otimes D) \\ \downarrow {\scriptstyle \alpha_{A,B \otimes C,D}} & & \downarrow {\scriptstyle \alpha_{A,B,C \otimes D}} \\ A \otimes ((B \otimes C) \otimes D) & \xrightarrow[id_A \otimes \alpha_{B,C,D}]{} & A \otimes (B \otimes (C \otimes D)) \end{array} \tag{1}$$

must commute—i.e. both ways of rebracketing must be equal. *Mac Lane's coherence theorem* can be applied, so that if the pentagon equation holds, then all ways of rebracketing 5 or 6 or N objects

must be equal [26].

Similarly, there is an equation that deals with coherence the two unitors and the associator, known as the *triangle equation.*[2] The triangle equation states that the following diagram must commute:



(2)

In this case, the coherence ensures that using the right unitor is the same as rebracketing and using the left unitor.

Monoidal categories are a generalized means of combining objects (or morphisms) in parallel. The category **Vect** has a monoidal structure with the tensor product functor given by the tensor product—hence, these are both represented by $\otimes$—and trivial unitors. The unit is the one-dimensional vector space, i.e. a complex number, i.e. a scalar—and it is true that the tensor product of any vector space with the one-dimensional vector space is isomorphic to itself. In quantum mechanics (and quantum information theory), we use the tensor product to combine systems and processes, which is exactly what a monoidal category allows us to do.

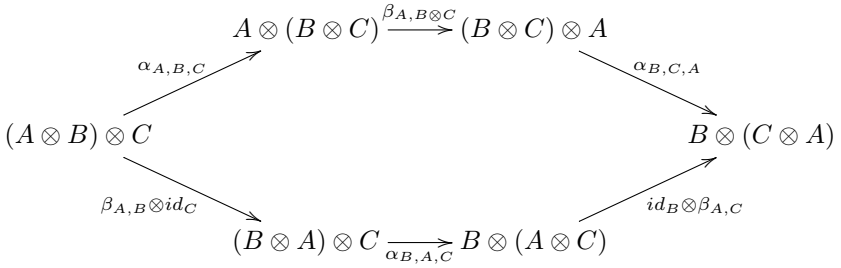
2.1.2 Braided Monoidal Categories

A *braided monoidal category* is a monoidal category equipped with natural isomorphism, β, known as a braid structure. The braiding has components $\beta_{A,B} : A \otimes B \to B \otimes A$. Thus, commuting monoidally combined objects in a braided monoidal category returns an object isomorphic to the original.

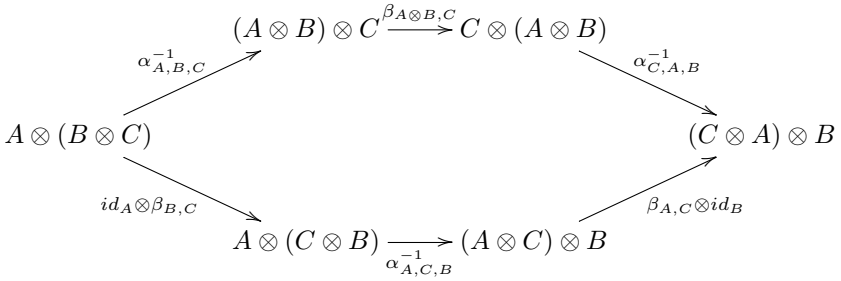
The braid structure must satisfy two equations, known as the *hexagon equations*, for coherence with the associator (and indirectly, for coherence with the unitors). The hexagon equations state

[2] Correct—it is shaped like a triangle.

that the following diagrams must commute:

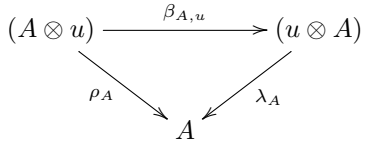


(3)



(4)

These equations show that the multiple ways of reordering and rebracketing three objects must be equal. Furthermore, these equations imply the coherence of the unitors with the braid structure—i.e. the following diagram commutes:



(5)

Thus, all the structures in a braided monoidal category are coherent.

If $\beta_{B,A} \circ \beta_{A,B} = id_{A\otimes B}$ for all objects, then the category is a *symmetric monoidal category*. In these categories, the braid operation is colloquially referred to as a *swap*—essentially since we do not have to keep track of how many times the objects "wind" around each other and only have to keep track of their order. Modular tensor categories are braided but *not* symmetric.[3]

[3] In fact, the modularity property prevents them from being symmetric—this intuitively makes sense, since we require nontriviality in at least some cases when anyons are wound around each other—q.v. Section 2.1.6.

2.1.3 Twisted Braided Monoidal Categories

A *twisted braided monoidal category* is a braided monoidal category equipped with a natural isomorphism, θ, known as a twist. The twist has components $\theta_A : A \to A$ and must satisfy a coherence equation—which is square-shaped but generally appears to be unnamed—that states that the following diagram must commute:

$$\begin{array}{ccc} A \otimes B & \xrightarrow{\theta_{A\otimes B}} & A \otimes B \\ {\scriptstyle \theta_A \otimes \theta_B}\downarrow & & \uparrow{\scriptstyle \beta_{B,A}} \\ A \otimes B & \xrightarrow[\beta_{A,B}]{} & B \otimes A \end{array} \tag{6}$$

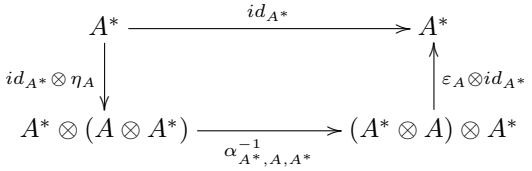
The twist can be explained by appealing to the analogy by which it was named. Objects in braided monoidal categories can be represented by strands (and drawn and calculated as such, using a graphical calculus that we will introduce in Section 4), then the monoidal combinations are strands placed next to each other, and the braiding is represented by, well, braiding the strand. A twisted braided monoidal category extends our objects so that they are ribbons. Thus, the ribbons can not only be braided, but also twisted. Then, equation 6 represents a topological coherence equation between two ribbons that are either twisted together or individually twisted and then braided.

2.1.4 Rigidity

A monoidal category can satisfy a property known as rigidity. A category that is rigid is also said to "have duals." When we interpret our objects as anyons, an object's dual represents its antiparticle.

A category is rigid if, for all objects A, there exists another object, A^*, for which there exists morphisms $\eta_A : u \to A \otimes A^*$ and $\varepsilon_A : A^* \otimes A \to u$, and the following diagrams commute:

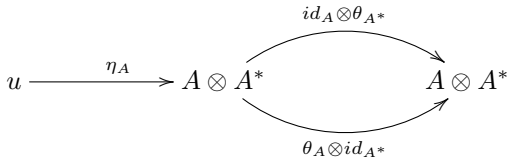
$$\begin{array}{ccc} A & \xrightarrow{\quad id_A \quad} & A \\ {\scriptstyle \eta_A \otimes id_A}\downarrow & & \uparrow{\scriptstyle id_A \otimes \varepsilon_A} \\ (A \otimes A^*) \otimes A & \xrightarrow[\alpha_{A,A^*,A}]{} & A \otimes (A^* \otimes A) \end{array} \tag{7}$$



(8)

where in equation 7 the left morphism is technically $(\eta_A \otimes id_A) \circ \lambda_A$, and in equation 8 the left morphism technically $(id_{A^*} \otimes \eta_A) \circ \rho_{A^*}$. Similarly, the right morphism in equation 7 is technically $\rho_A \circ (id_A \otimes \varepsilon_A)$, and the right morphism in equation 8 is technically $\lambda_{A^*} \circ (\varepsilon_A \otimes id_{A^*})$. In general, the use of the unitors will be implicit, and the equations will be drawn as shown (unless the equation is directly involving the unitors, as is the triangle equation).[4]

If we have a twisted braided monoidal category and a rigid structure, we need a coherence equation to ensure that the twist and the braid are compatible with rigidity. For all simple objects, A, and their duals, A^*, we require that the following diagram commutes:



(9)

A twisted braided monoidal category that is rigid, and satisfies equation 9, is sometimes referred to as a *ribbon category*. The analogy to "ribbons" was explained in the previous subsection as an intuitive explanation of the twist natural isomorphism. In fact, in must of the literature [7], the twist is often defined in terms of a ribbon structure—which is often treated as primary in the definition of ribbon categories and thus MTCs.

2.1.5 Semisimplicity

A category may satisfy a property known as *semisimplicity*. Essentially, in a semisimple category, there exists a *biproduct*, $\oplus$, that takes two objects, A and B, and returns an object, $A \oplus B$, which is the *direct sum* of the two objects. Furthermore, in a semisimple category *all* objects can be built in this manner from a finite direct sum of *simple objects*, where a simple object is an object

[4] Equations 7 and 8 do not indicate how to construct A^*, η_A, or ε_A for a given A. We will address their construction later in Section 4.6 when discussing the realization of MTCs within **2Vect** using our computer algebra system.

that cannot be reduced further.

Much of the literature [7, 31, 36] defines semisimplicity through a complicated notion leading from additive categories to abelian categories to semisimple categories. However, since we will only ever be working with modular tensor categories within **2Vect**, our definition is sufficient. Within **Vect**, the bifunctor $\oplus$ is essentially given by the direct sum on matrices, defined in the standard manner. Furthermore, in addition to semisimplicity, modular tensor categories require only a finite number of simple objects.

In our topological quantum computation model, we now have two bifunctors to account for. Combinations of objects using $\otimes$, the tensor product functor, amount to having those combinations of anyons—i.e. we can think of them as placed next to each other (abstractly, not physically) and we will move them around and braid them to enact a computation. This can be thought of as an *AND* in that we have anyon A *and* B. Combinations of objects using $\oplus$ are different possible fusion paths—i.e. $A \oplus B$ means we can have total anyon charge A *or* B. Simple objects are the different species of anyons we have in the model, and thus there will only be a finite number of them.

2.1.6 Modularity

A braided monoidal category may satisfy a property known as *modularity*. If, for all objects A, there exists at least one object B, for which the diagram does not commute:

$$\begin{array}{ccc} (A \otimes B) & \xrightarrow{\beta_{A,B}} & (B \otimes A) \\ & {\scriptstyle id_{A\otimes B}} \searrow \quad \swarrow {\scriptstyle \beta_{B,A}} & \\ & A & \end{array} \tag{10}$$

—with A, B, not made up of a direct sum combination of the unit, u—then the category is modular. Essentially, each nontrivial objects (i.e. objects that are not made up of a direct sum of the unit) must braid nontrivially with at least one other object. Clearly, this is essential for topological quantum computation, since if the "nontrivial" anyons braided trivially we would not be able to compute anything. Furthermore, if equation 10 did commute for all objects, then the

category would be symmetric monoidal. Thus, modular categories *cannot* be symmetric.[5]

A modular tensor category is a semisimple ribbon category with the modularity property (and with a finite number of simple objects). Thus, it is a twisted braided monoidal category that is rigid, modular, and semisimple.

2.1.7 Monoidal Functors

Functors also may have addition structure, usually to preserve the structure of the source and destination categories. For instance, monoidal functors preserve the structure of monoidal categories, braided monoidal functors preserve the structure of braided monoidal categories, and so on. Here, we will consider monoidal functors, as they will be considered later as an example of how to represent such functors within **2Vect** within our computer algebra system. A reference for this section may be found in Coecke and Paquette [14].

A *monoidal functor* is a functor equipped with a natural transformation, ϕ, and a morphism, ϕ_u. The natural transformation has components $\phi_{A,B} : FA \bullet FB \to F(A \otimes B)$, where here $\otimes$ represents the tensor product functor in the source category and $\bullet$ represents the tensor product functor in the destination category. A given functor is monoidal if the following diagrams commute:

$$\begin{array}{ccc} (FA \bullet FB) \bullet FC & \xrightarrow{\alpha_{2\,FA,FB,FC}} & FA \bullet (FB \bullet FC) \\ \downarrow{\scriptstyle \phi_{A,B} \bullet id_{FC}} & & \downarrow{\scriptstyle id_{FA} \bullet \phi_{B,C}} \\ F(A \otimes B) \bullet FC & & FA \bullet F(B \otimes C)) \\ \downarrow{\scriptstyle \phi_{A\otimes B,C}} & & \downarrow{\scriptstyle \phi_{A,B\otimes C}} \\ F((A \otimes B) \otimes C) & \xrightarrow[F\alpha_{1\,A,B,C}]{} & F(A \otimes (B \otimes C)) \end{array} \tag{11}$$

$$\begin{array}{ccc} FA \bullet u_2 & \xrightarrow{\rho_2} & FA \\ \downarrow{\scriptstyle id_{FA} \bullet \phi_u} & & \uparrow{\scriptstyle F\rho_1} \\ FA \bullet Fu_1 & \xrightarrow[\phi_{A,u_1}]{} & F(A \otimes u_1) \end{array} \tag{12}$$

[5] N.B. In the literature, modularity is often defined equivalently with the statement that for a category to be modular, a certain matrix, called the S-matrix, must be invertible. Please see Bakalov and Kirillov [7] for more details.

$$\begin{array}{ccc} u_2 \bullet FB & \xrightarrow{\lambda_2} & FB \\ {\scriptstyle \phi_u \bullet id_{FB}} \downarrow & & \uparrow {\scriptstyle F\lambda_1} \\ Fu_1 \bullet FB & \xrightarrow[\phi_{u_1,B}]{} & F(u_1 \otimes B) \end{array} \tag{13}$$

where the source category has monoidal structures α_1, ρ_1, and λ_1, and the destination category has monoidal structures α_2, ρ_2, and λ_2. The first diagram ensures the coherence of the monoidal functor operation and the associator natural isomorphism, and the second two diagrams ensure the coherence of the monoidal functor with the right and left unitors, respectively. Thus, the monoidal functor will preserve the monoidal structure of the underlying categories. However, a braided monoidal functor would require additional structures to preserve the braiding, and a modular tensor functor requires even more structures.

If the structures ϕ and ϕ_u are invertible, then the monoidal functor is *strong*. If they are identities, the monoidal functor is *strict*. We will mostly be interested in strong monoidal functors.

2.2 2Vect, the 2-Category

A 2-category extends the idea of a category—objects and morphisms—to include a higher level structure: morphisms between parallel morphisms. In lieu of such complicated terminology, the field has created the terms *0-cell*, *1-cell*, and *2-cell* to mean *object*, *morphism*, and *morphism-between-parallel-morphisms*, respectively.

The only 2-category we will be interested in is **2Vect**. Specifically, we are interested in **2Vect** endowed with additional structures so that it is symmetric 2-monoidal. This 2-category is the natural 2-category extension of the category **Vect**. It is within **2Vect** that we will represent modular tensor categories. Braided monoidal 2-categories were originally defined by Kapranov and Voevodsky [22, 23]. A rigorous presentation of **2Vect** without a monoidal structure is given by Elgueta [19]. A general rigorous treatment of monoidal 2-categories, including symmetric monoidal 2-categories, may be found in work of Schommer-Pries [35]. This presentation of a 2-category and its properties is mostly adapted from Mac Lane [26], with the focus on **2Vect** adapted from the other sources.

2.2.1 0-Cells

A *0-cell* or *object* of **2Vect** is the Cartesian product of a number of copies of the category **Vect** (which, of course, is itself a category). As such, there is an isomorphism between its 0-cells and the whole numbers, where the whole number indicates the number of copies.

Additionally, we can use the 2-monoidal structure to combined 0-cells using the monoidal product 2-functor, $\boxtimes$. 0-cells $\mathbf{N}$ and $\mathbf{M}$ can be combined as the 0-cell $\mathbf{N} \boxtimes \mathbf{M}$, meaning the monoidal product of N copies of **2Vect** and M copies of **2Vect** is the 0-cell representing N times M copies of **2Vect**. Via the isomorphism with the whole numbers, the 0-cell $\mathbf{N} \boxtimes \mathbf{M}$ is also isomorphic to the 0-cell $\mathbf{NM}$—i.e. the 0-cell given by the product of the numbers N and M.

2.2.2 1-Cells

A *1-cell* is a *morphism* or *arrow* of **2Vect**. Since the objects (i.e. 0-cells) of **2Vect** are copies of the category of **Vect**, a 1-cell is a *functor* between different numbers of copies of **Vect**.

Every 1-cell has a source 0-cell and destination 0-cell and is written $f : \mathbf{N} \to \mathbf{M}$. This can also be represented diagrammatically as

$$\mathbf{N} \xrightarrow{f} \mathbf{M}$$

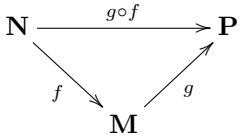
A 1-cell of **2Vect,** f maps n copies of **Vect** to m copies of **Vect**. If we "look inside" the objects of **2Vect**, the "states" will be a certain-dimensional vector space (including zero-dimensional) in each copy. As such, a 1-cell can be considered as an $m \times n$ matrix of vector spaces.[6] 1-cells may be composed via a type of *generalized matrix product* (see Section 3.1)—an operation that has the form of a matrix product, but is made up of operations other than multiplication and addition—with the direct sum as the "plus" operation and tensor product as the "times" operation. Thus, the 1-cell is able to *mix* the vector spaces from different categories together. Finally, we can "look inside" and access the "states" of a 0-cell $\mathbf{N}$ (a certain-dimensional vector space for each copy of **Vect**) with a 1-cell $\psi : \mathbf{1} \to \mathbf{N}$ (i.e. a $n \times 1$ dimensional matrix of vector spaces). Then, by composing this morphism with another compatible morphism, we can have a morphism *act* on a state.[7] The categorical viewpoint does away with the idea of "looking inside" and just considers

[6] See Table 2 for an example.

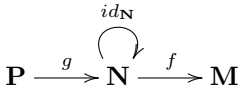
[7] This may seem like a strange hierarchy violation, since one may think that morphisms are above objects that are above states. Naively, states are elements of the object and we need to "look inside" the object to talk about

0-cells, 1-cells, and 2-cells. This is how we will access the objects and morphisms of the modular tensor categories that we will find *within* **2Vect**.[8]

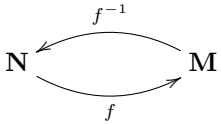
1-Cell Composition: if a 1-cell shares a destination with the source of a second 1-cell, then they may be composed. If $f : \mathbf{N} \to \mathbf{M}$ and $g : \mathbf{M} \to \mathbf{P}$, then they may be composed as $g \circ f : \mathbf{N} \to \mathbf{P}$.



Identity 1-Cell: every 0-cell, $\mathbf{N}$, has an identity 1-cell $id_{\mathbf{N}} : \mathbf{N} \to \mathbf{N}$ such that $f \circ id_{\mathbf{N}} = f$ and $id_{\mathbf{N}} \circ g = g$ for $f : \mathbf{N} \to \mathbf{M}$ and $g : \mathbf{P} \to \mathbf{N}$.



Inverse 1-Cell: the inverse of a 1-cell $f : \mathbf{N} \to \mathbf{M}$ is a 1-cell $f^{-1} : \mathbf{M} \to \mathbf{N}$, such that $f \circ f^{-1} = id_{\mathbf{M}}$ and $f^{-1} \circ f = id_{\mathbf{N}}$.



Monoidal Product: 1-cells can be combined using the monoidal product 2-functor, $\boxtimes$. If $f : \mathbf{N} \to \mathbf{M}$ and $g : \mathbf{A} \to \mathbf{B}$ are 1-cells, then monoidal product is

$$f \boxtimes g : \mathbf{N} \boxtimes \mathbf{A} \to \mathbf{M} \boxtimes \mathbf{B}$$

Swap 1-Cell: since **2Vect** is a symmetric monoidal 2-category, it is endowed with a 2-natural

states. However, the real hierarchy should place objects above both morphism and states—objects are atomic and morphisms let us go among objects. In that way, morphisms and states are at a parallel level (and a state can be thought of as a way of going between the unit object and another object) it is no surprise that there exists an isomorphism between them.

[8] This is similar to the relationship between objects, states, and morphisms within **Vect**. The objects are n-dimensional vector spaces. The morphism are linear maps between vector space. The states are vectors (i.e. one element of the ground field for every dimension of the vector space). A morphism goes between different vector spaces and can *mix* the elements from each dimension together. Additionally, we use morphisms to access the vectors (i.e. the states) of the objects. A linear map between a one-dimensional vector space and an n-dimensional vector space *is* a column vector. This is entirely analogous to **2Vect**, except the abstraction of everything is increase by one level.

isomorphism, S whose components are 1-cells, such as $S_{\mathbf{N},\mathbf{M}} : \mathbf{N} \boxtimes \mathbf{M} \to \mathbf{M} \boxtimes \mathbf{N}$, that swap order of monoidally combined 0-cells.

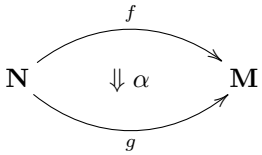
$$\mathbf{N} \boxtimes \mathbf{M} \xrightarrow{S_{\mathbf{N},\mathbf{M}}} \mathbf{M} \boxtimes \mathbf{N}$$

Since **2Vect** is symmetric, we require $S_{\mathbf{M},\mathbf{N}} \circ S_{\mathbf{N},\mathbf{M}} = id_{\mathbf{N}\boxtimes\mathbf{M}}$, and the action of S is referred to as a *swap* and not a *braid*.

2.2.3 2-Cells

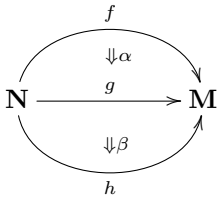
A *2-cell* is a *morphism* or *arrow* between 1-cells that have the same source and destination. It is a collection of separate linear maps between the vector spaces in the source 1-cell and the vector spaces in the destination 1-cell. Since 1-cells of **2Vect** are functors, 2-cells are *natural transformation* between the source and destination 1-cells.

If we have $f : \mathbf{N} \to \mathbf{M}$ and $g : \mathbf{N} \to \mathbf{M}$, then we can define a 2-cell $\alpha : f \Rightarrow g$. This is represented as

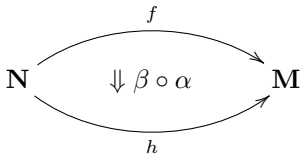


2-cells will be written with the $\Rightarrow$ arrow, while 1-cells will be written with the $\to$ arrow.

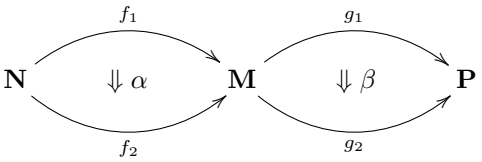
2-Cell Vertical Composition: analogous to 1-cell composition, two 2-cells may be composed vertically if the destination of the first 2-cell matches the source of the second 2-cell. If we have $f : \mathbf{N} \to \mathbf{M}$, $g : \mathbf{N} \to \mathbf{M}$, and $h : \mathbf{N} \to \mathbf{M}$, with $\alpha : f \Rightarrow g$ and $\beta : g \Rightarrow h$, then we have the following diagram:



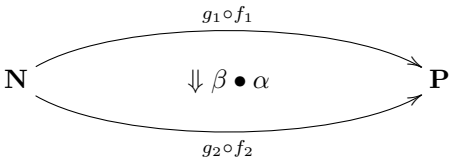
We can vertically compose α and β to get $\beta \circ \alpha : f \Rightarrow h$, or α followed by β:



2-Cell Horizontal Composition is given by the composition of a 2-cell whose 1-cells have a 0-cell destinations that matches the 0-cell source of the 1-cells of another 2-cell. In other words, if the source 1-cells of two 2-cells can be composed[9], then horizontal composition is the 2-cell that goes between the composed source 1-cells and the composed destination 1-cells. If we have $f_1 : \mathbf{N} \to \mathbf{M}$, $f_2 : \mathbf{N} \to \mathbf{M}$, $g_1 : \mathbf{M} \to \mathbf{P}$, $g_2 : \mathbf{M} \to \mathbf{P}$, $\alpha : f_1 \Rightarrow f_2$, and $\beta : g_1 \Rightarrow g_2$, then we have the following diagram:



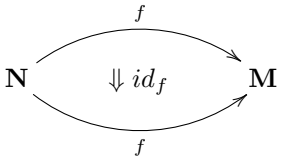
We can compose the 1-cells to get $g_1 \circ f_1 : \mathbf{N} \to \mathbf{P}$ and $g_2 \circ f_2 : \mathbf{N} \to \mathbf{P}$. We can also horizontally compose the 2-cells to get $\beta \bullet \alpha : g_1 \circ f_1 \Rightarrow g_2 \circ f_2$:

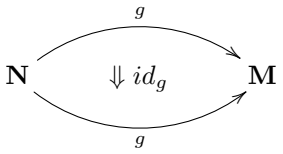


Identity 2-Cell: analogous to the identity 1-cell, every 1-cell has an identity 2-cell, such that for

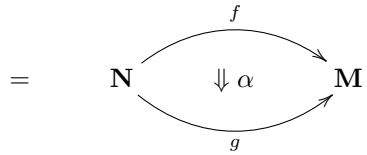
[9] which, given the definition of 2-cells, means their destination 1-cells can be composed as well.

$f : \mathbf{N} \to \mathbf{M}$ and $g : \mathbf{N} \to \mathbf{M}$, we have $id_f : f \Rightarrow f$ and $id_g : g \Rightarrow g$

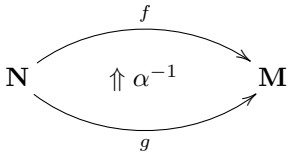




Furthermore, these must satisfy $\alpha \circ id_f = \alpha$ and $id_g \circ \alpha = \alpha$, for $\alpha : f \Rightarrow g$.[10]

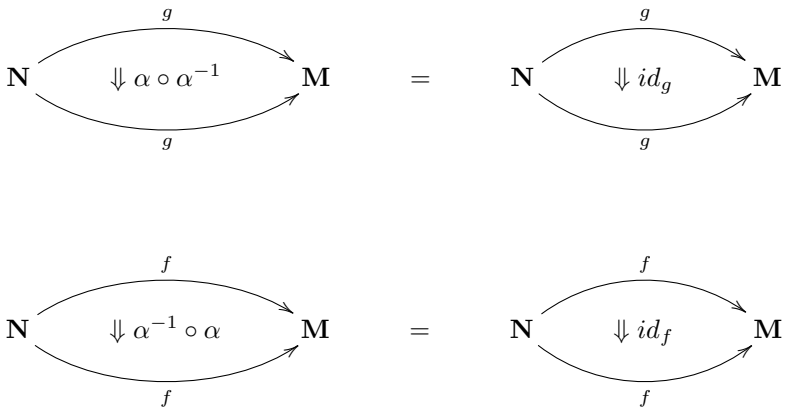


Inverse 2-Cell: analogous to the inverse 1-cell, the inverse of a 2-cell (under vertical composition), $\alpha^{-1} : g \Rightarrow f$

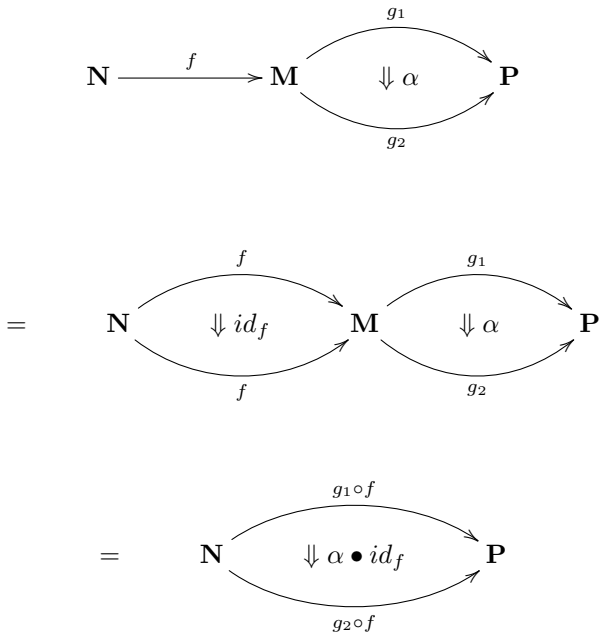


[10] Thus, the identity 2-cell is the identity under *vertical composition*.

exists if, for $\alpha : f \Rightarrow g$, we have $\alpha \circ \alpha^{-1} = id_g$ and $\alpha^{-1} \circ \alpha = id_f$

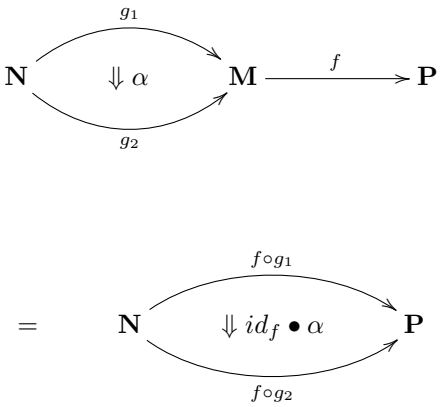


Whiskering is the name for the horizontal composition of a 2-cell with (the vertical composition) identity 2-cell of another 1-cell. In the following diagram, we can combine the 1-cell f with the 2-cell α by *whiskering*:



This makes it clear that the identity 2-cell is only the identity for vertical composition, since

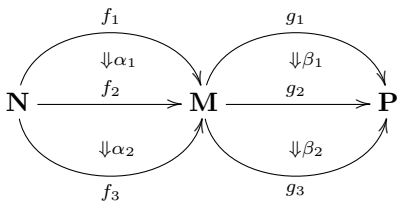
horizontal composition with the identity yields a 2-cell with different source and destination 1-cells. Finally, if the order is reversed, we can whisker the other way.



Monoidal Product: 2-cells can also be combined using the monoidal product 2-functor, $\boxtimes$. If $\alpha : f \Rightarrow g$ and $\beta : a \Rightarrow b$ are 2-cells, their monoidal product is given by

$$\alpha \boxtimes \beta : f \boxtimes a \Rightarrow g \boxtimes b$$

Interchange Law: in the following diagram, there are two ways to compose all the 2-cells together.



(14)

We can first compose vertically and then compose horizontally, to give $(\beta_2 \circ \beta_1) \bullet (\alpha_2 \circ \alpha_1) : g_1 \circ f_1 \Rightarrow g_3 \circ f_3$, or we can first compose horizontally and then compose vertically, to give $(\beta_2 \bullet \alpha_2) \circ (\beta_1 \bullet \alpha_1) : g_1 \circ f_1 \Rightarrow g_3 \circ f_3$. For all 2-categories, these 2-cells must be equal.

2.3 Topological Quantum Computation

2.3.1 *Any*ons

Quantum mechanics teaches us that there are two types of fundamental particles: *fermions* and *bosons*. We go on to learn from quantum field theory (particularly the standard model of particle physics) that fermions—such as electrons, protons, and neutrons—are particles of *matter*, and bosons—such as photons—are *force carriers*. The difference—we learn—between these particles is due to a property (i.e. quantum number) of these particles, playfully (yet misleadingly) called *spin*—an analogy intended to invoke the spinning top. Fermions have half-integer spin and bosons have integer spin. Spin is an *intrinsic angular momentum*, an angular momentum that is unlike and distinct from *rotational angular momentum*, as from classical physics. The total spin is a conserved quantity (i.e. a charge) and thus must correspond to a symmetry. Since spin is an angular momentum, we know it generates rotational symmetry. We find that under a 2π rotation, the boson wavefunction is unchanged, but the fermion wavefunction acquires a phase change of -1. However, spin is also connected to a different type of symmetry through the spin-statistics theorem. Fundamental particles of the same type are *indistinguishable*, we learn that exchanging them cannot affect the physics. Through the spin-statistics theorem, we learn that the generators of exchange symmetry are connected to spin.[11] Under exchange of two identical particles, the phase of the fermions is multiplied by -1, but phase of the wavefunction of bosons remains unchanged.[12]

We can shed more light on this by visualizing the exchange as a process. Consider the exchange of two particles in three spatial dimensions (with four dimensions total, since the process occurs in time). We move the particles around each other in a counterclockwise path (so they do not collide) that preserves each particle's orientation (so they are not also rotated), until each one occupies the position previously occupied by the other. Now, we repeat the process (still in a counterclockwise manner), and the particles occupy their original positions, having been exchanged twice. Due to the three spatial dimensions, this process can be continuously deformed into one where the particles are not exchanged at all[13]—i.e. it is topologically equivalent to no exchanges, a process

[11] The key connection between exchange symmetry (i.e. statistics) and a conserved angular momentum (i.e. spin)—the spin-statistics theorem—is predicated on the existence of antiparticles. See [32] for more details.

[12] Thus, fermions have antisymmetric wavefunctions and bosons have symmetric wavefunctions, leading to the appropriate behavior under exchange. This leads to the Pauli exclusion principle that no fermions can be in the same state, since the antisymmetry would cause the total wavefunction to be zero.

[13] To see this, first, deform the two exchanges into a process where one particle stays in place and the other is

where nothing happens—and the phase of the wavefunction must remain unchanged. Thus, for one exchange, we find the phase can change only by ± 1, leading to bosons and fermions.

The key to the above argument was three spatial dimensions. In two spatial dimensions, we have no extra dimension to move the loop over, and thus the double exchange cannot be continuously deformed into the nothing process—the identity process. The double exchange is in a different topological class than the identity process, and the wavefunction is allowed to change. However, exchange of indistinguishable particles is still a symmetry of the system (the physics must remain the same), so at most, the wavefunction can undergo a unitary operation. There are two cases to consider:

- The space of the unitary operation is one-dimensional, given by $e^{i\Phi}$. For $\Phi = 0$ we have familiar bosons, and for $\Phi = \pi$ we have familiar fermions. However, θ is not restricted to these cases and may take other values. These particles are playfully called *anyons*, because they may have "any" other *statistical angle*—as Φ is called [28]. Specifically, these particles are known as *abelian* anyons, since, for an N particle system, the order of the exchanges does not matter—i.e. the unitary operations all commute.

- The space of the unitary operation has dimension greater than one. In this case, there are a degenerate set of states describing the system, and an exchange of particles causes a rotation in this state space. In this case, different exchanges may not commute. These particles are known as *nonabelian* anyons.

Just as for fermions and bosons, we can relate—through the spin-statistics theorem—the behavior of these particles under exchange to the behavior of these particles under rotation. Under a 2π rotation, anyons will undergo a phase change $e^{i\theta}$, known as the *topological charge* or *topological spin* (i.e. plays the same role as the spin number) of the anyons. The species or type of anyon is entirely determined by its topological charge. As we will see, each species will have its own copy of the category **Vect** within the modular tensor category.

Like our fermions and bosons, anyons may appear in a bound state or be taken together as a compound system. The topological charge of anyons add in a similar fashion to the addition of

taken around it (again making sure to preserve the orientation). So far, we have only moved through two spatial dimensions. Now, use the third spatial dimensions to move the loop path of the moving particle around the straight world line of the non-moving particle. This loop can now be contracted, and we find that two particle exchanges is topologically equivalent to one in which nothing happens.

particles of different spin (i.e. the addition of angular momentum)—meaning that anyons combine to form composite anyons of possibly different topological charge. This is referred to as *fusion*, and the rules for combining anyons and the possible resulting particles are collectively known as *fusion rules*. Two abelian anyons have a unique way of adding together, i.e. they can only fuse into one species of anyon. A property of nonabelian anyons is the existence of at least one nontrivial fusion rule—meaning two nonabelian anyons can combine in different ways to form particles of different topological charge. This is not unlike a familiar property of angular momentum addition—two spin $\frac{1}{2}$ particles can combine in one way to form a spin 1 triplet or can combine in another way to form a spin 0 singlet.

It is about time to come clean; fundamental particles do not exist in $2+1$ dimensions. Furthermore, even when confined to two spatial dimensions, fundamental particles do not exhibit anyon statistics. However, in certain condensed matter many-particle systems, particle-like excitations, called *quasiparticles* do. Coined by Lev Landau, a quasiparticle is a low energy excitation of a many-body system whereby the interaction between all the particles (i.e. shielding, etc.) leads to clumps of energy that behave like particles. Specifically, in certain *fractional quantum Hall systems*—where emergent quasiparticles carry fractional electric charge—abelian and nonabelian anyons are theorized to exists [28].

2.3.2 Quantum Computation

Now what? Well—quantum computation is all about unitary operations. The basic model for quantum computation is system preparation, then unitary operations, then measurement [29]. *Quantum bits* or *qubits* are prepared in a certain state—the input, then entangled and acted on with various unitary operations designed as "quantum logic gates"—the computation, and finally measured—the result. However, it is immensely difficult—physically—to (a) apply an arbitrary unitary operation to different sets of qubits and (b) maintain the entanglement of all the qubits. Small perturbations can cause the qubits to decohere—all of this adding up to insurmountable errors in the computation. Fault-tolerant quantum computation is possible using complex error-correcting coding schemes, whereby the information is represented redundantly and is subjected to error-correcting processes. However, as the error-correction process has the potential to introduce additional errors, the overall computation can only be fault-tolerant if the error rate is considerably

small—a feat that is currently far from being achieved.

The solution is to fight the errors at the level of hardware rather than software [28]. In this conventional model, the quantum information is stored and processed locally. The qubits may become locally entangled with the environment and then thermal fluctuations can cause the state of the environment to change, leading to decoherence. Then, we require a quantum process to correct the errors. Topological quantum computers store and process the quantum information globally via what are known as *topological degrees of freedom*—degrees of freedom that are unaffected by local interactions [34].

Consider a system of nonabelian anyons. If they can be pushed around, exchanged—essentially *braided* at will—one can effect a unitary operation. Given certain species of nonabelian anyons, one could achieve an arbitrary unitary operation by braiding. As these braids are in separate topological classes, by definition, they cannot be continuously deformed into each other—i.e. they are *unaffected* by local perturbations and environmental fluctuations. They are naturally fault-tolerant, without any error correction required. Thus, this model of quantum computation is known as *topological quantum computation*.

2.3.3 Fibonacci Anyons

In order to see how quantum information may be represented with anyons, let us consider a specific model of TQC, known as the Fibonacci model. A model of TQC is a way of representing and processing the qubits with specific types of anyons. In this model, there are one or two species of anyons—depending on your reference—although everyone agrees that there is only one nontrivial species.[14] We will label the nontrivial anyon, τ, and the trivial anyon, u. As such, τ particles are their own antiparticles. In category theory, we will find that it is *rigidity* that allows objects to have duals—i.e. antiparticles.

Our presentation of the Fibonacci model will combine different approaches, taking from [28] and [32] for the physics and quantum computation intuition, and from [31] for the modular tensor categorical intuition—although not for the representation within **2Vect**. The values for the different structures of **Fib** was taken from Rowell, et al., who classified all MTCs up to four simple

[14] Some references consider the trivial anyon to be not unlike a *vacuum*, since an anyon particle and anyon antiparticle can created from the trivial particle—i.e. out of the vacuum—and can also annihilate back into it.

anyon species [33].

The first part of a model is to consider how the anyons can fuse together. The trivial anyon carries that name because it always fuses trivially—i.e. for any other anyon A, A and u fuse together to form A. Thus, we see why u can be considered the vacuum—or more aptly like a neutral particle among charges—i.e. it carries no topological charge. We write this fusion rule as $A \times u = u \times A = A$.[15] Thus, if the anyons in the Fibonacci model are to be nonabelian (which is required for TQC), the only possible nontrivial fusion rule can arise from $\tau \times \tau$ fusion. In fact, we have

$$\tau \times \tau = u + \tau \tag{15}$$

Two τ anyons can either fuse to form the trivial particle, u, or can fuse to form a single τ. Now, we will show why this model is called the Fibonacci model. It is easy to see—pardoning the slight abuse of notation—that

$$\tau^n = F_{n-1}u + F_n\tau \tag{16}$$

where F_n is the nth Fibonacci number, and τ^n means the fusion together of n anyons of charge τ.

Now, let us begin to introduce the structure of a modular tensor category. As we will see every modular tensor category realizes a different model of TQC. Furthermore, we will show how this structure lies within the 2-category **2Vect**, which we represent with numbers, matrices, and matrices of matrices.

The modular tensor category associated with the Fibonacci model is, reasonably, called **Fib**. Let us build up these structures bit by bit. Every MTC first must be a semi-simple monoidal category. Monoidal categories are equipped with a bifunctor that takes two objects of the category and combines them, returning another object in the category. A semi-simple category means all objects can be expressed as a direct sum of simple objects. Together, this is exactly what the fusion rules are!—a way of combining objects (fusing anyons) into direct sums of other objects of the category (i.e. they fuse in potentially different ways into other anyons of the model). Considering a 2-category as category of categories, it has categories as objects, functors as morphisms, and natural transformations as morphisms between parallel morphisms. Thus, **Fib** will be an object in

[15] Since particle exchange is a symmetry of the system, it cannot change the total topological charge (this is *the* fact that was used in defining topological charge and anyons). Thus, the factors in all the fusion rules must be commutative. We will also see later that this must be true categorically because MTCs are braided categories.

2Vect—specifically, the object **2** or two copies of **Vect**. The *product functor* of the MTC, which encapsulates the fusion rules of the TQC model, will be a morphism of **2Vect**. This is represented as a matrix—specifically for **Fib**, we have $m : \mathbf{4} \to \mathbf{2}$,

$$m = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \tag{17}$$

The four fusion rules (trivial and nontrivial) are encapsulated in the columns of m—the last column is the nontrivial rule. The unit, u, of the MTC picks out the trivial particle. In this case, we have chosen to represent it as $u : \mathbf{1} \to \mathbf{2}$,

$$u = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{18}$$

Here we see another feature of category theory—an object of the MTC (which represents a collection of the different types of particles in the anyon model) is represented by a morphism (i.e. a matrix) in the 2-category, specifically a 2×1 matrix. Similarly, we can access τ by $\tau : \mathbf{1} \to \mathbf{2}$,

$$\tau = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{19}$$

By the semisimple structure of **2Vect**, we can represent arbitrary "states," like $au + b\tau$, in the expected way—thus, for **Fib**, such a state is represented by two natural numbers, with equations 18 and 19 defining the basis.

Let us return to m. We can represent two anyons together—pre-fusion[16]—as a tensor product of two two-dimensional vector spaces. Thus, we represent $\tau \times \tau$ as $\tau \boxtimes \tau : \mathbf{1} \to \mathbf{4}$,

$$\tau \times \tau = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{20}$$

To determine how these particles fuse, we compose (matrix product) that state with m and see

[16] By pre-fusion, we mean the left side of equation 15.

we get the state that represents $u + \tau$—the correct fusion rule. Thus, fusion is represented by the action of the product functor on two objects when working "inside" the MTC, but this is also represented by the composition of corresponding of morphisms in **2Vect**. The majority of people will adopt the first viewpoint when they consider TQC categorically, but we will concentrate on the second viewpoint as it will allow us to represent *all* MTCs within our computer algebra system.

Now, let us consider the fusion of three anyons. If any of the anyons are trivial, then there is only one or zero ways for them to fuse together to form a given anyon. For instance, there is only one way for $(u \times \tau) \times u$ to fuse into a τ, and zero ways for it to fuse into a u—i.e. $(u \times \tau) \times u = 0u + 1\tau = \tau$. However, the fusion of three τ anyons is nontrivial: $(\tau \times \tau) \times \tau = u + 2\tau$. There is one way for them to fuse to u, but there are two ways for them to fuse to τ. The first fusion of $\tau \times \tau$ can give an u *or* a τ. Then, either of these can fuse with the third τ to give an overall charge of τ—we have $(\tau \times \tau) \times \tau \to u \times \tau \to \tau$ or $(\tau \times \tau) \times \tau \to \tau \times \tau \to \tau$. Thus, if the overall charge of the three anyons is τ, there are two distinct *fusion channels.*[17] As we will see later, the fusion rules for three anyons is given by the matrix product of m with m in a tensor product with the 2×2 identity matrix $m \circ (m \boxtimes id_{\mathbf{2}}) : \mathbf{8} \to \mathbf{2}$,

$$m(m \otimes I_{2\times 2}) = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 2 \end{bmatrix} \tag{21}$$

where in the morphism definition the tensor product is written with the symbols $\boxtimes$ and $id_{\mathbf{2}}$ to make contact with the categorical notation used later.[18]

The space of three τ anyon fusion into τ is a degenerate two-dimensional vector space. We can pick a basis using the intermediate fusion states: one basis state is $(\tau \times \tau) \times \tau \to u \times \tau \to \tau$ and the other basis state is $(\tau \times \tau) \times \tau \to \tau \times \tau \to \tau$. Bonesteel, *et al.*, introduced the following notation for basis states to make contact with familiar bra-ket notation: if we have $(\tau \times \tau) \to A$, meaning two τ fuse to an anyon A, then we represent this as $|\ (\bullet, \bullet)_A \rangle$ [9]. We use $\bullet$ to represent an anyon, and $(\bullet, \bullet)_A$ to represent the fusing of two anyons to A, and what they are should be clear from

[17] The use of $\to$ instead of $=$ indicates we are considering just one of possibly many fusion channels.
[18] The $\boxtimes$ symbol is used as the product functor in the 2-category **2Vect**.

context.[19] Thus, we have as our basis

$$(\tau \times \tau) \times \tau \to u \times \tau \to \tau \equiv| \ ((\bullet, \bullet)_u, \bullet)_\tau \rangle \equiv| \ 0 \rangle \tag{22}$$

$$(\tau \times \tau) \times \tau \to \tau \times \tau \to \tau \equiv| \ ((\bullet, \bullet)_\tau, \bullet)_\tau \rangle \equiv| \ 1 \rangle \tag{23}$$

These basis vectors are labeled very suggestively because, in fact, they will form our qubit. Essentially, in the Fibonacci model, three τ anyons of total charge τ form a qubit. Sometimes, three τ anyons will fuse to u, which we write as

$$(\tau \times \tau) \times \tau \to \tau \times \tau \to u \equiv| \ ((\bullet, \bullet)_\tau, \bullet)_u \rangle \equiv| \ NC \rangle \tag{24}$$

meaning "non-computational." This state is reached only in error, a process we will describe in Section 2.3.6.

However, there is another equivalent basis to consider. Instead of the first two anyons fusing and those intermediate states forming the basis, the second two anyons can fuse first and form the basis, giving:

$$\tau \times (\tau \times \tau) \to \tau \times u \to \tau \equiv| \ (\bullet, (\bullet, \bullet)_u)_\tau \rangle \equiv| \ + \rangle \tag{25}$$

$$\tau \times (\tau \times \tau) \to \tau \times \tau \to \tau \equiv| \ (\bullet, (\bullet, \bullet)_\tau)_\tau \rangle \equiv| \ - \rangle \tag{26}$$

We can define a change of basis matrix, $\alpha_\tau^{\tau\tau\tau}$, such that for a state $| \ \psi \rangle$ in the 0, 1 basis, $\alpha_\tau^{\tau\tau\tau} \ | \ \psi \rangle$ is in the $\pm$ basis. We label α with three τ on top to indicate the fusion of three τ anyons, and the subscript indicates they anyon they fuse into—the total charge of the three—which, in this case, is also τ.

Unsurprisingly, this change of basis matrix is part of the monoidal structure of **Fib**. Every monoidal category is equipped with a *natural isomorphism*, α, whose components are morphisms such that

$$(A \otimes B) \otimes C \xrightarrow{\alpha_{A,B,C}} A \otimes (B \otimes C) \tag{27}$$

where we are using $\otimes$ to represent the product functor for the MTC. This is exactly what the

[19] Although, we suppose, this could also be written as $| \ (\tau, \tau)_A \rangle$ for clarity—or in a different model with more than one nontrivial anyon, we could write a basis given by the fusion $A \times B \to C$ as $| \ (A, B)_C \rangle$.

matrix $\alpha_\tau^{\tau\tau\tau}$ *is*—the τ, τ, τ component of the associator, which is a morphism. Thus, the entire associator is a collection of matrices—which we can represent as a matrix of matrices for all the different simple object combinations

$$\alpha = \begin{bmatrix} [1] & \emptyset_{0\times 0} & \emptyset_{0\times 0} & [1] & \emptyset_{0\times 0} & [1] & [1] & [1] \\ \emptyset_{0\times 0} & [1] & [1] & [1] & [1] & [1] & [1] & \begin{bmatrix} \varphi^{-1} & \varphi^{-\frac{1}{2}} \\ \varphi^{-\frac{1}{2}} & -\varphi^{-1} \end{bmatrix} \end{bmatrix} \tag{28}$$

where $\emptyset_{0\times 0}$ is the 0×0 dimensional matrix, and $\varphi = \frac{1+\sqrt{5}}{2}$, the *golden ratio*. We invoke zero-dimensional matrices because, for instance, there is no way for three u anyons to fuse to a τ anyon. Thus, this fusion is represented by a zero-dimensional vector space, and the change of basis matrix is a 0×0 dimensional matrix. Our change of basis matrix, $\alpha_\tau^{\tau\tau\tau}$, is the matrix in the lower right hand corner, which we might also choose to index as $\alpha_{2,8}$. Since the morphisms of a 2-category are functors, natural transformations are morphism between parallel morphisms in the 2-category, and thus α has a source and destination, which may be written as $\alpha : m \circ (m \boxtimes id_{\mathbf{2}}) \Rightarrow m \circ (id_{\mathbf{2}} \boxtimes m)$, where the double arrow $\Rightarrow$ lets us known that this is defining a natural transformation. In fact, we can represent any natural transformation as a matrix of matrices in this manner.

For the most part, we have described the important aspects of the monoidal structure of **Fib**, as well as the fusion of the anyons. How about quantum computation—i.e. braiding? For this, we require a braided monoidal structure, which provides another natural isomorphism, $\beta : m \Rightarrow m \circ S$, where S is the *swap* 1-cell (which is part of the symmetric structure of **2Vect** and allows us to exchange objects monoidally combined by the product 2-functor $\boxtimes$). The components of β given the phase change associated with braiding or exchanging two anyons in a clockwise manner—or swapping the order of two objects monoidally combined in the MTC. Using the 0, 1 basis, we can represent the braiding unitary operation of braiding two τ's by

$$\beta^{\tau\tau} = \begin{bmatrix} e^{-\frac{4}{5}i\pi} & 0 \\ 0 & e^{\frac{3}{5}i\pi} \end{bmatrix} \tag{29}$$

We further see how these anyons are nonabelian and how that connects with a nontrivial fusion

rule. First, $\tau \times \tau$ can fuse into u or τ, and, second, they braid different depending on their charge. We can represented the whole natural isomorphism, $\beta : m \Rightarrow m \circ S$, as

$$\beta = \begin{bmatrix} [1] & \emptyset_{0\times 0} & \emptyset_{0\times 0} & \left[e^{-\frac{4}{5}i\pi}\right] \\ \\ \emptyset_{0\times 0} & [1] & [1] & \left[e^{\frac{3}{5}i\pi}\right] \end{bmatrix} \tag{30}$$

and we see that $\beta_u^{\tau\tau} = e^{-\frac{4}{5}i\pi}$ and $\beta_\tau^{\tau\tau} = e^{\frac{3}{5}i\pi}$, using the notation from before. The other elements describe what we expect—for instance, braiding $\tau \times u$ causes no phase change, since u is the trivial particle and braiding with it is like doing nothing.

Computation is performed by successive braiding of the anyons that form our qubits. In this case, that means we can build up unitary operations by application of the braid matrix given by $\beta^{\tau\tau}$ and then associator basis change matrix, given by $\alpha_\tau^{\tau\tau\tau}$, to regroup the anyons we want to braid. This will be explored further in Section 2.3.5.

Finally, we will consider braided monoidal categories with a twist. The twist, θ, is a natural isomorphism whose components describe—effectively—a twist in the anyon's world line or a 2π rotation. It is this quantity, in Section 2.3.1, that we called *topological charge* or *topological spin*. We find that when u undergoes a 2π rotation, it undergoes no phase change, since it is the trivial particle. However, when τ undergoes a 2π rotation, it undergoes a phase change of $e^{\frac{4}{5}i\pi}$—which is both nontrivial and different from bosons and fermions, since they can only change phase by ± 1. As a natural isomorphism, we represent it as $\theta : id_{\mathbf{2}} \Rightarrow id_{\mathbf{2}}$,

$$\theta = \begin{bmatrix} [1] & \emptyset_{0\times 0} \\ \\ \emptyset_{0\times 0} & \left[e^{\frac{4}{5}i\pi}\right] \end{bmatrix} \tag{31}$$

where again the zero-dimensional matrices indicate that if you put a twist in a particle's world line—rotate it by 2π—it cannot change into a different species of anyon—i.e. its topological charge is conserved. In Section 2.1.3, we will show how the twist and braid structures are intricately related through the twist axiom (which is also the all important connection between spin and statistics mention in Section 2.3.1).

categorical name	categorical symbol	TQC name	TQC symbol
product functor	m	fusion rules	N
associator	α	F-matrix (basis change)	F
braiding	β	R-matrix	R
twist	θ	topological spin	Θ

Table 1: Different notation and names used in category theory literature and topological quantum computing literature.

2.3.4 Notation

We would like to clear up some notational differences between the category theory literature—for instance, [5, 31, 36]—and the topological quantum computing literature—for instance, [28, 32]. Unfortunately, even within the category theory literature, there is no agreement for the symbols representing the various structures.

In this work, we have chosen to use the categorical symbols, choosing those that are (roughly) the most common and least likely to cause confusion or collide with other symbols. On the other hand, symbols in the TQC work are mostly standardized. Table 1 provides a guide for converting between the different notations. The category theory symbols and structures should be familiar by this point in the work, as should their roles within topological quantum computing. The names and symbols within TQC literature for these structures are provided for readers interested in mediating between different works.

2.3.5 Computation in the Fibonacci Model

Picking up from where we left off in Section 2.3.3, we would like to how to enact a computation in the Fibonacci model. We know that the matrix representing the braiding of two τ anyons is given by $\beta^{\tau\tau}$, defined by equation 29. Specifically, two τ anyons with total charge u will undergo phase change $\beta_u^{\tau\tau} = e^{-\frac{4}{5}i\pi}$, and two τ anyons with total charge τ will undergo phase change $\beta_\tau^{\tau\tau} = e^{\frac{3}{5}i\pi}$.

Given three τ Fibonacci anyons paired as $(\tau \times \tau) \times \tau$, we know that we have a basis, $|\ ((\bullet,\bullet)_u,\bullet)_\tau\rangle = |\ 0\rangle$, $|\ ((\bullet,\bullet)_\tau,\bullet)_\tau\rangle = |\ 1\rangle$, and $|\ ((\bullet,\bullet)_\tau,\bullet)_u\rangle = |\ NC\rangle$ (defined by equations 22–24), determined by the intermediate fusion states and total charge of the three anyons. The

three-dimensional vector space reduces into a two-dimensional total charge τ part—i.e. our qubit—spanned by the vectors $|\,0\rangle$ and $|\,1\rangle$, and a one-dimensional total charge u, spanned by $|\,NC\rangle$.

If we exchange the first two τ anyons (in a clockwise manner), a state $|\,0\rangle$ would undergo phase change $\beta_u^{\tau\tau}$, a state $|\,1\rangle$ would undergo phase change $\beta_\tau^{\tau\tau}$, and a state $|\,NC\rangle$ would undergo phase change $\beta_\tau^{\tau\tau}$—since the phase change is determined by the charge of the intermediate fusion state of the first two anyons. This can be described by the unitary matrix

$$U_1 = \begin{bmatrix} \begin{bmatrix} \beta_u^{\tau\tau} & 0 \\ 0 & \beta_\tau^{\tau\tau} \end{bmatrix} & 0_{2\times 1} \\ 0_{1\times 2} & [\beta_\tau^{\tau\tau}] \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} e^{-\frac{4}{5}i\pi} & 0 \\ 0 & e^{\frac{3}{5}i\pi} \end{bmatrix} & 0_{2\times 1} \\ 0_{1\times 2} & \left[e^{\frac{3}{5}i\pi}\right] \end{bmatrix} \tag{32}$$

where we have a direct sum between the action on the qubit and the non-computational sector. The upper left 2×2 matrix is a unitary operation on our qubit. However, we do not have enough generators to fully describe all the possible braids of our anyons. More importantly, this operation alone does not support universal quantum computation. Instead, we need at least another operation.

Luckily, we can also exchange the second and third anyons. This provides another distinct unitary operation, even though we are still just exchanging two τ, due to an implicit change of basis involved in such a exchange. Remember, our anyons are paired as $(\tau \times \tau) \times \tau$, but to exchange the last two anyons, we need to pair them as $\tau \times (\tau \times \tau)$. To determine the action of exchanging the second and third anyons, we need to change from the 0, 1 basis to the $\pm$ basis[20], we need to rebracket—we need to apply the associator, α. Our change of basis matrix, built from the associator (defined in equation 28) is

$$V = \begin{bmatrix} \alpha_\tau^{\tau\tau\tau} & 0_{2\times 1} \\ 0_{1\times 2} & \alpha_u^{\tau\tau\tau} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \varphi^{-1} & \varphi^{-\frac{1}{2}} \\ \varphi^{-\frac{1}{2}} & -\varphi^{-1} \end{bmatrix} & 0_{2\times 1} \\ 0_{1\times 2} & [1] \end{bmatrix} \tag{33}$$

As expected, the non-computational sector remains the same in both bases.

Now, in the $\pm$ basis, if we exchange the second and third τ anyons, a state $|\,+\rangle$ would undergo

[20] Given by $|\,(\bullet,(\bullet,\bullet)_u)_\tau\rangle = |\,+\rangle$ and $|\,(\bullet,(\bullet,\bullet)_\tau)_\tau\rangle = |\,-\rangle$, defined in equations 25 and 26.

phase change $\beta_u^{\tau\tau}$, a state $|-\rangle$ would undergo phase change $\beta_\tau^{\tau\tau}$, and a state $|NC\rangle$ would undergo phase change $\beta_\tau^{\tau\tau}$—since now the phase change is determined by the charge of the intermediate fusion state of the final two anyons. This is described by the same matrix, U_1, except it is now taken to be in a different basis.

To determine the action of exchanging our second and third anyon in the 0, 1 basis, we first change to the $\pm$ basis, apply the exchange matrix, then change back to the original basis.

$$U_2 = V^{-1}U_1V = \begin{bmatrix} \begin{bmatrix} e^{\frac{4}{5}i\pi}\varphi^{-1} & e^{-\frac{3}{5}i\pi}\varphi^{-\frac{1}{2}} \\ e^{-\frac{3}{5}i\pi}\varphi^{-\frac{1}{2}} & -\varphi^{-1} \end{bmatrix} & 0_{2\times 1} \\ 0_{1\times 2} & \begin{bmatrix} e^{\frac{3}{5}i\pi} \end{bmatrix} \end{bmatrix} \tag{34}$$

All possible braids on three anyons can be created by exchanging either the first two or the last two (in a clockwise or a counterclockwise manner, with the action of a counterclockwise braid represented by the inverses of U_1 and U_2) [28]. In fact, U_1 and U_2 will generate a representation of the braid group and can approximate an arbitrary two-dimensional unitary operation by different combinations of braids [32]. To entangle more qubits together, we can consider more anyons—i.e. six anyons of charge τ will represent two qubits.

Using the method demonstrated in this section (exchanging and rebracketing), we can determine the action of braiding different anyons together. The inverse process—which generally requires brute force or clever computation [28]—will allow us find the (approximate) braid for a given unitary operation. Thus, we can piece together which braids represent the unitary operations that are standard for universal quantum computation. Finally, it can be shown that all the required gates can be constructed for the Fibonacci model, and thus it supports universal quantum computation [31].[21]

2.3.6 Errors

This discussion of errors is mostly adapted from Nayak, *et al.* [28].

Much of this work has been focused on topological quantum computation as a naturally fault-tolerant error free model of quantum computation. By encoding the quantum information topo-

[21] In fact, it can be done with only the two clockwise exchange operations [28].

logically, it is robust against local perturbations and environment fluctuations.

Thus, errors in topological quantum computing are nonexistent in the traditional sense. Are there any errors in this model? Yes—although they are most unlike the debilitating errors that currently plague non-topological approaches to computation. Essentially, errors can only result from other anyons entering the system and unintentionally braiding with the computational anyons representing the qubits. Such anyons can be unintentionally created from the vacuum (and are more likely, the greater the temperature) and can wander around "wreaking havoc."

Luckily, a large majority of these processes do not cause any errors. If an anyon anti-anyon pair is created from the vacuum and one of them braids with another anyon and then they annihilate, the qubit's total state is unchanged (within an overall phase), and no error is caused. If an anyon anti-anyon pair is created from the vacuum and one of them annihilates with an anyon involved in our computation, but its partner takes the annihilated anyon's place, no error is caused. In fact, an error can only occur if a thermal anyon braids with more than one computational anyon. This process can be suppressed by keeping the computational anyons physically separated—making it likely the stray anyons will annihilate long before they braid with more than one computational anyon.

Finally, in most cases, braids can only approximate a desired unitary operation. This means that such operations will not be exact and can lead to error in the computation. It is through this approximate braiding of multiple qubits together that a non-computational state, $|\ NC\rangle$, can be reached—a process known as *leakage error* [9]. However, this error can be made arbitrarily small by increasing the length of the braid, and their length only grows logarithmically as the accuracy is increased [32]. Thus, this source of error is entirely controllable and can easily be suppressed to the desired degree.

3 Realization

The computer algebra system we developed was implemented in Mathematica [37], but can be adapted for any computer algebra package. Here, we will describe generally how to realize such a system for **2Vect** and MTCs.

Before we can represent MTCs, it is necessary to fully develop the mathematical helper functions

to represent the symmetric monoidal 2-category **2Vect**.

3.1 Generalized Operations

To facilitate the implementation of **2Vect**, we require four generalized matrix operations:

Generalized Unary Element Operation is a unary operation that takes a matrix, A, and a unary function, f, and applies it element-wise to A. This is defined as

$$B_{ij} = f(A_{ij}) \tag{35}$$

where B is the output matrix. The function, f, is referred to as a *generalized operation*.

Generalized Binary Element Operation is a binary operation that takes two matrices, A and B, and a binary function, f, and applies it element-wise to A and B. This is defined as

$$C_{ij} = f(A_{ij}, B_{ij}) \tag{36}$$

where C is the output matrix. The function, f, is referred to as a *generalized operation*.

Generalized Matrix Product is a binary operation that takes two matrices, A and B, and two binary functions, f and g, and combines them in a way analogous to the *matrix product* ($\sum_j A_{ij}B_{jk}$) to form a new matrix. Specifically, the *generalized matrix product* is given by

$$C_{ik} = \underset{j}{G} f(A_{ij}, B_{jk}) \tag{37}$$

where C is the output matrix, and G_j is taken to mean repeated application of g for j iterations[22]. Because of the roles these operations play, f is referred to as a *generalized times* and g is referred to as a *generalized plus*.

Generalized Tensor Product is a binary operation that takes two matrices, A and B, and a binary function, f, and combines them analogously to the *Kronecker product* or *tensor*

[22] in the same way that $\sum_i$ indicates i repeated iterations of addition.

product, to form a new matrix. The *tensor product* of two matrices, A and B, is given by

$$(A \otimes B)_{\alpha\beta} = A_{ij}B_{kl} \tag{38}$$

$$\alpha = a(i-1) + k \tag{39}$$

$$\beta = b(j-1) + l$$

where, if A has dimensions $m \times n$ and B has dimensions $a \times b$, then $A \otimes B$ has dimensions $ma \times nb$. Analogously, the *generalized tensor product* is given by

$$C_{\alpha\beta} = f(A_{ij}, B_{kl}) \tag{40}$$

where C is the output matrix and if A has dimensions $m \times n$ and B has dimensions $a \times b$, C has dimensions $ma \times nb$ and α and β are defined the same as above. In this case, f is also referred to as a *generalized times*.

3.2 Linear Algebra Extensions

In order to work with the symmetric monoidal 2-category **2Vect**, we need to make two modifications to the standard set of linear algebra functions:

- Functions need to be extended to handle *zero-dimensional matrices*.

  Since 0 is a perfectly good object in **2Vect**, we can have morphisms from 0 to other objects, from other objects to 0, or from 0 to 0. These morphisms or 1-cells will be represented by matrices of zero dimensions, and it is necessary to keep track of the size of both the rows and columns of these matrices.

- An entire set of functions need to be defined over matrices of matrices, or *2-matrices*.

  Objects in **2Vect** are represented by whole numbers, 1-cells are represented by matrices, and 2-cells are represented by matrices of matrices. The real task in representing **2Vect**

is implementing the 2-cell operations, which ultimately means understanding operations on 2-matrices.

3.2.1 Zero-Dimensional Matrices

A zero-dimensional matrix is an $m \times 0$ dimensional matrix, a $0 \times n$ dimensional matrix, or a 0×0 dimensional matrix (with $m \neq 0$ and $n \neq 0$). Since zero-dimensional matrices have no elements, there exists only one such matrix for each pair of dimensions.[23] In the following, we will extend many familiar *mathematical* operations so they may be defined for zero-dimensional matrices.

Matrix Product: consider two matrices, A and B, of dimensions $a \times b$ and $c \times d$, to be multiplied as AB. The matrix product is still only defined when the inner dimensions of the input matrices match (i.e. $b = c$), with the resulting matrix AB being $a \times d$ dimensional. This leaves two cases:

- *Inner dimensions are zero* ($b = c = 0$): AB will be an $a \times d$ matrix of zeroes.[24]
- *One or both of the outer dimensions are zero* ($a = 0$ or $d = 0$ or both): AB will be an $a \times d$ zero-dimensional matrix (since either a or d or both are zero).

Tensor Product (or Kronecker Product): if either input is a zero-dimensional matrix, the tensor product is also a zero-dimensional matrix whose dimensions are still given by equation 39.

Direct Sum: as a semisimple monoidal category, **2Vect** requires a second monoidal product given by the direct sum[31]. The direct sum is given by

$$A \oplus B = \begin{bmatrix} A & 0_{m \times b} \\ 0_{a \times n} & B \end{bmatrix} \tag{41}$$

where $0_{m \times b}$ is taken to mean a $m \times b$ block of zeros, and if A has dimensions $m \times n$ and B has dimensions $a \times b$, then $A \oplus B$ has dimensions $(m + a) \times (n + b)$.

[23] A *zero-dimensional matrix* is distinct from a *zero matrix*. A zero matrix has a non-zero number of rows and a non-zero number of columns, and all the entries are zeroes. A zero-dimensional matrix is a matrix that has zero rows or zero columns, or both. As such, there are no entries.

[24] Since A and B really carry no information other than their dimension, there is nothing else that this could possibly be.

Essentially, equation 41 still holds for the zero-dimensional implementation, as long as zero is inserted for the appropriate dimensions. Effectively, the direct sum of a matrix A and a matrix B, with zero as a row (column) dimension, will add columns (rows) of zeros to A equal to the non-zero dimension of B.[25] For example, if A has dimensions $m \times n$ (both non-zero) and B has dimensions $0 \times b$ matrix, then their direct sum will be

$$A \oplus B = \begin{bmatrix} A & 0_{m \times b} \end{bmatrix}$$

which is exactly what one would expect by plugging into equation 41. This means that the 0×0 matrix is the identity for the direct sum. Additionally, if A and B have complimentary zero dimensions, such as $m \times 0$ and $0 \times n$ respectively, the result of their direct sum will be an $m \times n$ matrix of zeroes.[26]

Identity Matrix: as previously stated, there exists only one zero-dimensional matrix for each pair of dimensions $m \times n$, with $m = 0$ or $n = 0$ or both. Since identity matrices are square, the only zero-dimensional matrix with an identity is the 0×0 matrix. The matrix product of this matrix with itself must be itself, and thus it is its own identity.

Inverse Matrix: in category theory, for a matrix A to have an inverse A^{-1}, it must be a left inverse and a right inverse.

$$A^{-1}A = AA^{-1} = I \tag{42}$$

Thus, the inverse is only defined if the matrix is square, so only the 0×0 matrix has an inverse. As there is only one 0×0 matrix, the only candidate for the inverse it itself. Since the matrix product of this matrix with itself is itself (which is also the identity matrix for the 0×0 matrix), it has the properties required by equation 42 and is its own inverse.

Adjoint Matrix of a zero-dimensional matrix is given by swapping the dimensions, since zero-dimensional matrices have no elements. Thus, the adjoint of the $0 \times n$ (or $m \times 0$) matrix is the $n \times 0$ (or $0 \times m$) matrix.

[25] in this case, the number of the columns (rows) of B

[26] As with the matrix product A and B really carry no information other than their dimension, so there is nothing else that this could possibly be.

3.2.2 Two-Matrices

A *2-matrix* has the form of a matrix, but its elements themselves are also matrices. These elements will be referred to as *element-matrices* or *inner-matrices*. Consider a 2-matrix A with outer dimensions $m \times n$, and element-matrices A_{ij} with dimensions $a_{ij} \times b_{ij}$. An element-matrix element is indexed by $(A_{ij})_{\alpha\beta}$, where the first two indices, i and j range over the outer dimensions m and n, the the second two indices, α and β, range over the inner dimensions a_{ij} and b_{ij}. 2-matrices were introduced by Elgueta in [19] in a slightly different context—as the 2-category **2Mat**, the 2-category analog of the category **Mat**. Here, we focus 2-matrices from a non-categorical perspective—simply as matrices of matrices. Please see Section 3.3.3 for a specific example of a 2-matrix.

Many operations that exist for regular matrices or *1-matrices* have 2-matrix analogues. Most involve the use of generalized operations, as defined in Section 3.1. Furthermore, these operations will correspond to the required set of operations on 2-cells **2Vect**.

Element-Matrix Product takes a pair of 2-matrices of the same outer dimensions and returns a 2-matrix given by the element by element matrix product. For the *element-matrix product* to exist, the outer matrix dimensions of both 2-matrices must be the same (so they have the same number of element-matrices in the same positions) and the matrix product must exist between all the corresponding pairs of element-matrices. If A and B are 2-matrices of dimensions $m \times n$ and $p \times q$ and the element-matrix product $A \circ B$ is to be computed, then we must have $m = p$ and $n = q$. Furthermore, for each corresponding element-matrix A_{ij} and B_{ij} of dimensions $a_{ij} \times b_{ij}$ and $c_{ij} \times d_{ij}$, we must have $b_{ij} = c_{ij}$ for all $i = \{1, ..., m\}$ and $j = \{1, ..., n\}$. If the element-matrix product $A \circ B$ exists, then it is given by

$$((A \circ B)_{ij})_{\alpha\beta} = \sum_{\gamma=1}^{b_{ij}} (A_{ij})_{\alpha\gamma} (B_{ij})_{\gamma\beta} \tag{43}$$

where i and j range over m and n, respectively, and α and β range over a_{ij} and d_{ij}, respectively. This is an example of a generalized binary element operation with the matrix product as the generalized operation.

Outer-Matrix Product is a generalized matrix product with the tensor product as the generalized times and the direct sum as the generalized plus. The only constraint is that if A and

B are 2-matrices of dimensions $m \times n$ and $p \times q$ then for the *outer-matrix product* to exist, we must have $n = p$. Furthermore, if the inner dimensions are $a_{ij} \times b_{ij}$ and $c_{jk} \times d_{jk}$, with $i = \{1, ..., m\}$, $j = \{1, ..., n\}$, and $k = \{1, ..., q\}$, then by equation 37, the outer-matrix product $A \bullet B$ is given by

$$(A \bullet B)_{ik} = \bigoplus_{j=1}^{n} A_{ij} \otimes B_{jk} \tag{44}$$

Identity Two-Matrix is the identity for the element-matrix product. Thus, if A is a 2-matrix with outer dimensions $m \times n$, then the *identity 2-matrix* I is also an $m \times n$ 2-matrix whose element-matrices I_{ij} are individually identity matrices to their corresponding element-matrices of A such that

$$A = A \circ I = I \circ A \tag{45}$$

The identity two matrix is only defined for 2-matrices whose element-matrices are all square. It can be implemented by a generalized unary element operation with a *get identity matrix from matrix* function as the generalized operation.

Inverse Two-Matrix is the corresponding inverse for the element-matrix product. Thus, if A is a 2-matrix with outer dimensions $m \times n$, then its *inverse 2-matrix* A^{-1} is also an $m \times n$ 2-matrix whose element-matrices A_{ij}^{-1} are individually inverse matrices to the corresponding element-matrices of A and subject to the expected existence conditions for matrix inverses such that

$$I = A \circ A^{-1} = A^{-1} \circ A \tag{46}$$

where I is the previously defined identity 2-matrix corresponding to A. For A^{-1} to exist, the matrix inverse must exist for each element-matrix of A.[27] A function to give the inverse 2-matrix can be implemented by a generalized unary element operation with a *get inverse matrix from matrix* function as the generalized operation.

Two-Tensor Product takes a pair of 2-matrices and returns a 2-matrix that is the tensor product on both the outer and inner matrices. It is a generalized tensor product with the tensor product as the generalized times. If A and B are 2-matrices of dimensions $m \times n$ and $a \times b$

[27] and thus the element-matrices of A must be square if A has an inverse.

and the two-tensor product $A \boxtimes B$ is to be computed, then

$$(A \boxtimes B)_{\alpha\beta} = A_{ij} \otimes B_{kl} \tag{47}$$

$$\alpha \equiv a(i-1) + k$$

$$\beta \equiv b(j-1) + l$$

where $A \boxtimes B$ has outer dimensions $ma \times nb$.[28]

Additionally, we could define a *2-direct sum* and potentially two types of *2-adjoints*, but none of these operations are required for implementing **2Vect**.

For similar reasons, we do not extend the functionality of the 2-matrix operations we just defined to handle 2-matrices whose outer dimensions are zero. In some small situations, it is possible for them to arise, however they are not needed to represent the modular tensor categories of interest to TQC.[29]

3.3 2Vect, the representation

We finally have the required functionality to implement the semisimple symmetric monoidal 2-category **2Vect** using a computer algebra system. This Section will heavily mirror Section 2.2, providing implementations for all the features defined there.

3.3.1 0-Cells

As stated in Section 2.2.1, there is a isomorphism between *0-cells* of **2Vect** and the whole numbers. Thus, we will use this isomorphism to represent 0-cells as whole numbers.

Since we are not just implementing **2Vect**, but we are also endowing it with a 2-monoidal structure, we can combine 0-cells using the monoidal product 2-functor, $\boxtimes$. This has the action of simply multiplying the numbers used to represent the 0-cells. For instance, in our representation, $\mathbf{2} \boxtimes \mathbf{3} = \mathbf{6}$, meaning the monoidal product of two copies of **2Vect** and three copies of **2Vect** is

[28] N.B. In equation 47, the symbol $\otimes$ is used to mean the standard tensor product, and the symbol $\boxtimes$ is used to represent the two-tensor product we are defining. However, later we will begin to use $\boxtimes$ mainly to represent the product 2-functor for monoidal 2-categories.

[29] However, most relevant 2-matrices *will* have some zero-dimensional matrices as element-matrices, as we saw in Section 2.3.3 for the representation of **Fib**.

2Vect	label	actual representation	isomorphism
0-cell	$\mathbf{2}$	$\begin{bmatrix} \mathbf{Vect}_a \\ \mathbf{Vect}_b \end{bmatrix}$	2
1-cell	$f : \mathbf{2} \to \mathbf{3}$	$\begin{bmatrix} \mathbb{C}^n & \mathbb{C}^m \\ \mathbb{C}^p & \mathbb{C}^q \\ \mathbb{C}^r & \mathbb{C}^s \end{bmatrix}$	$\begin{bmatrix} n & m \\ p & q \\ r & s \end{bmatrix}$
"state" of 0-cell: $\mathbf{2}$	$\psi \in \mathbf{2}$	$\begin{bmatrix} \mathbb{C}^a \\ \mathbb{C}^b \end{bmatrix}$	$\begin{bmatrix} a \\ b \end{bmatrix}$
"state" as a 1-cell	$\psi : \mathbf{1} \to \mathbf{2}$	$\begin{bmatrix} \mathbb{C}^a \\ \mathbb{C}^b \end{bmatrix}$	$\begin{bmatrix} a \\ b \end{bmatrix}$

Table 2: Example of a 0-cell, 1-cell, and "states" of a 0-cell for **2Vect**. The 0-cell is actually two copies of the category **Vect**, but we represent it as the number 2. The 1-cell is actually a 3×2 matrix of vector spaces, but we represent it as a 3×2 matrix of numbers. A state of $\mathbf{2}$ is a 2×1 matrix of vector spaces (i.e. a column vector), but we represent it as a 2×1 matrix of numbers. Furthermore, the equivalence between the two ways of thinking about "states" should be clear. From the categorical viewpoint of **2Vect**, we do not "look inside" the 0-cells to find the states, but can access them via 1-cells. We can then apply morphisms to them via 1-cell composition. (q.v. footnote 8 for analogy to **Vect**.)

isomorphic to the 0-cell representing six copies of **2Vect**. Thus, the monoidal product represents all the ways of pairing together a copy of **Vect** from each of the factors.[30] Additionally, from this it should be clear that $\mathbf{1}$ is the unit object of the 2-monoidal structure of **2Vect**, $\mathbf{1} \boxtimes \mathbf{N} = \mathbf{N} \boxtimes \mathbf{1} = \mathbf{N}$, and that the left and right unitors are the identity.

Furthermore, $\mathbf{1}$ is simply equal to the category **Vect**, and $\mathbf{0}$ represents no copies of **Vect**. For the modular tensor categories of interest, we will not make any use of $\mathbf{0}$.

3.3.2 1-Cells

As there is an isomorphism between 0-cells and whole numbers, there is also an isomorphism between matrices of vector spaces and matrices of whole numbers. Thus, rather than the complicated generalized matrix product we described in Section 2.2.2, we can use the standard matrix product to compose our representation of 1-cells and generally treat them as matrices of numbers. An example of the actual and isomorphic representations of 0-cells and 1-cells is shown in Table 2.

[30] N.B. The 0-cells $\mathbf{2} \boxtimes \mathbf{3}$, $\mathbf{3} \boxtimes \mathbf{2}$, and $\mathbf{6}$ are *different* but isomorphic objects. However, they are equal in our representation, since as numbers, $2 \cdot 3 = 3 \cdot 2 = 6$. We will see in the next section how this *strictification* in our representation leads to the need for additional categorical structures—in this case, the swap 1-cell.

1-Cell Composition: As 1-cells are represented by matrices, the composition of 1-cells f : $\mathbf{N} \to \mathbf{M}$ and g : $\mathbf{M} \to \mathbf{P}$ is simply given by the matrix product gf. We will always write the first 1-cell on the right and the last 1-cell on the left, to highlight the isomorphism between one cells and matrix operators acting on vectors. Usually, when it is clear that the matrices we are referring to are 1-cells, we will write 1-cell composition as $g \circ f$.

Identity 1-Cell: The identity 1-cell $id_{\mathbf{N}}$ is given by the $n \times n$ identity matrix.

Inverse 1-Cell: The inverse 1-cell of f, if it exists, will be given by the matrix inverse of f.

Monoidal Product is given by the tensor product or Kronecker product of the matrices representing the 1-cells. Thus, for 1-cells represented by the matrices f and g, the matrix representing their monoidal product is given by $f \otimes g$.[31]

Swap 1-Cell: The swap 1-cell is implemented by an explicit construction of a swap matrix from the input of the two objects to be swapped. Essentially, the matrix must reverse the order of the factors in the original tensor product. For example, consider $\mathbf{2} \boxtimes \mathbf{3}$, which is isomorphic to $\mathbf{6}$. In its actual representation, $\mathbf{2} \boxtimes \mathbf{3}$ is six copies of **Vect** constructed from $\mathbf{2}$ and $\mathbf{3}$ in the following manner.

$$\begin{bmatrix} \mathbf{Vect}_{\mathbf{6}_a} \\ \mathbf{Vect}_{\mathbf{6}_b} \\ \mathbf{Vect}_{\mathbf{6}_c} \\ \mathbf{Vect}_{\mathbf{6}_d} \\ \mathbf{Vect}_{\mathbf{6}_e} \\ \mathbf{Vect}_{\mathbf{6}_f} \end{bmatrix} = \begin{bmatrix} \mathbf{Vect}_{\mathbf{2}_a} \boxtimes \mathbf{Vect}_{\mathbf{3}_a} \\ \mathbf{Vect}_{\mathbf{2}_a} \boxtimes \mathbf{Vect}_{\mathbf{3}_b} \\ \mathbf{Vect}_{\mathbf{2}_a} \boxtimes \mathbf{Vect}_{\mathbf{3}_c} \\ \mathbf{Vect}_{\mathbf{2}_b} \boxtimes \mathbf{Vect}_{\mathbf{3}_a} \\ \mathbf{Vect}_{\mathbf{2}_b} \boxtimes \mathbf{Vect}_{\mathbf{3}_b} \\ \mathbf{Vect}_{\mathbf{2}_b} \boxtimes \mathbf{Vect}_{\mathbf{3}_c} \end{bmatrix} \tag{48}$$

where $\mathbf{Vect}_{\mathbf{3}}$ is taken to mean a copy of **Vect** that came from $\mathbf{3}$ and the roman letter indexes

[31] The 2-monoidal product will always be written with $\boxtimes$, even when referring to the matrix representation and the normal matrix tensor product. This is to avoid confusing with the monoidal product we will define for our modular tensor categories, which will be written with $\otimes$.

which copy. After the application of the swap 1-cell $S_{\mathbf{2},\mathbf{3}}$, it should be in the form

$$\begin{bmatrix} \mathbf{Vect}_{\mathbf{3}_a} \boxtimes \mathbf{Vect}_{\mathbf{2}_a} \\ \mathbf{Vect}_{\mathbf{3}_a} \boxtimes \mathbf{Vect}_{\mathbf{2}_b} \\ \mathbf{Vect}_{\mathbf{3}_b} \boxtimes \mathbf{Vect}_{\mathbf{2}_a} \\ \mathbf{Vect}_{\mathbf{3}_b} \boxtimes \mathbf{Vect}_{\mathbf{2}_b} \\ \mathbf{Vect}_{\mathbf{3}_c} \boxtimes \mathbf{Vect}_{\mathbf{2}_a} \\ \mathbf{Vect}_{\mathbf{3}_c} \boxtimes \mathbf{Vect}_{\mathbf{2}_b} \end{bmatrix}$$

The fact that $\mathbf{2} \boxtimes \mathbf{3} = \mathbf{3} \boxtimes \mathbf{2} = \mathbf{6}$ in our representation, but they are not equal 0-cells, means that the swap goes from the 0-cell $\mathbf{6}$ to itself. This is due to the strictification induced by our representation. Thus, $S_{\mathbf{2},\mathbf{3}} : \mathbf{6} \to \mathbf{6}$ keeps the first element the same, moves the second element to the third element, moves the third element to the fifth element, and all the rest.[32] Its matrix representation is

$$S_{\mathbf{2},\mathbf{3}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Its construction is determined by where an element is located in the first column and where it needs to move to in the swapped column. For instance, the third element moves to the fifth element, so $(S_{\mathbf{2},\mathbf{3}})_{5,3} = 1$. The column indicates where it's moving from and the row indicates where it is moving to. All the other entries are zeroes. This makes sense given the rules of the matrix product and linear maps. Furthermore, it's easy to see that $S_{\mathbf{N},\mathbf{M}}$ will be unitary and that $S_{\mathbf{N},\mathbf{M}} = (S_{\mathbf{M},\mathbf{N}})^{\mathrm{T}}$. Thus, we will have $S_{\mathbf{M},\mathbf{N}} \circ S_{\mathbf{N},\mathbf{M}} = id_{\mathbf{N}\boxtimes\mathbf{M}}$, as required.

[32] The order $\mathbf{Vect}_{\mathbf{3}_a} \boxtimes \mathbf{Vect}_{\mathbf{2}_a}$ versus $\mathbf{Vect}_{\mathbf{2}_a} \boxtimes \mathbf{Vect}_{\mathbf{3}_a}$ does not matter since they are isomorphic. It is true that $\mathbf{2} \boxtimes \mathbf{3}$ and $\mathbf{3} \boxtimes \mathbf{2}$ are isomorphic as well, and we are constructing exactly that isomorphism. However, the order of the copies of $\mathbf{Vect}$ within an element of the column doesn't matter in our representation from the 2-categorical viewpoint.

Finally, we need a method for determining where the elements move to. This can be easily seen in the above example, where

$$\begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix} \rightarrow \begin{bmatrix} 1 \\ 4 \\ 2 \\ 5 \\ 3 \\ 6 \end{bmatrix}$$

The first element will always stay the same. The next element is the fourth, or 3 greater. In general, the next element is always taken from the current position plus m when computing $S_{\mathbf{N},\mathbf{M}}$. However, if the position exceeds $n \times m$, the total number of elements in the column, then it wraps around plus 1. For instance, 3 more than the fourth element is 7, minus 2×3 is 1, plus 1 is 2. So,

$$x_{i+1} = \begin{cases} x_i + m, & x_i + m \leq nm \\ 1 + [(x_i + m) \bmod nm], & x_i + m > nm \end{cases} \tag{49}$$

where i is the old position, x_i is the new position, and x_{i+1} is the next new position for the old position $i + 1$. From this, we can find x_i as a function of i

$$x_i = 1 + [(i-1)m \bmod nm] + \frac{i - 1 - [(i-1) \bmod n]}{n} \tag{50}$$

Then, for all x_i, we have

$$(S_{\mathbf{N},\mathbf{M}})_{x_i,i} = 1 \tag{51}$$

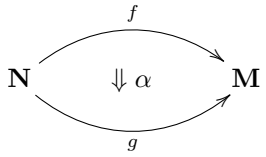
and 0 everywhere else. Finally, if either $\mathbf{M}$ or $\mathbf{N}$ equal 0, then $S_{\mathbf{N},\mathbf{M}}$ will simply be given by the 0×0 matrix.[33]

[33] The swap is exactly akin to a change of basis in the column representation of 0-cell "states," and S is the change of basis map. While categorical viewpoint abstracts away from bases and coordinates, it is sometimes helpful to take this viewpoint, as in Section 3.4.

3.3.3 2-Cells

The 2-cells of ***2Vect*** *are 2-matrices.*

As with 1-cells, the column dimension relates to the source and the row dimension relates to the destination. For 2-cells, the outer dimension row (column) counts the number of simple objects or the number of copies of **Vect** of the destination (source) 0-cells of the 1-cells. For instance, consider the 2-cell given by the following diagram,



As a 2-matrix, the outer dimensions of α are $m \times n$ (i.e. there are $m \times n$ element-matrices).

Additionally, both f and g are $m \times n$ matrices. Thus, each element-matrix of a 2-cell has a corresponds to an element in both the source and destination 1-cells (which represents the dimension of a vector space). Each element-matrix is a linear map from a vector space given by the element in the 1-cell source to the vector space given by the element in the 1-cell destination, and these numbers will be the dimensions of the corresponding element-matrix.[34] Ergo, the element-matrix α_{ij} will be dimension $g_{ij} \times f_{ij}$.

Consider the following example. If $f : \mathbf{2} \to \mathbf{3}$ and $g : \mathbf{2} \to \mathbf{3}$, they will both be 3×2 dimensional matrices going from 2 copies of **Vect** to 3 copies of **Vect**. Each element represents a vector space of dimension given by that element. Let them be given by

$$f = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 2 & 1 \end{bmatrix} \tag{52}$$

[34] This makes the elements of the element-matrices *actually* numbers (i.e. whatever the ground field of the **Vect** is) and not representations isomorphic to some other structure.

$$g = \begin{bmatrix} 3 & 4 \\ 2 & 5 \\ 2 & 1 \end{bmatrix}$$

then $\alpha : f \Rightarrow g$ would have the form

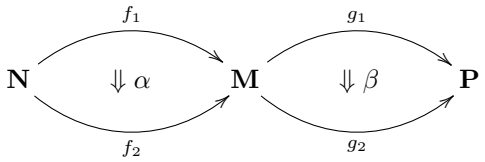
$$\begin{bmatrix} \begin{bmatrix} a_{1,1} \\ a_{2,1} \\ a_{3,1} \end{bmatrix} & \begin{bmatrix} b_{1,1} & b_{1,2} \\ b_{2,1} & b_{2,2} \\ b_{3,1} & b_{3,2} \\ b_{4,1} & b_{4,2} \end{bmatrix} \\ \begin{bmatrix} c_{1,1} & c_{1,2} & c_{1,3} \\ c_{2,1} & c_{2,2} & c_{2,3} \end{bmatrix} & \begin{bmatrix} d_{1,1} & d_{1,2} & d_{1,3} & d_{1,4} \\ d_{2,1} & d_{2,2} & d_{2,3} & d_{2,4} \\ d_{3,1} & d_{3,2} & d_{3,3} & d_{3,4} \\ d_{4,1} & d_{4,2} & d_{4,3} & d_{4,4} \\ d_{5,1} & d_{5,2} & d_{5,3} & d_{5,4} \end{bmatrix} \\ \begin{bmatrix} e_{1,1} & e_{1,2} \\ e_{2,1} & e_{2,2} \end{bmatrix} & [f_{1,1}] \end{bmatrix} \tag{53}$$

where $\{a_{i,j}, ..., f_{1,1}\}$ are all of the ground field (in our case almost always $\mathbb{C}$). Here we exploit no isomorphism and represent 2-cells exactly as they are.

We define 2-cells of **2Vect** to have the following properties / endow them with the following operations:

2-Cell Vertical Composition essentially entails element-wise composing the linear maps of the 2-cell, which of course results in a new collection of linear maps. The vertical composition of 2-cells $\alpha : f \Rightarrow g$ and $\beta : g \Rightarrow h$ to get $\beta \circ \alpha : f \Rightarrow h$ will be given by the *element-matrix product* $\beta \circ \alpha$ defined in Section 3.3.3 by equation 43.

2-Cell Horizontal Composition: Given the following definition of 1-cells and 2-cells,

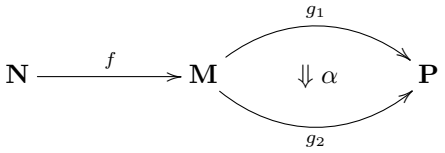


the 2-cell horizontal composition of α and β is given by *outer-matrix product* $\beta \bullet \alpha$ defined in Section 3.3.3 by equation 44.[35]

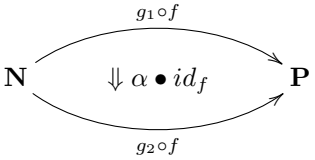
Identity 2-Cell: If $\gamma : f \Rightarrow f$, the identity 2-cell id_f is given by the *identity 2-matrix* on γ, defined in Section 3.3.3 by equation 45.

Inverse 2-Cell is given by the *inverse 2-matrix* defined in Section 3.3.3 by equation 46.

Whiskering: There is no special implementation for this operation with 2-matrices. Given the following diagram



The identity 2-matrix of f is id_f and the whiskering is given by $\alpha \bullet id_f$.[36]

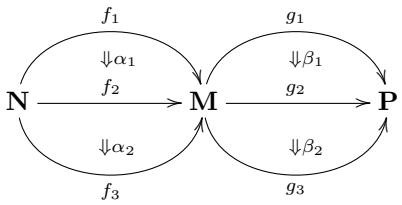


[35] Note that the order of α and β in $\beta \bullet \alpha$ is consistent with the order of the outer-matrix product and with the order of 1-cell composition, but reverses the order of α and β in the first diagram. Many authors choose the opposite ordering, but we will write it this way to highlight the fact that 2-cells are 2-matrices and horizontal composition is simply a higher level form the composition of operators (which are often written to the left of that which they act on).

[36] Some authors will write the whiskering as $\alpha \bullet f$. While usually clear from context, this notation suggests that a 2-cell α and a 1-cell f can be combined using the outer-matrix product operation, $\bullet$. We will always write whiskering as a horizontal composition between 2-cells (i.e. $\alpha \bullet id_f$).

Monoidal Product on 2-cells is given by the *two-tensor product* on the 2-matrices α and β defined in Section 3.3.3 by equation 47. This will be written as $\alpha \boxtimes \beta$.

Interchange Law: The interchange law stated that given the following diagram



(54)

we can first compose vertically and then compose horizontally, to give $(\beta_2 \circ \beta_1) \bullet (\alpha_2 \circ \alpha_1) : g_1 \circ f_1 \Rightarrow g_3 \circ f_3$, or we can first compose horizontally and then compose vertically, to give $(\beta_2 \bullet \alpha_2) \circ (\beta_1 \bullet \alpha_1) : g_1 \circ f_1 \Rightarrow g_3 \circ f_3$, and those 2-cells must be equal. We can bring further light to the interchange law by noting that statements about 2-cells are also statements about 2-matrices. The interchange law is, in fact, a 2-matrix identity, $(\beta_2 \circ \beta_1) \bullet (\alpha_2 \circ \alpha_1) = (\beta_2 \bullet \alpha_2) \circ (\beta_1 \bullet \alpha_1)$, which can be proven using the definition of the element-matrix product and the outer-matrix product.

3.4 Structural Isomorphisms

While working with *symmetric 2-monoidal 2-categories* like **2Vect**, we quickly find that many expressions that are equal when working with *symmetric monoidal categories* will only hold up to isomorphism. The presence of 2-cells provides an extra "degree of freedom" in the structure by which certain equations no longer hold.

For instance, in the definition of a symmetric monoidal category (for instance, **Vect**), we are told $S \circ (f \otimes g) = (g \otimes f) \circ S$, where f and g are morphisms, and S is the swap operation as defined in Section 3.3.2 [12].[37] In **2Vect**, these 1-cells will no longer be of the same *type*. Furthermore, related equations between 2-cells will no longer hold. To mediate between type, **2Vect** is endowed with a structural 2-cell isomorphism, denoted $\sigma_{f,g}$, to allow monoidally combined 1-cells to commute with the swap.[38]

[37] Technically, in Section 3.3.2 we were referring to the swap operation for a *symmetric monoidal 2-category*. The difference will shortly be made clear.

[38] The 2-cell is denoted $\sigma_{f,g}$ because it is the structure tailored to two *specific* 1-cells, f and g, that is a 2-cell. The

However, by no longer of the same *type*, we mean something subtle. The two 1-cells, $S \circ (f \boxtimes g)$ and $(g \boxtimes f) \circ S$, must always give the same functor (i.e. the same 1-cell matrix) in our representation. But these are not the *same* one cell, even if they have the same representation. This is due to the strictification implicit in our representation of 1-cells, our treating matrices of vector spaces as matrices of numbers. A similar problem occurred with our representation of 0-cells, requiring the swap 1-cell. Furthermore, composed parallel 1-cells leads to horizontal composition in the 2-cells that go between them. At the 2-cell level, we will have different 2-cells, and without any subtlety, $id_S \bullet (\mu \boxtimes \nu) \neq (\nu \boxtimes \mu) \bullet id_S$, where $\mu : f \Rightarrow f'$ and $\nu : g \Rightarrow g'$.[39] Essentially, this is just a statement about 2-matrices, and in many cases about commutativity or associativity of the underlying tensor products and direct sums. From a non-categorical perspective, going from $S \circ (f \boxtimes g)$ to $(g \boxtimes f) \circ S$ will induce an implicit change of basis either the element-matrices or outer-matrices of the associated 2-cells (or both).

We will list three such discovered structures required for realizing modular tensor categories in **2Vect** and discuss two other cases where it could be possible for a structure to arise, but argue that these cases will always be equal to the identity 2-cell. For the three that require structural isomorphisms, we will explain the equation that only holds up to isomorphism and then explicitly construct the structure for **2Vect**.[40]

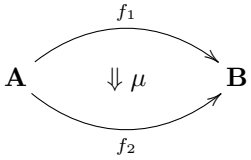
components of σ are 2-cells, and σ itself is some higher level 2-transformation. (cf. the relationship between natural transformations and their component arrows). This will be true of all of the structural isomorphism presented in this section.

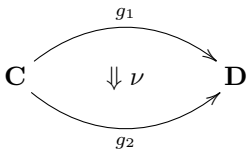
[39] This is not a surprise since we are not using any strictification in our representation of 2-cells—we are representing them as they actually are (q.v. Section 3.3.3).

[40] While one may think that explicitly constructing 2-cells would be difficult, it is made significantly easier by the use of generalized operations. First, we determine how to solve one element-matrix of the structural isomorphism, and then we use a generalized operation to get the entire 2-matrix solution. The generalized operation required is the one that will provide the correct form for the structural isomorphism. For instance, for σ, the 1-cells are combined using the monoidal product, $\boxtimes$, which is just the tensor product. Thus, constructing σ involves a generalized tensor product between those 1-cells.

3.4.1 Swap Structure: σ

As stated above, $S \circ (f \otimes g)$ and $(g \otimes f) \circ S$ are not of the same type in **2Vect**. Instead, if we have the following definitions of 1-cells and 2-cells

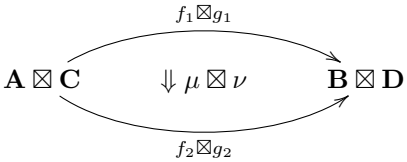


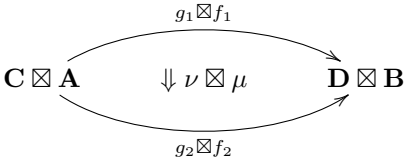


$$\mathbf{A} \boxtimes \mathbf{C} \xrightarrow{S_{\mathbf{A},\mathbf{C}}} \mathbf{C} \boxtimes \mathbf{A}$$

$$\mathbf{B} \boxtimes \mathbf{D} \xrightarrow{S_{\mathbf{B},\mathbf{D}}} \mathbf{D} \boxtimes \mathbf{B}$$

We can monoidally combine the 1-cells in two different ways





Just considering the 1-cells with the subscript 1 for the moment, the first these diagrams can be post-composed with $S_{\mathbf{B},\mathbf{D}}$ to get $S_{\mathbf{B},\mathbf{D}} \circ (f_1 \otimes g_1)$, while the second can be pre-composed with $S_{\mathbf{A},\mathbf{C}}$

to get $(g_1 \otimes f_1) \circ S_{\mathbf{A},\mathbf{C}}$. However, because of the freedom in our 2-category, these do not have to necessarily be equal, but can be related by a 2-cell

$$\mathbf{A} \boxtimes \mathbf{C} \quad \overset{S_{\mathbf{B},\mathbf{D}} \circ (f_1 \boxtimes g_1)}{\underset{(g_1 \boxtimes f_1) \circ S_{\mathbf{A},\mathbf{C}}}{\Downarrow \sigma_{f_1,g_1}}} \quad \mathbf{D} \boxtimes \mathbf{B} \tag{55}$$

Furthermore, for coherence, the following diagram must commute

$$\begin{array}{ccc} S_{\mathbf{B},\mathbf{D}} \circ (f_1 \boxtimes g_1) & \xRightarrow{\sigma_{f_1,g_1}} & (g_1 \boxtimes f_1) \circ S_{\mathbf{A},\mathbf{C}} \\ \Big\Downarrow {\scriptstyle id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu)} & & \Big\Downarrow {\scriptstyle (\nu \boxtimes \mu) \bullet id_{S_{\mathbf{A},\mathbf{C}}}} \\ S_{\mathbf{B},\mathbf{D}} \circ (f_2 \boxtimes g_2) & \xRightarrow[\sigma_{f_2,g_2}]{} & (g_2 \boxtimes f_2) \circ S_{\mathbf{A},\mathbf{C}} \end{array} \tag{56}$$

where the 0-cells have been omitted for clarity.[41] Here lies the problem with assuming $S_{\mathbf{B},\mathbf{D}} \circ (f_1 \boxtimes g_1)$ and $(g_1 \boxtimes f_1) \circ S_{\mathbf{A},\mathbf{C}}$ are the same. If σ is the identity, then we would have $id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu) = (\nu \boxtimes \mu) \bullet id_{S_{\mathbf{A},\mathbf{C}}}$, which is a statement about 2-matrices that is generally incorrect. Essentially, we find that the two-tensor product (defined by equation 47), like the normal matrix product, does not commute for arbitrary factors. However, it will be true that $(g_1 \boxtimes f_1) \circ S_{\mathbf{A},\mathbf{C}}$ will be the same functor as $S_{\mathbf{B},\mathbf{D}} \circ (f_1 \boxtimes g_1)$, and thus be represented by the same matrix. Thus, σ must be applied to go between these 1-cells, otherwise 2-cell equations will be incorrect.

Explicit Construction Like the swap 1-cell described in Section 3.3.2, σ will be explicitly constructed, given the 1-cells to be interchanged with the swap. The swap 1-cell swapped the order of monoidally combined 0-cells, which in practice meant reversing the order of two columns tensored together. The structure σ increases the level of abstraction twofold.

First, rather than fixing columns tensored together, we have to fix matrices tensored together. Considering the swap as a change of basis, since our 0-cells transformed like $\vec{x'} = S\vec{x}$ (where $\vec{x}$ just means our column representation for states of the 0-cells, see equation 48), then schematically our

[41] N.B. We can think of $id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu)$ as the whiskering of 1-cell $S_{\mathbf{B},\mathbf{D}}$ with 2-cell $\mu \boxtimes \nu$.

matrices will transform as $A' = SAS^{-1}$.[42] The upside is that if we come up with a transformation to reorder the rows (as we did for our 0-cells represented in a column), we will automatically reorder the columns by applying the inverse (or transpose, since the swap matrices are unitary) of the swap to the other side.

Second, the 2-cells are matrices of matrices, and the two-tensor product operates on both the outer-matrices and inner-matrices. However, this problem is rendered trivial by the use of generalized operations, q.v. footnote 40. Formally, to reorder $(\mu \boxtimes \nu)$ to $(\nu \boxtimes \mu)$, we need to swap both the order of the inner-matrices and the outer-matrices (i.e. we need to reorder the element-matrices and the elements of the element-matrices). However, due to the use of generalized operations, the explicit construction of our swap matrix (the swap 1-cell) is all we need.

However, we are not reordering $(\mu \boxtimes \nu)$ to $(\nu \boxtimes \mu)$. We are reordering $id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu)$ to $(\nu \boxtimes \mu) \bullet id_{S_{\mathbf{A},\mathbf{C}}}$. It turns out that id_S horizontally composed with a 2-cell is exactly the operation that reorders the element-matrices. Thus, we are left with reordering the elements of the element-matrices. However, we already know how to reorder (i.e. change the basis) of normal matrices via the swap 1-cell.

We now need a way to create one for all the possible element-matrix combinations from the input of a pair of 1-cells, such as f_2 and g_2. Each element of f_2 will be multiplied each element of g_2 as the row dimension for each of the element-matrices of $\mu \boxtimes \nu$. We want a swap 1-cell to reorder the rows for each of these element-matrices. The matrix operation that creates pairs of each element from one matrix with every element from another is a generalized tensor product (equation 40). To get a swap 1-cell for each pair, we use the swap 1-cell operation (equations 50 and 51) as the generalized times.[43] For each element in the form of a tensor product between f_2 and g_2, we will get a swap 1-cell element-matrix that, when applied to the proper element-matrix of $\mu \boxtimes \nu$, will fix the order of the element-numbers. Let us call generalized tensor product with swap 1-cell as the generalized times σ'.

However, since σ_{f_2,g_2} is not being post-composed with $(\mu \boxtimes \nu)$ but rather $id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu)$, $\sigma'_{f_2,g_2} \neq \sigma_{f_2,g_2}$. Since $id_{S_{\mathbf{B},\mathbf{D}}}$ already reordered the outer-matrices of $(\mu \boxtimes \nu)$, we need to reorder

[42] Using the fact that σ is invertible, this essentially the content of equation 56. However, technically this swap will be not the same as the one on the other side of A since they will be over different 0-cells. Explicitly we have $(\nu \boxtimes \mu) \bullet id_{S_{\mathbf{A},\mathbf{C}}} = (\sigma_{f_2,g_2}) \circ \left(id_{S_{\mathbf{B},\mathbf{D}}} \bullet (\mu \boxtimes \nu)\right) \circ (\sigma_{f_1,g_1})^{-1}$, which is, in spirit, the same as $A' = SAS^{-1}$.

[43] N.B. The *generalized times* can be any binary operation. It is labeled as such because it is applied to the matrix factors in the *form* of the tensor product times.

the outer-matrices of σ' so that the element-matrices will be compatible under vertical composition. Thus, we also need to vertically compose with $id_{S_{\mathbf{B},\mathbf{D}}}$, giving

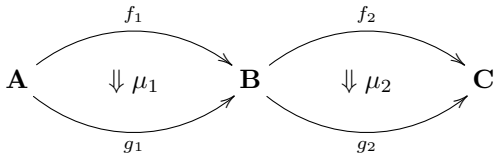
$$\sigma_{f_2,g_2} = id_{S_{\mathbf{B},\mathbf{D}}} \circ \sigma'_{f_2,g_2} \tag{57}$$

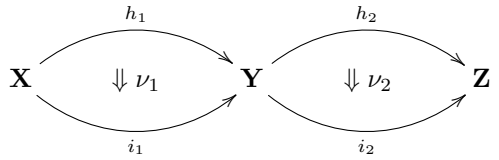
If one of the input 1-cells (i.e. f_1, f_2, g_1, or g_2) is a zero-dimensional matrix, then σ will have outer dimensions 0×0. This is one of the few situations where this 2-matrix arises, however, as stated in Section 3.2.2, they will not be needed for modular tensor categories.

3.4.2 Interchange Structure: τ

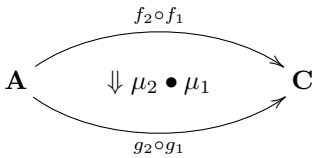
A structure is needed to mediate the interchange of the operations $\boxtimes$ and $\bullet$ for 2-cells.

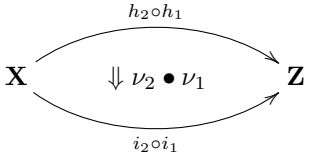
Given the following definitions of 1-cells and 2-cells



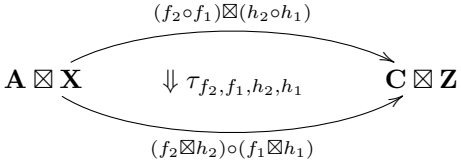


in each diagram, the 2-cells may be composed horizontally to give



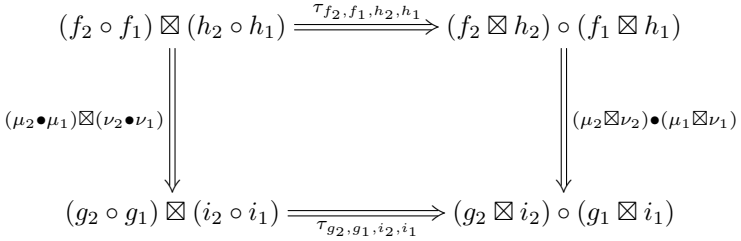


Just considering the 1-cells f_j and h_j for the moment, the 1-cells on the top of these last two diagrams can be monoidally combined to get $(f_2 \circ f_1) \boxtimes (h_2 \circ h_1)$. However, f_1 and h_1 can be monoidally combined first before being composed with a monoidally combined f_1 and h_1 to give $(f_2 \boxtimes h_2) \circ (f_1 \boxtimes h_1)$, which is the same 1-cell and will have the same matrix representation. For a 1-category, these would have to be of the same type. However, because of the freedom in our 2-category, these do not have to be the same, but can be related by a 2-cell



(58)

Furthermore, for coherence, the following diagram must commute



(59)

where again the 0-cells have been omitted for clarity. As in Section 3.4.1, the functor given by the source and destination 1-cells of τ will be the same (they will be represented by the same matrix). However, τ is always the identity, then we would have $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1) = (\mu_2 \boxtimes \nu_2) \bullet (\mu_1 \boxtimes \nu_1)$, which is a statement about 2-matrices that does not hold for arbitrary μ_1, μ_2, ν_1, and ν_2. Fundamentally, the issue is that the tensor product does not generally distribute over the direct sum.[44]

[44] N.B. This is a different interchange from the one introduced by equation 14. Here, τ mediates an interchange of $\boxtimes$ and $\circ$. The one given by equation 14 is an interchange of $\circ$ and $\bullet$, and is a *law* of 2-categories.

Explicit Construction We will also need to explicitly construct τ, given the four 1-cells to be interchanged. Our 2-cells will still transform schematically as $A' = \tau A \tau^{-1}$, so we will still only need to reorder the rows of the inner- and outer-matrices of our 2-cells (q.v. **Explicit Construction** in Section 3.4.1 and footnote 42).

As stated in footnote 40, the construction of a structural isomorphism is given by a combination of generalized operations linked to achieve the appropriate form. For τ, it will involve two generalized matrix products (equation 37) and the generalized tensor product (equation 40).

The element-matrices of the 2-cell $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$ will have row dimensions given by $(g_2 \circ g_1) \boxtimes (i_2 \circ i_1)$ and column dimensions given by $(f_2 \circ f_1) \boxtimes (h_2 \circ h_1)$. Vertically post-composing τ_{g_2,g_1,i_2,i_1} with $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$ will fix the row ordering problem. Thus, τ_{g_2,g_1,i_2,i_1} must be able to be vertically post-composed with $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$ and the resulting 2-cell $\tau_{g_2,g_1,i_2,i_1} \circ ((\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1))$ must have the same element-matrix row dimensions as the original. Therefore, τ_{g_2,g_1,i_2,i_1} must have the same outer-matrix dimensions as $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$ and have square element-matrices given by the row dimension of the element matrices of $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$, which is given by the elements of $(g_2 \circ g_1) \boxtimes (i_2 \circ i_1)$. Ergo, the form of τ_{g_2,g_1,i_2,i_1} will be dictated by the generalized matrix product on $g_2 \circ g_1$ and the generalized matrix product on $i_2 \circ i_1$ the results then combined by a generalized tensor product. This will be sure to combine the elements of the 1-cells in the correct form and create the correct number of element-matrices in the correct place. However, we have not yet discussed the content of these element-matrices, which will be set by defining the generalized times and generalized plus for the generalized matrix product and the generalized times for the generalized tensor product. The generalized matrix products will be operations designed to generate lists of the necessary parameters. Given such a list, the generalized tensor product will solve the individual element-matrices and yield the correct form for τ.

Consider a single element-matrix of $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$. Even simpler, consider one column of a single element-matrix—remember, we only have to reorder the rows. The number of rows will be given by the sum and product of some different numbers: a row of the matrix g_2, a column of the matrix g_1, a row of the matrix i_2, and a column of the matrix i_1. In fact, the g's multiplied in a vector product, the i's will be multiplied in a vector product, and then those numbers will be (scalar) multiplied. Alternatively, the same number can be reached by first taking the tensor product of the matrices indexed by 2 and a separate tensor product of the matrices indexed by 1

and then taking a row from the 2's, a column from the 1's, and finally combining them in a vector product.

Let us consider the general case. Consider the following definitions for $g_2 : \mathbf{B} \to \mathbf{C}$, $g_1 : \mathbf{A} \to \mathbf{B}$, $i_2 : \mathbf{Y} \to \mathbf{Z}$, and $i_1 : \mathbf{X} \to \mathbf{Y}$

$$g_2 = \begin{bmatrix} a_1 & \cdots & a_B \\ \vdots & \ddots & \vdots \end{bmatrix}$$

$$g_1 = \begin{bmatrix} b_1 & \cdots \\ \vdots & \ddots \\ b_B & \cdots \end{bmatrix}$$

$$i_2 = \begin{bmatrix} c_1 & \cdots & c_Y \\ \vdots & \ddots & \vdots \end{bmatrix}$$

$$i_1 = \begin{bmatrix} d_1 & \cdots \\ \vdots & \ddots \\ d_Y & \cdots \end{bmatrix}$$

The two different ways of combining them gives two equivalent-in-value-but-different-in-form $CZ \times AX$ dimensional 1-cells, given by

$$(g_2 \circ g_1) \boxtimes (i_2 \circ i_1) = \begin{bmatrix} (a_1b_1 + \cdots + a_Bb_B) \times (c_1d_1 + \cdots + c_Yd_Y) & \cdots \\ \vdots & \ddots \end{bmatrix}$$

$$(g_2 \boxtimes i_2) \circ (g_1 \boxtimes i_1) = \begin{bmatrix} a_1c_1b_1d_1 + \cdots + a_1c_Yb_1d_Y + \cdots + a_Bc_1b_Bd_1 + \cdots + a_Bc_Yb_Bd_Y & \cdots \\ \vdots & \ddots \end{bmatrix}$$

where the order of all the factors has been preserved in the operation performed.

Consider the top left cell (the only one we show). As numbers, it is true that $(a_1b_1 + \cdots + a_Bb_B) \times (c_1d_1 + \cdots + c_Yd_Y) = a_1c_1b_1d_1 + \cdots + a_1c_Yb_1d_Y + \cdots + a_Bc_1b_Bd_1 + \cdots + a_Bc_Yb_Bd_Y$. However, from these 1-cells we ascertain that the top left element-matrix of $(\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1)$ is schematically ordered as $(\mathbf{a_1} \otimes \mathbf{b_1} \oplus \cdots \oplus \mathbf{a_B} \otimes \mathbf{b_B}) \otimes (\mathbf{c_1} \otimes \mathbf{d_1} \oplus \cdots \oplus \mathbf{c_Y} \otimes \mathbf{d_Y})$, while the top left element-matrix of $(\mu_2 \boxtimes \nu_2) \bullet (\mu_1 \boxtimes \nu_1)$ is schematically ordered as $(\mathbf{a_1} \otimes \mathbf{c_1} \otimes \mathbf{b_1} \otimes \mathbf{d_1}) \oplus \cdots$

$\oplus(\mathbf{a_1} \otimes \mathbf{c_Y} \otimes \mathbf{b_1} \otimes \mathbf{d_Y}) \oplus \cdots \oplus(\mathbf{a_B} \otimes \mathbf{c_1} \otimes \mathbf{b_B} \otimes \mathbf{d_1}) \oplus \cdots \oplus(\mathbf{a_B} \otimes \mathbf{c_Y} \otimes \mathbf{b_B} \otimes \mathbf{d_Y})$. By schematically ordered, we mean that μ_2 has an element matrix, $\mathbf{a_i}$, whose row size is given by a_i, and so on for all the rest. For instance, we represent $\mu_2 : f_2 \Rightarrow g_2$ as

$$\mu_2 = \begin{bmatrix} \mathbf{a_1} & \cdots & \mathbf{a_B} \\ \vdots & \ddots & \vdots \end{bmatrix}$$

where $\mathbf{a_i}$ means an a_i-by-unconcerned element-matrix.[45] These will be the smallest unit of *row chunk* or *row block* that needs to be reordered. Only showing one column, and representing the content of the rows as stated, the two different orderings of the top left element-matrix look like

$$((\mu_2 \bullet \mu_1) \boxtimes (\nu_2 \bullet \nu_1))_{1,1} = \begin{bmatrix} (\mathbf{a_1} \otimes \mathbf{b_1})_{1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{a_1 b_1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{a_1 b_1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{a_B b_B,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{a_B b_B,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \end{bmatrix} \tag{60}$$

[45] Hopefully, the difference between g_2 and the representation of μ_2 should be clear from the context. Essentially, a_i is a *number* representing the row size and $\mathbf{a_i}$ is an a_i-by-unconcerned *matrix*. Also within this scope of variables, $\mathbf{A}$ is the *0-cell* source of g_1 and A is its *whole number* representation.

$$((\mu_2 \boxtimes \nu_2) \bullet (\mu_1 \boxtimes \nu_1))_{1,1} = \begin{bmatrix} \mathbf{a_1} \otimes \mathbf{c_1} \otimes \mathbf{b_1} \otimes \mathbf{d_1} \\ \vdots \\ \mathbf{a_1} \otimes \mathbf{c_Y} \otimes \mathbf{b_1} \otimes \mathbf{d_Y} \\ \vdots \\ \mathbf{a_B} \otimes \mathbf{c_1} \otimes \mathbf{b_B} \otimes \mathbf{d_1} \\ \vdots \\ \mathbf{a_B} \otimes \mathbf{c_Y} \otimes \mathbf{b_B} \otimes \mathbf{d_Y} \end{bmatrix} \tag{61}$$

$$= \begin{bmatrix} (\mathbf{a_1} \otimes \mathbf{c_1})_{1,\cdot} \times (\mathbf{b_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{c_1})_{a_1 c_1,\cdot} \times (\mathbf{b_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{c_Y})_{1,\cdot} \times (\mathbf{b_1} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{c_Y})_{a_1 c_Y,\cdot} \times (\mathbf{b_1} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{c_1})_{1,\cdot} \times \mathbf{b_B} \otimes \mathbf{d_1} \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{c_1})_{a_B c_1,\cdot} \times \mathbf{b_B} \otimes \mathbf{d_1} \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{c_Y})_{1,\cdot} \times \mathbf{b_B} \otimes \mathbf{d_Y} \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{c_Y})_{a_B c_Y,\cdot} \times \mathbf{b_B} \otimes \mathbf{d_Y} \end{bmatrix}$$

where, with apologies for the abuse of notation, we use $(\mathbf{a_i} \otimes \mathbf{b_i})_{k,\cdot}$ to mean the element given by the kth row, $k = \{1, \ldots, a_i b_i\}$, with the dot indicating no concern for the column, of the matrix represented by $\mathbf{a_i} \otimes \mathbf{b_i}$. Thus, $(\mathbf{a_i} \otimes \mathbf{b_i})_{k,\cdot} \times (\mathbf{c_j} \otimes \mathbf{d_j})$ means a number, $(\mathbf{a_i} \otimes \mathbf{b_i})_{k,\cdot}$, scalar multiplies the matrix, $(\mathbf{c_j} \otimes \mathbf{d_j})$. In the following, we also use $i = \{1, \ldots, B\}$ and $j = \{1, \ldots, Y\}$ as indices. Equation 61 is written two ways to help decipher the notation.

From the numbers given by a_1, ..., a_i,..., a_B, b_1, ..., b_i,..., b_B, c_1, ..., c_j,..., c_Y, d_1, ...,

d_j,..., and d_Y, we want to find a matrix that reorders the column given by equation 60 to the column given by equation 61. As discussed above, the generalized matrix products will be used with the generalized tensor product to generate such a list of parameters for each element-matrix in τ, which will then be computed as follows. It will be a two step process. First, we will reorder to the following form

$$\begin{bmatrix} (\mathbf{a_1} \otimes \mathbf{b_1})_{1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{a_1 b_1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_1} \otimes \mathbf{b_1})_{a_1 b_1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{1,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{a_B b_B,\cdot} \times (\mathbf{c_1} \otimes \mathbf{d_1}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{1,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \\ \vdots \\ (\mathbf{a_B} \otimes \mathbf{b_B})_{a_B b_B,\cdot} \times (\mathbf{c_Y} \otimes \mathbf{d_Y}) \end{bmatrix} = \begin{bmatrix} \mathbf{a_1} \otimes \mathbf{b_1} \otimes \mathbf{c_1} \otimes \mathbf{d_1} \\ \vdots \\ \mathbf{a_1} \otimes \mathbf{b_1} \otimes \mathbf{c_Y} \otimes \mathbf{d_Y} \\ \vdots \\ \mathbf{a_B} \otimes \mathbf{b_B} \otimes \mathbf{c_1} \otimes \mathbf{d_1} \\ \vdots \\ \mathbf{a_B} \otimes \mathbf{b_B} \otimes \mathbf{c_Y} \otimes \mathbf{d_Y} \end{bmatrix} \tag{62}$$

which involves reordering the rows in equation 60. Since all the terms involving matrix $\mathbf{a_1} \otimes \mathbf{b_1}$ come before all the terms involving matrix $\mathbf{a_i} \otimes \mathbf{b_i}$, which come before all the terms involving matrix $\mathbf{a_B} \otimes \mathbf{b_B}$, we can generally solve the problem by solving the part involving $\mathbf{a_i}$ and then direct summing the matrices together for all i.

Focusing on terms involving $\mathbf{a_i}$, let us consider an arbitrary element of equation 60, such as $(\mathbf{a_i} \otimes \mathbf{b_i})_{k,\cdot} \times (\mathbf{c_j} \otimes \mathbf{d_j})$. This is a *matrix* with row dimension given by the number $c_j d_j$. It initially resides at position $Y(k-1)+j$, where by *resides* we mean indexing over all such matrix terms (i.e. treating the matrix terms as atomic), but not indexing over all the rows within all such matrix

terms. In equation 62, the term's position is $a_i b_i(j-1)+k$. Essentially, we are going from groups of size Y, which the index j ranges over, to groups of size $a_i b_i$, which the index k ranges over. To make the exchange, in the portion of τ we are building, we want to put a $c_j d_j \times c_j d_j$ identity matrix at *chunk* position $(a_i b_i(j-1)+k, Y(k-1)+j)$. If $(\tau'_i)_{1,1}$ is created as a matrix of *chunk* dimensions $a_i b_i Y \times a_i b_i Y$, then we have

$$((\tau'_i)_{1,1})_{a_i b_i(j-1)+k, Y(k-1)+j} = I_{c_j d_j \times c_j d_j} \tag{63}$$

for all j and k, and everything else is zero.[46] Then, for all the $\mathbf{a_i} \otimes \mathbf{b_i}$ terms, we direct sum together to solve the first part of our problem, going from equation 60 to 62.

$$(\tau')_{1,1} = \bigoplus_{i=1}^{B} (\tau'_i)_{1,1} \tag{64}$$

For step two, we have to go from equation 62 to equation 61. This entails reordering the middle two matrices in a tensor product. However, we already know how to do this using our swap 1-cell. For instance, to change $\mathbf{a_i} \otimes \mathbf{b_i} \otimes \mathbf{c_j} \otimes \mathbf{d_j}$ to $\mathbf{a_i} \otimes \mathbf{c_j} \otimes \mathbf{b_i} \otimes \mathbf{d_j}$, we would need $I_{a_i \times a_i} \otimes S_{b_i, c_j} \otimes I_{d_j \times d_j}$, where S_{b_i, c_j} is the swap 1-cell matrix. This must be done for each element in equation 62, which can all be direct summed together as

$$(\tau'')_{1,1} = \bigoplus_{i=1}^{B} \bigoplus_{j=1}^{Y} I_{a_i \times a_i} \otimes S_{b_i, c_j} \otimes I_{d_j \times d_j} \tag{65}$$

Finally, the final top left element-matrix of the interchange 2-cell $\tau_{g_2, g_1, i_2, i_1}$ is given as the matrix product of the two steps

$$(\tau_{g_2, g_1, i_2, i_1})_{1,1} = ((\tau'')_{1,1})((\tau'')_{1,1}) \tag{66}$$

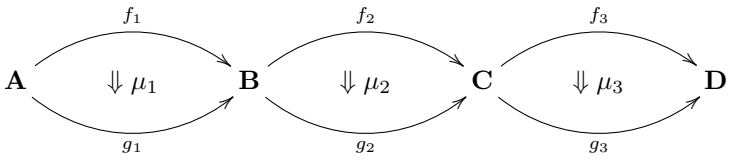
[46] Since all the *chunks* are of different sizes, the creation of $(\tau'_i)_{1,1}$ is rather nontrivial. One solution is to create a matrix of actual dimensions $a_i b_i Y \times a_i b_i Y$ and then at position $(a_i b_i(j-1)+k, Y(k-1)+j)$ store the size of the identity matrix to be created, $c_j d_j$. Then, we have a matrix of different values corresponding to the size of the identity matrix to be created there. Next, using a unary element operation with *Get Identity Matrix* as the generalized operation, identity matrices of correct size will be created in the correct positions, as long as zero matrices of appropriate size are also created to pad the rows and columns appropriately. For example, $\text{Get Identity Matrix}\left(\begin{bmatrix} 0 & 3 \\ 2 & 0 \end{bmatrix}\right) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$, so that an identity matrix is returned for each element of the input, with the zero padding occurring appropriately.

By applying this construction within a generalized tensor product as described at the top of the section, the entire structural isomorphism τ may be constructed. As expected, if any of the parameters are 0, everything still works as described using the rules for zero-dimensional matrices.

3.4.3 Associator Structure: ω

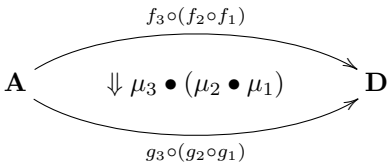
A structure is needed to mediate the rebracketing of horizontally composed 2-cell (the $\bullet$ operation).

Given the following definitions of 1-cells and 2-cells



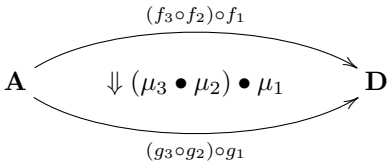
the 2-cells may be composed horizontally in two different orders to give

(67)



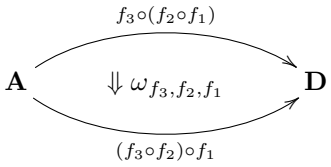
or

(68)



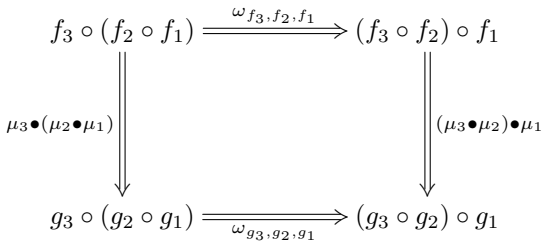
Just considering the 1-cells labeled by f for the moment, we have two different ways of composing them, $f_3 \circ (f_2 \circ f_1)$ or $(f_3 \circ f_2) \circ f_1$. In a 1-category, morphisms are associative by the axioms, and these would have to be of the same type. However, because of the freedom in our 2-category, they

can be isomorphically related by a 2-cell



(69)

Furthermore, for coherence, the following diagram must commute



(70)

where again the 0-cells have been omitted for clarity.[47] If ω equals the identity, then we would have $\mu_3 \bullet (\mu_2 \bullet \mu_1) = (\mu_3 \bullet \mu_2) \bullet \mu_1$, which is not true as we will explicitly show. In this case, part of the issue is that while $(A \oplus \ldots) \otimes B = (A \otimes B) \oplus \ldots$, $A \otimes (B \oplus \ldots) \neq (A \otimes B) \oplus \ldots$, for arbitrary matrices A and B. As for the other structural isomorphisms, $f_3 \circ (f_2 \circ f_1)$ and $(f_3 \circ f_2) \circ f_1$ will be the same functor and thus have the same matrix representation.

Explicit Construction The explicit construction of ω will be very similar to the construction of τ. As expected, ω transforms 2-cells as $A' = \omega A \omega^{-1}$,[48] and for the same reasons as before we only need to focus on reordering the rows of the 2-cell to be transformed. Similarly, if we solve the ordering problem for one element-matrix, we can use general operations to solve the entire 2-cell. We will use the same type of notation used in defining τ, so, if necessary, check back to Section 3.4.2.

As before, we will consider the general case. Consider the following definitions for $g_3 : \mathbf{A} \to \mathbf{B}$, $g_2 : \mathbf{B} \to \mathbf{C}$, and $g_1 : \mathbf{C} \to \mathbf{D}$, with only the first and last entries needed for constructing the top

[47] As with the other double arrow diagrams, if the 0-cells were not omitted, we would have a three-dimensional coherence diagram created by placing the diagram given by equation 67 on top of the diagram given by equation 68.

[48] i.e. $(\mu_3 \bullet \mu_2) \bullet \mu_1 = (\omega_{g_3,g_2,g_1}) \circ (\mu_3 \bullet (\mu_2 \bullet \mu_1)) \circ (\omega_{f_3,f_2,f_1})^{-1}$

left element-matrix of the 2-cell listed

$$g_3 = \begin{bmatrix} a_1 & \cdots & a_C \\ \vdots & \ddots & \vdots \end{bmatrix}$$

$$g_2 = \begin{bmatrix} b_{1,1} & \cdots & b_{1,B} \\ \vdots & \ddots & \vdots \\ b_{C,1} & \cdots & b_{C,B} \end{bmatrix}$$

$$g_1 = \begin{bmatrix} c_1 & \cdots \\ \vdots & \ddots \\ c_B & \cdots \end{bmatrix}$$

The elements of each of these are whole numbers representing the row dimension of the two cell to be transformed, $\mu_3 \bullet (\mu_2 \bullet \mu_1)$. Then, the two different ways of bracketing them gives two equivalent $D \times A$ dimensional 1-cells represented by

$$(g_3 \circ (g_2 \circ g_1)) = \tag{71}$$

$$\begin{bmatrix} a_1(b_{1,1}c_1 + \cdots + b_{1,B}c_B) + \cdots + a_C(b_{C,1}c_1 + \cdots + b_{C,B}c_B) & \cdots \\ \vdots & \ddots \end{bmatrix}$$

$$((g_3 \circ g_2) \circ g_1) = \tag{72}$$

$$\begin{bmatrix} (a_1b_{1,1} + \cdots + a_Cb_{C,1})c_1 + \cdots + (a_1b_{1,B} + \cdots + a_Cb_{C,B})c_B & \cdots \\ \vdots & \ddots \end{bmatrix}$$

where, again the order of the factors is the key to the reordering and has been preserved.[49] We

[49] N.B. At the two cell level, the second ordering will also be equivalent to $((g_3 \circ g_2) \circ g_1) = \begin{bmatrix} a_1b_{1,1}c_1 + \cdots + a_Cb_{C,1}c_1 + \cdots + a_1b_{1,B}c_B + \cdots + a_Cb_{C,B}c_B & \cdots \\ \vdots & \ddots \end{bmatrix}$. Thus, we will find we can distribute $\otimes$ over $\oplus$ from the right but not from the left.

will represent 2-cells by connecting them to their respective 1-cells, for instance, $\mu_2 : f_2 \Rightarrow g_2$, as

$$\mu_2 = \begin{bmatrix} \mathbf{b_{1,1}} & \cdots & \mathbf{b_{1,B}} \\ \vdots & \ddots & \vdots \\ \mathbf{b_{C,1}} & \cdots & \mathbf{b_{C,B}} \end{bmatrix}$$

where $\mathbf{b_{1,1}}$ means a $b_{1,1}$-by-unconcerned element-matrix. Again, these will be smallest unit of *row chunk* that needs to be reordered. The ellipsis indicate all the other terms if $B > 2$ or $C > 2$, where B and C refer to the 0-cell source and destination of g_2.[50] Using this representation, we find that the top left element-matrix of $(\mu_3 \bullet (\mu_2 \bullet \mu_1))$ will be ordered as

$$\mathbf{a_1} \otimes (\mathbf{b_{1,1}} \otimes \mathbf{c_1} \oplus \cdots \oplus \mathbf{b_{1,B}} \otimes \mathbf{c_B}) \oplus \cdots \oplus \mathbf{a_C} \otimes (\mathbf{b_{C,1}} \otimes \mathbf{c_1} \oplus \cdots \oplus \mathbf{b_{C,B}} \otimes \mathbf{c_B})$$

while the top left element-matrix of $((\mu_3 \bullet \mu_2) \bullet \mu_1)$ will be ordered as

$$(\mathbf{a_1} \otimes \mathbf{b_{1,1}} \oplus \cdots \oplus \mathbf{a_C} \otimes \mathbf{b_{C,1}}) \otimes \mathbf{c_1} \oplus \cdots \oplus (\mathbf{a_1} \otimes \mathbf{b_{1,B}} \oplus \cdots \oplus \mathbf{a_C} \otimes \mathbf{b_{C,B}}) \otimes \mathbf{c_B}$$

$$= (\mathbf{a_1} \otimes \mathbf{b_{1,1}} \otimes \mathbf{c_1}) \oplus \cdots \oplus (\mathbf{a_C} \otimes \mathbf{b_{C,1}} \otimes \mathbf{c_1}) \oplus \cdots \oplus (\mathbf{a_1} \otimes \mathbf{b_{1,B}} \otimes \mathbf{c_B}) \oplus \cdots \oplus (\mathbf{a_C} \otimes \mathbf{b_{C,B}} \otimes \mathbf{c_B})$$

Compressing everything to one column, and representing the *row chunks* as described, the two

[50] Of course, if $B = C = 1$, then μ_2 would only have one element-matrix, $\mathbf{b_{1,1}}$.

different orderings look like

$$(\mu_3 \bullet (\mu_2 \bullet \mu_1))_{1,1} = \begin{bmatrix} (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \end{bmatrix} \tag{73}$$

$$((\mu_3 \bullet \mu_2) \bullet \mu_1)_{1,1} = \begin{bmatrix} \mathbf{a_1} \otimes \mathbf{b_{1,1}} \otimes \mathbf{c_1} \\ \vdots \\ \mathbf{a_C} \otimes \mathbf{b_{C,1}} \otimes \mathbf{c_1} \\ \vdots \\ \mathbf{a_1} \otimes \mathbf{b_{1,B}} \otimes \mathbf{c_B} \\ \vdots \\ \mathbf{a_C} \otimes \mathbf{b_{C,B}} \otimes \mathbf{c_B} \end{bmatrix} \tag{74}$$

$$= \begin{bmatrix} (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \end{bmatrix}$$

where, as before, we use $(\mathbf{a_i})_{j,\cdot}$ to mean the element given by the jth row, $j = \{1, \ldots, a_i\}$, with the dot indicating no concern for the column, of the matrix represented by $\mathbf{a_i}$. Thus, $(\mathbf{a_i})_{j,\cdot} \times (\mathbf{b_{i,k}} \otimes \mathbf{c_k})$ means a number, $(\mathbf{a_i})_{j,\cdot}$, scalar multiplies the matrix, $(\mathbf{b_{i,k}} \otimes \mathbf{c_k})$. In the following, we also use $i = \{1, \ldots, C\}$ and $k = \{1, \ldots, B\}$ as indices. Equation 74 is written two ways to help decipher the notation.

From the numbers a_1, ..., a_i,..., a_C, $b_{1,1}$, ..., $b_{i,1}$,..., $b_{C,1}$, $b_{1,k}$, ..., $b_{i,k}$,..., $b_{C,k}$, $b_{1,B}$, ..., $b_{i,B}$,..., $b_{C,B}$, c_1, ..., c_k,..., and c_B, we want to find a matrix that reorders the column given by equation 73 to the column given by equation 74. This list of parameters can be generated by generalized matrix products. These parameters can then be fed to a generalized unary element operation that will compute all of ω. As with τ, this will be a two-step process. First we will

reorder to the form

$$\begin{bmatrix} (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_1})_{1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_1})_{a_1,\cdot} \times (\mathbf{b_{1,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,1}} \otimes \mathbf{c_1}) \\ \vdots \\ (\mathbf{a_C})_{1,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \\ \vdots \\ (\mathbf{a_C})_{a_C,\cdot} \times (\mathbf{b_{C,B}} \otimes \mathbf{c_B}) \end{bmatrix} = \begin{bmatrix} \mathbf{a_1} \otimes \mathbf{b_{1,1}} \otimes \mathbf{c_1} \\ \vdots \\ \mathbf{a_1} \otimes \mathbf{b_{1,B}} \otimes \mathbf{c_B} \\ \vdots \\ \mathbf{a_C} \otimes \mathbf{b_{C,1}} \otimes \mathbf{c_1} \\ \vdots \\ \mathbf{a_C} \otimes \mathbf{b_{C,B}} \otimes \mathbf{c_B} \end{bmatrix} \tag{75}$$

which involves reordering the rows in equation 73. Since all the terms involving matrix $\mathbf{a_1}$ come before all the terms involving matrix $\mathbf{a_i}$, which come before all the terms involving matrix $\mathbf{a_C}$, we can generally solve the problem by solving the part involving $\mathbf{a_i}$ and then direct summing the matrices together for all i.

Focusing on terms involving $\mathbf{a_i}$, let us consider an arbitrary element of equation 73, such as $(\mathbf{a_i})_{j,\cdot} \times (\mathbf{b_{i,k}} \otimes \mathbf{c_k})$. This is a *matrix* with row dimension given by the number $b_{i,k}c_k$. It originally resides at row position $B(j-1)+k$, where by *resides* we mean the same thing we meant when constructing τ—that when indexing the matrix terms are to be treated as atomic. In equation 75, it has moved to position $a_i(k-1)+j$. We are really just grouping the elements differently, going from groups of B, which the index k ranges over, to groups of a_i, which the index j ranges over. To make these exchanges, in the portion of ω we are building, we want to put a $b_{i,k}c_k \times b_{i,k}c_k$ identity matrix at *chunk* position $(a_i(k-1)+j, B(j-1)+k)$. If $(\omega_i')_{1,1}$ is created as a matrix of

chunk dimension $a_iB \times a_iB$, then we have

$$((\omega_i')_{1,1})_{a_i(k-1)+j,B(j-1)+k} = I_{b_{i,k}c_k \times b_{i,k}c_k} \tag{76}$$

for all j and k, and everything else is zero.[51] Then, we have

$$(\omega')_{1,1} = \bigoplus_{i=1}^{C} (\omega_i')_{1,1} \tag{77}$$

which successfully solves the first part of our problem, going from equation 73 to75.

Now, to go from75 to 74, we need to construct a second reordering.[52] For this step, we will be collapsing our chunks together, grouping $(\mathbf{a_i})_1, \ldots, (\mathbf{a_i})_j, \ldots, (\mathbf{a_i})_{a_i}$ back together into the *matrix chunk* $\mathbf{a_i}$. Thus, an arbitrary chunk is given by $\mathbf{a_i} \otimes \mathbf{b_{i,k}} \otimes \mathbf{c_k}$. After applying $(\omega')_{1,1}$, these terms are in groups of size B (with elements within the group indexed by k). But we want them in groups of size C (with elements in the group indexed by i). The initial position of $\mathbf{a_i} \otimes \mathbf{b_{i,k}} \otimes \mathbf{c_k}$ is $B(i-1)+k$, but we want to move it to $C(k-1)+i$. Thus, if $(\omega'')_{1,1}$ is created as a matrix of *chunk* dimension $CB \times CB$, we want to put an $a_ib_{i,k}c_k \times a_ib_{i,k}c_k$ identity matrix at *chunk* position $(C(k-1)+i, B(i-1)+k)$.[53]

$$((\omega'')_{1,1})_{C(k-1)+i,B(i-1)+k} = I_{a_ib_{i,k}c_k \times a_ib_{i,k}c_k} \tag{78}$$

for all i and k, and everything else is zero (q.v. footnote 46 as this is not a trivial operation). Finally, $(\omega_{g_3,g_2,g_1})_{1,1}$ is given by

$$(\omega_{g_3,g_2,g_1})_{1,1} = ((\omega'')_{1,1})((\omega')_{1,1}) \tag{79}$$

i.e. the matrix product of the two steps. As expected, if any of the parameters are 0, everything still works as described using the rules for zero-dimensional matrices.

Thus, the top left element-matrix of ω_{g_3,g_2,g_1} is solved. The other element-matrices are solved analogously and everything can be done in a tidy way using generalized operations. Essentially, a

[51] q.v. footnote 46 for ways to construct such a matrix.

[52] While constructing τ involved two steps as well, this second step is different from the second step involved with τ, q.v. Section 3.4.2.

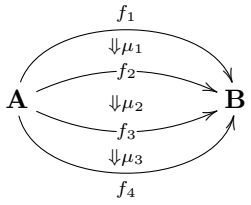
[53] These *chunk* sizes are different than the chunk sizes discussed going into equation 76.

generalized matrix product can be used to generate the parameters governing an element-matrix (i.e. $a_1, \ldots, a_i, \ldots, a_C, b_{1,1}, \ldots, b_{i,1}, \ldots, b_{C,1}, b_{1,k}, \ldots, b_{i,k}, \ldots, b_{C,k}, b_{1,B}, \ldots, b_{i,B}, \ldots, b_{C,B}\ c_1, \ldots, c_k, \ldots$, and c_B), and then for each element-matrix, a generalized unary element operation is called using the algorithm just described to generate $(\omega_{g_3,g_2,g_1})_{1,1}$.

Furthermore, it is worth mentioning that as a type of associator (for horizontal composition, $\bullet$), ω will satisfy some higher order pentagon equation. However, this pentagon equation would be above the 2-cell level and could not be explicitly described or solved for within **2Vect** in the manner that the monoidal associator can be described for *all objects* (q.v. Section 4.1.3). However, in this case, solving such an equation is unnecessary because we explicitly constructed a solution.

3.4.4 Other Associator Structures

In Section 3.4.3, we considered an associator for horizontal composition, the $\bullet$ operation. For completeness, it is worth considering if structural isomorphisms are necessary for other operations on 2-cells. First, the association of vertical composition, or $\circ$, is guaranteed by the axioms of 2-categories [26]. Given the following definitions of 1-cell and 2-cells,



we have that $(\mu_3 \circ \mu_2) \circ \mu_1 = \mu_3 \circ (\mu_2 \circ \mu_1)$. Since $\circ$ is given by a binary element operation with matrix products as the generalized operation, the fact that 2-cell vertical composition is associative is really just the statement that the matrix product is associative.

That leaves the 2-monoidal product, $\boxtimes$. That is, for $\mu_1 : f_1 \Rightarrow g_1$, $\mu_2 : f_2 \Rightarrow g_2$, and $\mu_3 : f_3 \Rightarrow g_3$, do we have $(\mu_3 \boxtimes \mu_2) \boxtimes \mu_1 = \mu_3 \boxtimes (\mu_2 \boxtimes \mu_1)$? This is exactly a question of whether the version of **2Vect** we are using is strict, i.e. whether the associator for its 2-monoidal structure is equal to the identity. Since $\boxtimes$ is a generalized tensor product with the tensor product as the generalized times, this operation is essentially a tensor product. Since the tensor product is associative, then $(\mu_3 \boxtimes \mu_2) \boxtimes \mu_1 = \mu_3 \boxtimes (\mu_2 \boxtimes \mu_1)$ will hold and the 2-associator is equal to the

identity.

4 Diagrammatic Language

The diagrammatic language (or graphical calculus) as formulated by Joyal and Street for working with symmetric monoidal categories [21] needs to be extended in two ways to work with symmetric monoidal 2-categories, such as **2Vect**.

For 1-categories, each picture represents a morphism or *process*[54]—time flows from bottom to top, objects are represented by lines, and morphisms are represented by boxes. Objects are monoidally combined by being placed next to each other. The identity morphism is just a line—i.e. nothing happens to the object. Two different pictures, two different morphisms, are either equal or unequal; the structure of 1-categories does not allow for any other relationship.

In 2-categories, pictures can be related in many ways—they can be equal, they can be isomorphic, or related by a non-invertible 2-cell. Thus, we can have arrows between pictures—arrows between morphisms—i.e. 2-cells. In order to find modular tensor categories within the structure of **2Vect**, we will write its axioms as an equation among 2-cells. By going to the 2-cell level, we ensure that we confirm the axioms for all objects in the category.[55] Thus, we will have two sets of pictures for each axiom, one for each path through the commutative diagram representing the axiom.

As explained in Section 3.4, some structural equations in symmetric monoidal 1-categories only hold up to isomorphism in our monoidal 2-categories. It was mentioned that this is due to the related 1-cells being of different type and was caused by the strictification of our representation. For instance, the structural isomorphism τ was required because the 1-cells $(g \circ f) \boxtimes (i \circ h)$ and $(g \boxtimes i) \circ (f \boxtimes h)$, while represented by the same matrix, are, in fact, different functors. However, in monoidal 1-categories, they are equal and thus represented by the same picture. Thus, for 2-categories, we need a way to pictorially capture the fact that they have different bracketings. The solution is to two-dimensionally bracket or *box* the diagram. Figure 1 is the diagram for the 2-cell $\tau_{g,f,i,h}$, which mediates between the 1-cells $(g \circ f) \boxtimes (i \circ h)$ and $(g \boxtimes i) \circ (f \boxtimes h)$. Here, one can

[54] This goes hand-in-hand with the focus of category theory being the morphisms rather than the objects.

[55] The semisimple property of **2Vect** ensures that if we just test the axioms for all the different combinations of the simple objects, the axioms will hold for all objects. This is exactly what testing the axioms at the 2-cell level does.

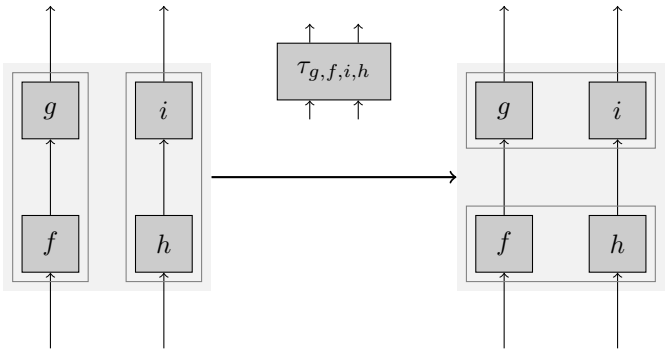


Figure 1: τ structural isomorphism

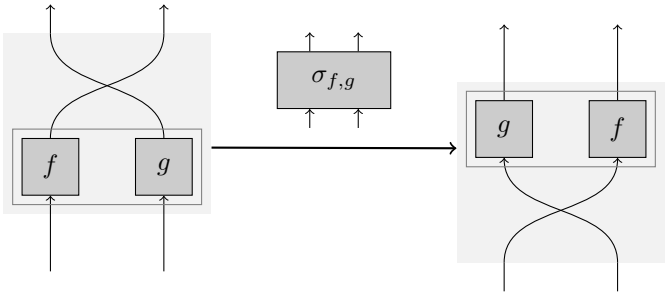


Figure 2: σ structural isomorphism

see that without the boxing, i.e. as it would be for 1-categories, both the source and destination 1-cells would be represented by the same picture.

Thus, we require diagrams for the other structural isomorphisms. Figure 2 is the diagram for the 2-cell $\sigma_{f,g}$, which moves the 1-cells f and g past a swap (and in the process swaps their order). Figure 3 is the diagram for the 2-cell $\omega_{h,g,f}$, which mediates between the 1-cells $h \circ (g \circ f)$ and $(h \circ g) \circ f$.

Furthermore, all diagrams of **2Vect** in our representation must be boxed. A 2-cell, defined diagrammatically, will have a specific source and destination diagram. A 2-cell can only be applied if the boxing—i.e. the type—matches. As we will see, when rewriting the axioms of modular tensor categories in the diagrammatic language, we will encounter cases where we will want to apply a 2-cell, such as the associator, but the boxing is wrong—a type mismatch. However, the structural isomorphisms can change the boxing—they can *rebox*. Thus, many structural isomorphisms will

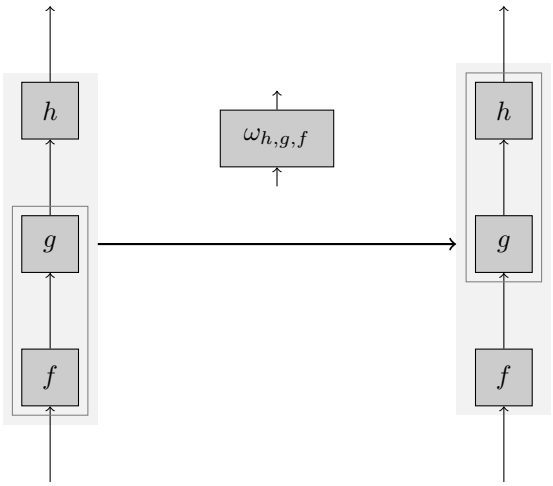


Figure 3: ω structural isomorphism

need to be inserted into the MTC axioms in order to represent them within a symmetric monoidal 2-category, like **2Vect**. These three structural isomorphisms, τ, σ, and ω, and their inverses (just reverse the source and destination 1-cells), provide enough structure to overcome our strictification and allow us to represent all the MTC axioms in the diagrammatic language—which will then let us represent the axioms in **2Vect** in our computer algebra system.[56]

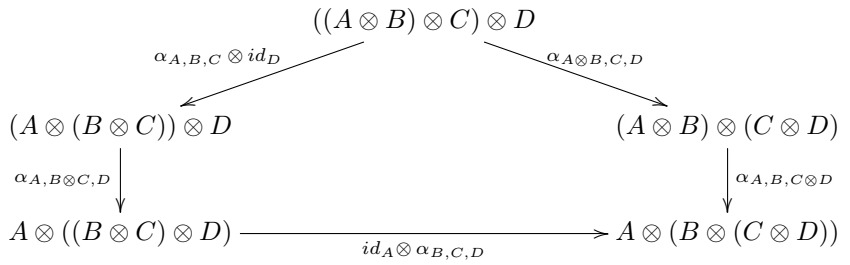
The testing of the MTC structures proceeds very straightforwardly and will become clear in the following example for the pentagon equation. The testing of the MTC properties rigidity and modularity requires slightly more than rote translation of the axioms into **2Vect** diagrams. Finally, since **2Vect** and **Vect** are automatically semi-simple, we do not have to test this property.

4.1 Monoidal Categories: Pentagon Equation

As an example, we will show how to convert the pentagon equation for monoidal categories—equation 1, reproduced below for convenience—into a diagrammatic form. This is a necessary step in order to get a handle on all the implied structural isomorphisms made necessary by our representation. Once in this form, the axiom can be understood by our computer algebra representation

[56] Technically, σ is a rewriting rule, not a reboxing rule.

of **2Vect** as described in Section 3.



(80)

First, for all the new structures, we need to establish the source and destination type—i.e. the boxing of the source and destination 1-cells—so we won't have a type mismatch when applying them. In this case, the new structure is α, and we will establish its type by boxing the source and destination as shown in Figure 4. Remember, from the perspective of 1-categories, α is a natural transformation with components $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$. However, in **2Vect**, natural transformations are 2-cells, and the tensor product functor for the MTC inside **2Vect** (which was just shown as $\otimes$) is a 1-cell of **2Vect** that we call m. Thus, we represent $(A \otimes B) \otimes C$ as a diagram in **2Vect** with stacked m's, and we represent $A \otimes (B \otimes C)$ as a diagram in **2Vect** with stacked m's in a different way, to show their different bracketing. Additionally, we must *box* these source and destination diagrams. Since we are *defining* α, we can pick any boxing—i.e. since it's a definition we get to define its type. Boxing is done by picking two elements of the diagram at a time (that are either next to each other or follow each other) and putting a box around them.[57] Once the type is set, it must be used consistently through the same equation. Thus, in order to apply a 2-cell, such as α, it may be necessary to rearrange the boxing into the appropriate form to get the correct type. This can only be done by the structural isomorphisms, and this process of diagramming the equation will force us to insert all the necessary structures.

Once we have diagrams for all our new structures, we start by drawing the 1-cell start state, $((A \otimes B) \otimes C) \otimes D$, as a diagram in **2Vect**. This is done using the same method above to draw $A \otimes (B \otimes C)$, including the boxing. Again, the start state may be arbitrarily boxed (as long as the rules for boxing are followed) from the beginning as long as it is kept consistent. However, let us box in the way shown in Figure 5.

[57] N.B. Lines are implicitly the identity 1-cell. Thus, an arbitrary number of boxes may be drawn and removed from lines, since they may represent one or more identities.

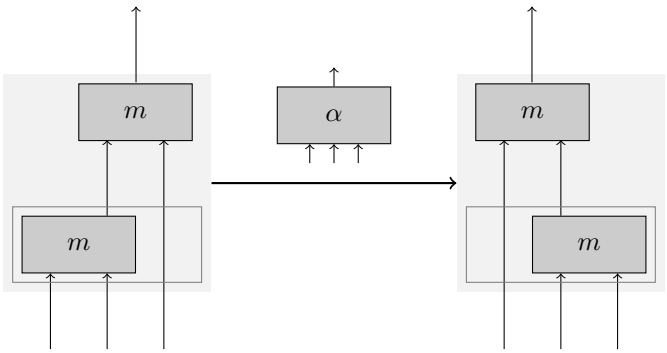


Figure 4: α

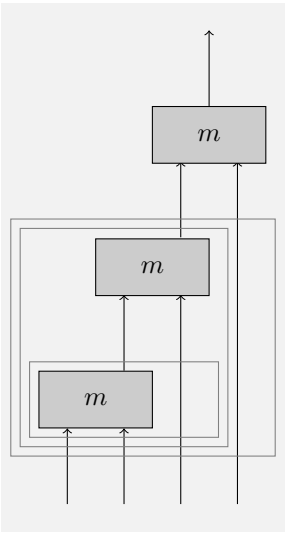


Figure 5: The start state to the pentagon equation, $((A \otimes B) \otimes C) \otimes D$.

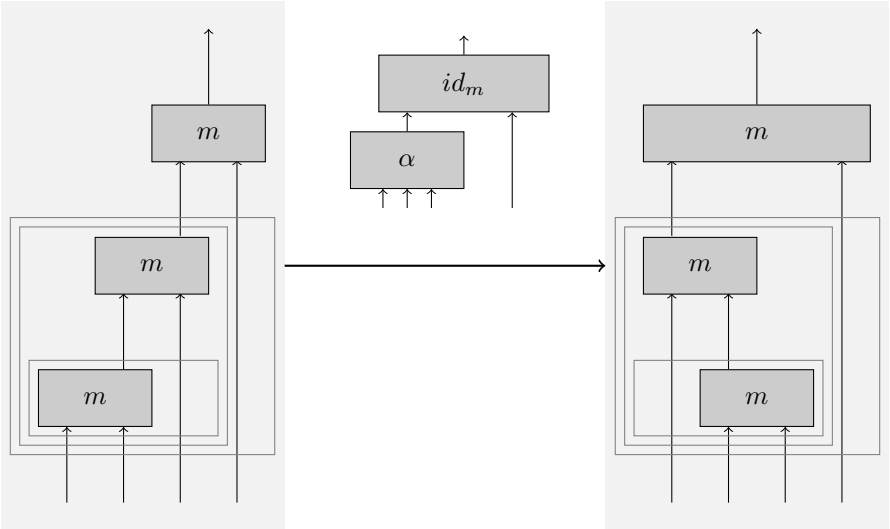


Figure 6: pentagon equation, path 1, step 1: $\alpha_{A,B,C} \otimes id_D$

Now, since both paths through the pentagon must be equal, we will have a string of diagrams for each path.

4.1.1 Path 1

We we start with the longer three-legged path. Thanks to our fortuitous initial boxing, the diagram is in the right form for $\alpha_{A,B,C} \otimes id_D$ (as it is written from the MTC perspective) to be applied—we can see embedded in the lower left of the diagram the correct type for the source of α as we boxed it in Figure 4. Thus, we apply α to that part and apply the 2-cell identity to the m at the top. The result is shown in Figure 6. In the destination 1-cell, we see that α was applied in the lower left, and the rest of the diagram remained unchanged. The 2-cell we applied, $\alpha_{A,B,C} \otimes id_D$ from the view of the MTC, is shown drawn above the arrow. From the **2Vect** perspective, this is $id_m \circ (\alpha \boxtimes id_{id_\mathbf{N}})$, where $\mathbf{N}$ is the base 0-cell for this MTC. In this, we correspond the $\alpha_{A,B,C}$ with the α, the id_D with the $id_{id_\mathbf{N}}$ (that is, the identity 2-cell taken from the identity 1-cell on the base 0-cell of the category), and the $\otimes$ with the id_m. Thus, we can begin to see that by writing this equation within the semisimple category **2Vect**, we will be able to test the equation for *all* simple objects (and thus all objects) of the MTC. We are no longer concerned with arbitrary objects A, B, C, and D, but instead will test for every combination by operating at the 2-cell level of **2Vect**.

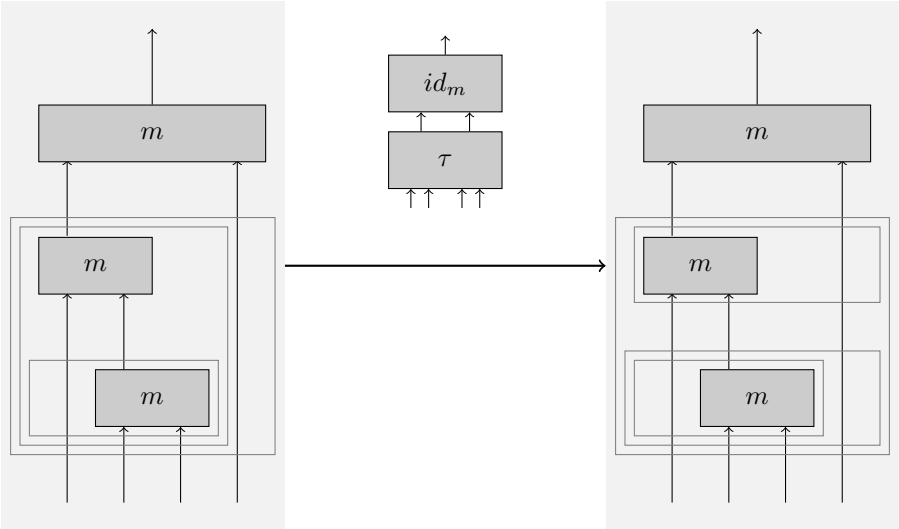


Figure 7: pentagon equation, path 1, step 2: τ reboxing

Now, we want to apply (from the MTC perspective) $\alpha_{A,B\otimes C,D}$—however, the type is not right. In fact, we need to make two reboxing moves. First, as we see the source types match, we need to apply τ. This is shown in 7. In order to apply τ, we implicitly used a box in the right-most line of the source 1-cell of the diagram to conform with the input source type for τ. This can be added since there is an implied boxing of the many identities that the line consists of (q.v. footnote 57).[58]

Now, we will apply ω to the outer two boxes, which will finally align the top part of the diagram so that the next step in the actual pentagon equation may be applied. This is shown in Figure 8, where we have used an implicit box around the top m in the source 1-cell—any single element can be considered boxed—to make the input consistent with the source 1-cell of ω.

Now, we will apply the next actual step in the pentagon equation $\alpha_{A,B\otimes C,D}$. We see that the top of the 1-cell destination diagram from our last step is in the correct form for the application of α, and this corresponds to exactly the next step in the pentagon equation. The application of this step is shown in Figure 9.

Now, again, before we can apply the final leg of this path through the pentagon, we need to rebox—this time to effectively undo the reboxing from before. First we apply a ω^{-1}, Figure 10,

[58] For the pentagon equation, all of the τ's will take at least two identities as arguments—rendering the τ equal to the identity 2-cell—and thus could be omitted. However, they have all been included in all the equations we will present, for completeness, and so the reader can gain an understanding of how they ought to be applied.

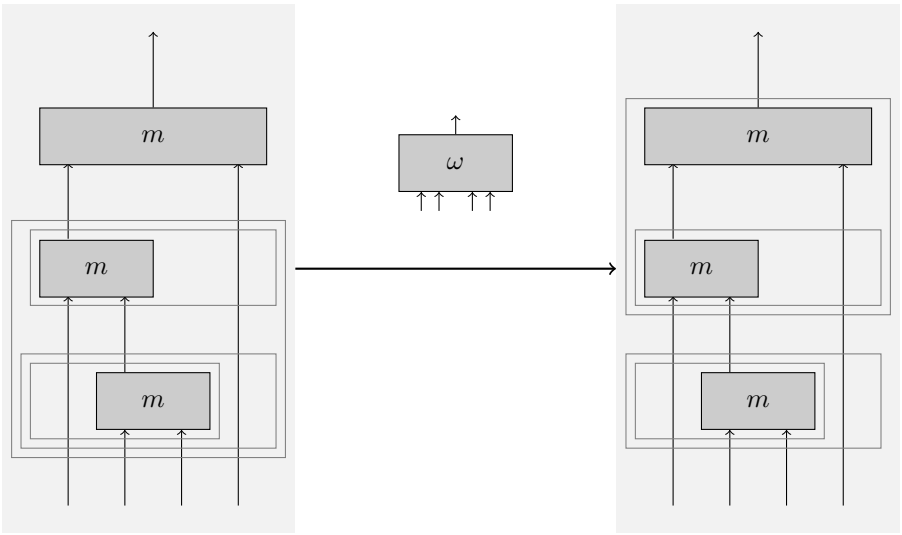


Figure 8: path 1, step 3: ω reboxing.

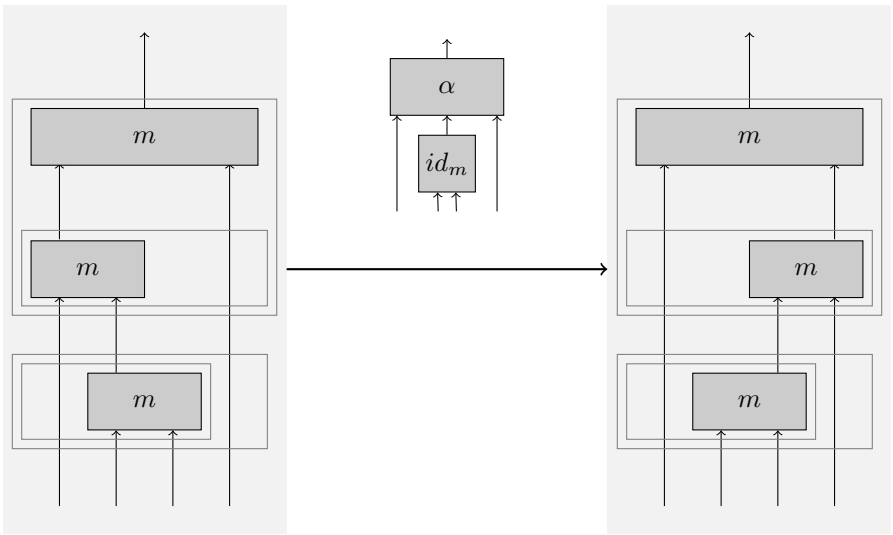


Figure 9: pentagon equation, path 1, step 3: $\alpha_{A,B\otimes C,D}$

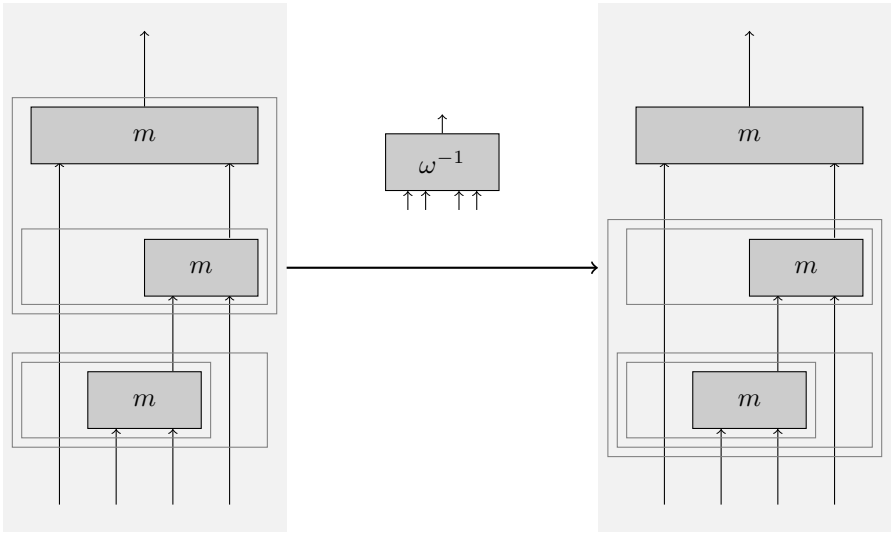


Figure 10: pentagon equation, path 1, step 4: ω^{-1} reboxing

and then we apply a τ^{-1}, Figure 11. However, notice the difference in the bottom row boxing from the destination 1-cell from Figure 10 and the source 1-cell from Figure 11, which, ostensible, should be the same picture. However, using the freedom given to us because the associator for the 2-monoidal structure of **2Vect**, $\boxtimes$, is the identity (q.v. Section 3.4.4), we can associate parallel levels without applying any structural isomorphism 2-cell. Also, for clarity, we have omitted the box around the left-most line in the destination 1-cell of Figure 11.[59]

Finally, we are in exactly the correct form to apply the last leg of the pentagon, which is (from the MTC perspective) $id_A \otimes \alpha_{B,C,D}$. This is shown in Figure 12. The destination 1-cell of this Figure is required result; the diagram represents a particular boxing (although many are possible) of $A \otimes (B \otimes (C \otimes D))$.

4.1.2 Path 2

The short path through the pentagon proceeds in an identical fashion, with two caveats.

First, since we started with an arbitrary boxing of the first 1-cell state, $((A \otimes B) \otimes C) \otimes D$—which happened to be convenient for application of the first leg of the first path—we cannot simply apply the first leg of the new path. Almost always, some reboxing will be necessary. In this case,

[59] If this was not true (for instance, if we used a different 2-monoidal structure on **2Vect**), then this reboxing would come at the cost of applying another structural isomorphism 2-cell).

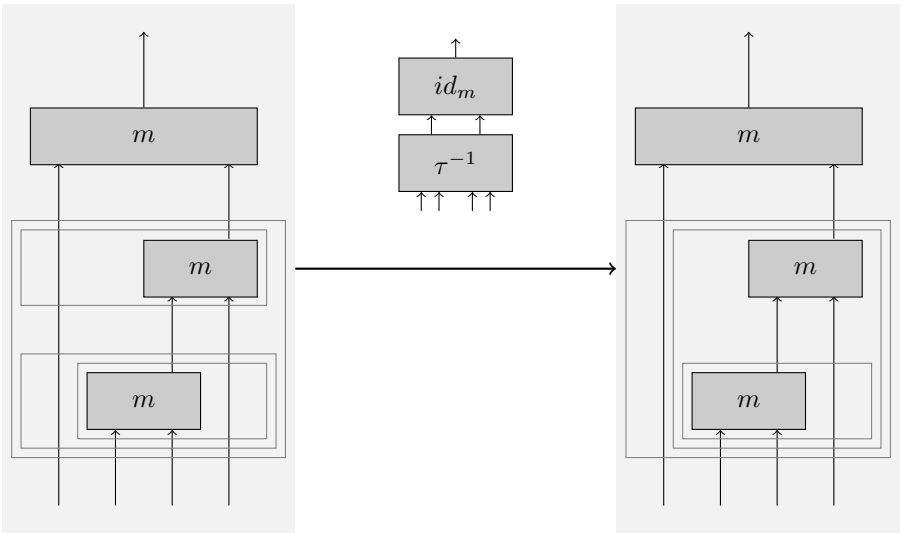


Figure 11: pentagon equation, path 1, step 5: τ^{-1} reboxing

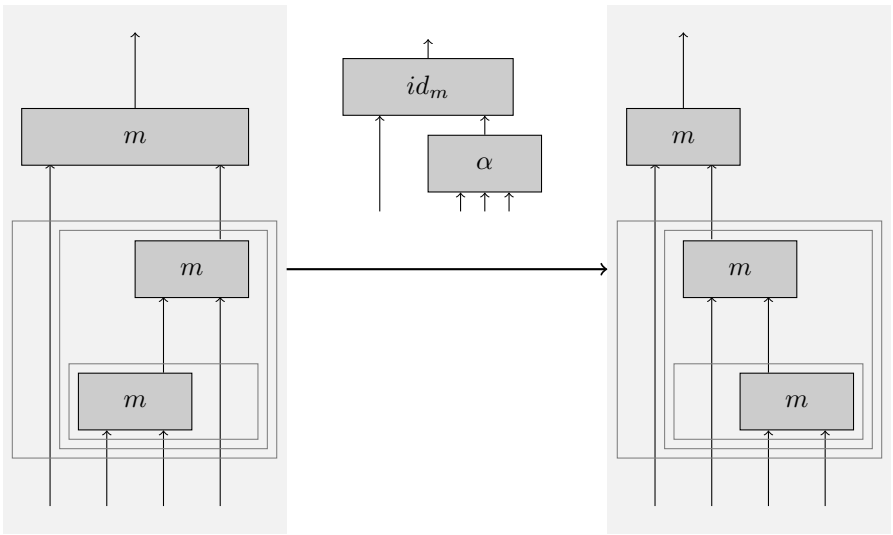


Figure 12: pentagon equation, path 1, step 6: $id_A \otimes \alpha_{B,C,D}$

to apply $\alpha_{A\otimes B,C,D}$, we first need to apply τ, followed by ω.

Second, similar to the first caveat, just because the end state, $A \otimes (B \otimes (C \otimes D))$, is reached, does not mean that the process is complete. For the first path, we ended with an arbitrary boxing of that state. For the second path, most likely we will end with a different boxing. Thus, reboxing steps will need to be taken to ensure that the final diagrams for both paths match—i.e. so they are the same type and fit for comparison. Thus, the construction of the second path is not any different from the construction of the first path, except that it requires both reboxing at the beginning and at the end, which is necessitated by the boxing we chose to start and end with in the first path.

4.1.3 Comparison of Paths

Now, we can use the diagrams of both paths to implement the pentagon equation in our computer algebra system so that it may be used to check whether an associator and product functor satisfy it, or so that given a product functor, associators can be solved for. While the 1-cells play the most important role in drawing the diagrams (and it's there boxing that determines the need for structural isomorphisms), we are ultimately interested in the 2-cells. It may not look like it, but each path really specifies a 2-cell. The statement that both paths through the pentagon equation are equal is the statement that the 2-cell represented by both paths are equal.

The full diagrams were drawn to determine where the structural isomorphisms were needed. Now, we focus on the 2-cells, the diagrams above the arrows in each path. These *arrows* are the 2-cells, and the vertical composition of the entire path represents the 2-cell of that path.[60] Each diagram that sits on an arrow represents a 2-cell that can be represented in our computer algebra system. Each 2-cell is built up and then composed, so that the entire path is constructed. Then, the 2-cells from each path are equated. This can either return true or false (if the relevant structures were entered completely with the purpose of being tested) or the equations generated can be used to solve for a structure (i.e. in this case, the associator).

Finally, an alternate interpretation of the structural isomorphisms is that they exist to ensure both paths operate in consistent bases. For instance, a different boxing of the final state, $A\otimes(B\otimes(C\otimes D))$, for the second path means that the 2-cell's element-matrices resulting from that path

[60] N.B. Horizontal composition is represented as bottom to top flow within the 2-cell diagram that sits above the arrow.

with be expressed in a different basis. In fact, given that the 2-cells are matrices of matrices and the element-matrices of the structural isomorphisms are unitary, it is clear that ω and τ are applied exactly as would be expected for changing the basis of a 2-cell. Heuristically, first we applied τ, then we applied ω, then we applied the 2-matrix whose basis we are changing, $\alpha_{A,B\otimes C,D}$, then we applied ω^{-1}, and finally we applied τ^{-1}. Schematically, (using the right to left order of 2-cell vertical composition), this looks like $\tau^{-1} \circ \omega^{-1}\alpha \circ \omega \circ \tau$, which we are allowed to associate as $\tau^{-1} \circ (\omega^{-1}\alpha \circ \omega) \circ \tau$.

As an example of this, let us consider the pentagon equation for **Fib**. We will focus on the matrix $\alpha_{\tau,\tau,\tau}$, which is the two-dimensional change of basis matrix for the fusion of three τ anyons.[61] Let us assume it is unknown and write it as

$$\alpha_{\tau,\tau,\tau} = \begin{bmatrix} w & x \\ y & z \end{bmatrix} \tag{81}$$

Let us also assume the rest of the 2-cell α is trivial (as it is). Now, if we were to compute both paths of the pentagon equation, we will get a 2-cell for each path, P and Q. Let P represent path 1 (the path with three legs), and Q represent path 2 (the path with two legs). The pentagon equation is the statement that $P = Q$. For each of these 2-cells, there is a three-dimensional element-matrix for the fusion of four τ anyons, $P_{\tau,\tau,\tau,\tau}$ and $Q_{\tau,\tau,\tau,\tau}$. If we naïvely did not use τ and ω, we would find

$$P_{\tau,\tau,\tau,\tau} = \begin{bmatrix} 1 & 0 \\ 0 & (\alpha_{\tau,\tau,\tau})^3 \end{bmatrix} \tag{82}$$

$$Q_{\tau,\tau,\tau,\tau} = \begin{bmatrix} 1 & 0 \\ 0 & (\alpha_{\tau,\tau,\tau})^2 \end{bmatrix} \tag{83}$$

i.e. the equation would effectively be $\alpha^2 = \alpha^3$, leading to a trivial associator. Every application of

[61] N.B. α mediates changes of basis in the fusion space. However, the structural isomorphisms change the basis of 2-cells themselves, such as the basis of α.

a leg of the pentagon equation would just multiply the matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & w & x \\ 0 & y & z \end{bmatrix} \tag{84}$$

which is clearly incorrect. However, the structural isomorphisms will change that basis of that matrix—mixing around w, x, y, and z—leading to the correct expressions for $P_{\tau,\tau,\tau,\tau}$ and $Q_{\tau,\tau,\tau,\tau}$:

$$P_{\tau,\tau,\tau,\tau} = \begin{bmatrix} w & xy & xz \\ xy & w^2 + xyz & wx + xz^2 \\ yz & wy + yz^2 & xy + z^3 \end{bmatrix} \tag{85}$$

$$Q_{\tau,\tau,\tau,\tau} = \begin{bmatrix} xy & w & xz \\ w & 0 & x \\ yz & y & z^2 \end{bmatrix} \tag{86}$$

The solution of $P = Q$ will now provide the correct α for **Fib**. Thus, we see that type safety and boxing are the same as ensuring consistent bases are used.

4.2 Monoidal Categories: Triangle Equation

As we have seen, essentially, the game is to use the reboxing tools to get the diagram into the appropriate form to apply the structure required in the axiom. As such, we will no longer describe the step by step process, but simply show definitions of the new structures and list any difficulties or peculiarities specific to the equation in question.

The definitions for the new structures, λ and ρ, are shown in Figures 13 and 14, respectively. The formulation of the diagrams for the triangle equation, equation 2, proceeds exactly as it did in the previous section.

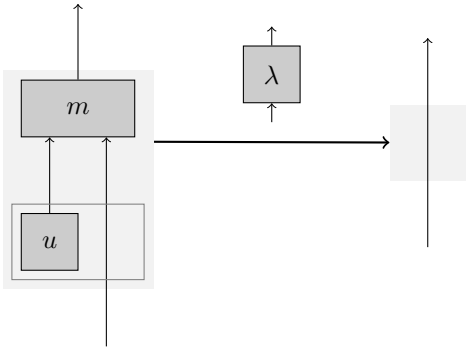


Figure 13: λ

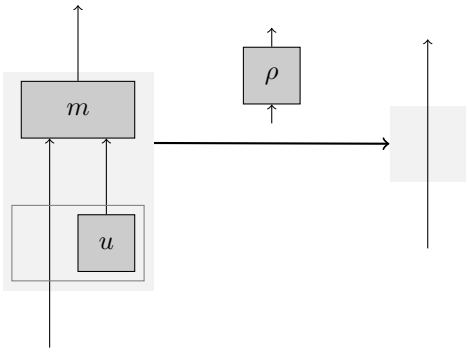


Figure 14: ρ

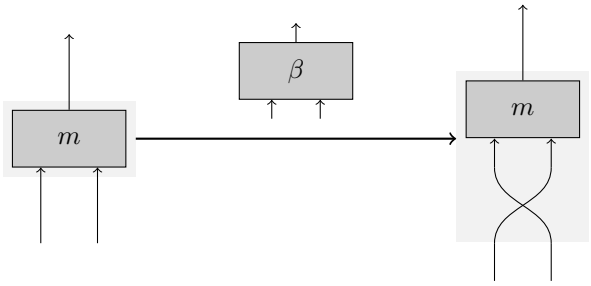


Figure 15: β

4.3 Braided Monoidal Categories

For both hexagon equations, equations 3 and 4, there is one additional structure, the braid β, whose definition is shown in Figure 15.

Additionally, these are the first equations to require the use of the swap 1-cell (constructed in Section 3.3.2), which is drawn as wires crossing. Necessary to the construction of these paths (and previously unmentioned) is the fact that the swap satisfies the following properties:

$$(S_{\mathbf{A},\mathbf{C}} \boxtimes id_{\mathbf{B}}) \circ (id_{\mathbf{A}} \boxtimes S_{\mathbf{B},\mathbf{C}}) = S_{\mathbf{A}\boxtimes\mathbf{B},\mathbf{C}} \tag{87}$$

$$(id_{\mathbf{B}} \boxtimes S_{\mathbf{A},\mathbf{C}}) \circ (S_{\mathbf{A},\mathbf{B}} \boxtimes id_{\mathbf{C}}) = S_{\mathbf{A},\mathbf{B}\boxtimes\mathbf{C}} \tag{88}$$

In our representation, we find that these 2-cells are just equal and do not require any structural isomorphism. The respective 1-cell diagrams for these equations are shown in Figures 16 and 17. With the addition of these swap properties, the formulation of the hexagon diagrams proceeds exactly as they did in the previous sections.

4.4 Twisted Braided Monoidal Categories

For the twist axiom, equation 6, the newly defined structure is the twist, θ, shown in Figure 18. Remember, the twist goes from the identity 1-cell to the identity 1-cell. In the particle

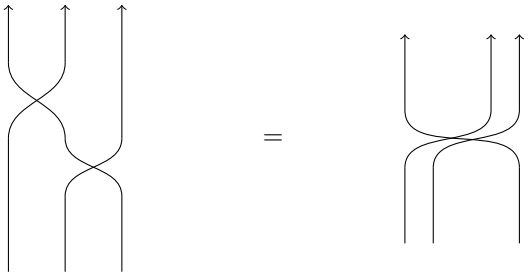


Figure 16: swap property: $(S_{\mathbf{A},\mathbf{C}} \boxtimes id_{\mathbf{B}}) \circ (id_{\mathbf{A}} \boxtimes S_{\mathbf{B},\mathbf{C}}) = S_{\mathbf{A}\boxtimes\mathbf{B},\mathbf{C}}$

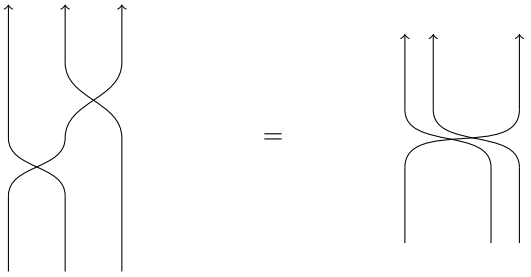


Figure 17: swap property: $(id_{\mathbf{B}} \boxtimes S_{\mathbf{A},\mathbf{C}}) \circ (S_{\mathbf{A},\mathbf{B}} \boxtimes id_{\mathbf{C}}) = S_{\mathbf{A},\mathbf{B}\boxtimes\mathbf{C}}$

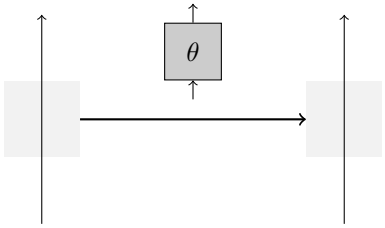


Figure 18: θ

interpretation, θ causes a 2π rotation—thus, if our lines had width, we would show its action as the twisting of a ribbon. Additionally, a property of the swap 1-cell that was described in Section 3.3.2 is required for making these diagrams: $S_{\mathbf{M},\mathbf{N}} \circ S_{\mathbf{N},\mathbf{M}} = id_{\mathbf{N}\boxtimes\mathbf{M}}$. Otherwise, the formulation of these diagrams is very straightforward.

4.5 Monoidal Functors

For the monoidal functor equations, equations 11–13, there are three new structures: F, ϕ, and ϕ_u. The first one, F, is the monoidal functor. A functor is a 1-cell of **2Vect**, thus, this will be represented as a 1-cell (a box) with the symbol F. The second one, ϕ, is a natural transformation and will be represented as a 2-cell (and is shown in Figure 19). Finally, ϕ_u, is a morphism of the destination category. Morphisms internal to a category within **2Vect**—i.e. morphisms internal to an MTC—can be accessed from the 2-category perspective via 2-cells. Thus, ϕ_u will be represented as a 2-cell and is shown in Figure 20. Since we are dealing with two categories, the monoidal structures from the source category are labeled with the subscript 1, and the monoidal structures from the destination category are labeled with the subscript 2.

4.6 Rigidity

In Section 2.1.4, we provided the conditions for a category to be rigid (equations 7 and 8)—essentially that every object must have a dual. However, we did not specify how to construct the duals, which is necessary for actually checking the rigidity conditions.

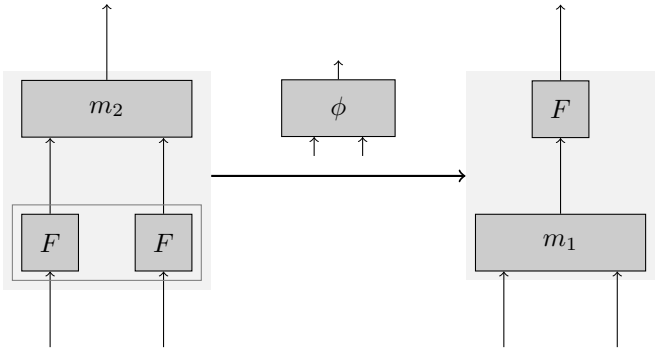


Figure 19: ϕ

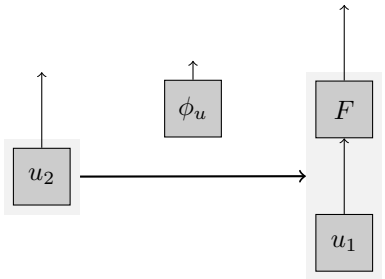


Figure 20: ϕ_u

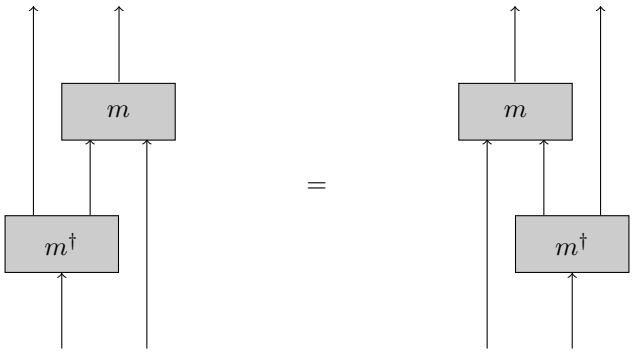


Figure 21: Frobenius condition

4.6.1 The Frobenius Condition and Constructing Duals

Luckily, there is an easy way to find a dual "candidate" for a given object [8]. If tensor product functor for the monoidal category satisfies the *Frobenius condition*, then there is a simple formula to find the dual candidate given an object.

The Frobenius condition is shown in Figure 21, where $m^\dagger$ is taken to mean the adjoint of the 1-cell m (which, of course, represents the fusion rules) and is given by the matrix adjoint of the matrix representation of m. Categorically, we are guaranteed that this adjoint exists by the structure of **2Vect** [6]. The Frobenius condition—which is a relationship only among 1-cells and does not involve boxing—is a precursor for a category to be rigid.

If a category satisfies the Frobenius condition, then there is an easy way to construct the dual candidate for a given object.[62] The formula, for the dual A^* to the object A, is shown in Figure 22. Heuristically, using the particle interpretation of rigidity, a particle antiparticle pair is created from the vacuum—where the unit, as always, is effectively the vacuum, and $m^\dagger$ is a "reverse fusion" that creates the pair. Then, $A^\dagger$, the adjoint of the morphism representing the object A, is in a sense "contracted" with original particle, thus projecting onto the space of the antiparticle, the dual A^*.

[62] We say dual candidate because equations 7 and 8 still need to be satisfied for the category to be rigid.

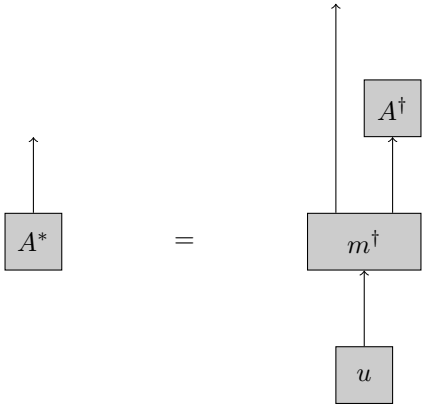


Figure 22: dual candidate A^*

4.6.2 Testing Rigidity

Since our category is semisimple, we only have to worry about testing the simple objects. However, testing the rigidity property is slightly more complicated than just creating diagrams of the equations. For each pair of objects A and A^*, the maps $\eta_A : u \to A \otimes A^*$ and $\varepsilon_A : A^* \otimes A \to u$ also need to be constructed. Luckily, there is only a one-dimensional space of such maps, and they can just be picked to within a scalar constant. Thus, the nontrivial path through equation 7 (or equation 8) may be off by a constant, which can be rescaled in the definitions of η_A and ε_A. The only way these equations can fail to hold is if the nontrivial path is zero.

Thus, rigidity—at its essence—is a question of whether certain contractions of the 2-cell associator α and its inverse α^{-1} are zero. Let's focus on equation 7, and thus α, as an example. For each simple object, first we generate its dual as per the prescription in Figure 22. Then, we look at the element-matrix $\alpha_{A,A^*,A}$. The dimensions of this square matrix are determined by the dimension d of the fusion space of $A \times A^*$ (with the very likely possibility that $d < n$, where n is the number of simple objects). Since η_A is a map from u to $A \otimes A^*$ and ε_A is a map from $A^* \otimes A$ to u, we want to contract $\alpha_{A,A^*,A}$ with the basis vectors for fusing to and fusing from u—i.e. make the quadratic form $\overrightarrow{x}^\dagger \alpha_{A,A^*,A} \overrightarrow{x}$, where $\overrightarrow{x}$ is the basis vector for fusing to u in the d-dimensional fusion subspace. If this number is not equal to zero for all objects A, then the condition set by equation 7 is satisfied. The condition set by equation 8 is similar except it concerns α^{-1} instead

of α.

As a specific example, consider a category with three simple objects: u, σ, and ψ. The nontrivial fusion rules are $\sigma \times \sigma = u + \psi$, $\sigma \times \psi = \psi \times \sigma = \sigma$, and $\psi \times \psi = u$ [33]. The MTC resulting from these fusion rules is commonly called the *Ising MTC*. Now, from this it is clear that σ is self-dual. Furthermore, let the morphism that represents the unit, $u : \mathbf{1} \to \mathbf{3}$, and the nontrivial particles, $\sigma : \mathbf{1} \to \mathbf{3}$ and $\psi : \mathbf{1} \to \mathbf{3}$, be given by[63]

$$u = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{89}$$

$$\sigma = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{90}$$

$$\psi = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{91}$$

To test the duality condition equation 7 for σ, we need the matrix $\alpha_{\sigma,\sigma,\sigma}$. For this category, it is given by

$$\alpha_{\sigma,\sigma,\sigma} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{92}$$

which is two-dimensional, since $\sigma \times \sigma$ can fuse to either u or ψ. However, the basis given by equations 89–91 is three-dimensional, corresponding to the total number of simple objects in the category. Since $\sigma \times \sigma$ cannot fuse to ψ, we want to take the basis vector given by equations 89 for u and remove the space corresponding to ψ—in this case the first zero—to give

$$u_\sigma = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{93}$$

[63] q.v. equations 18 and 19.

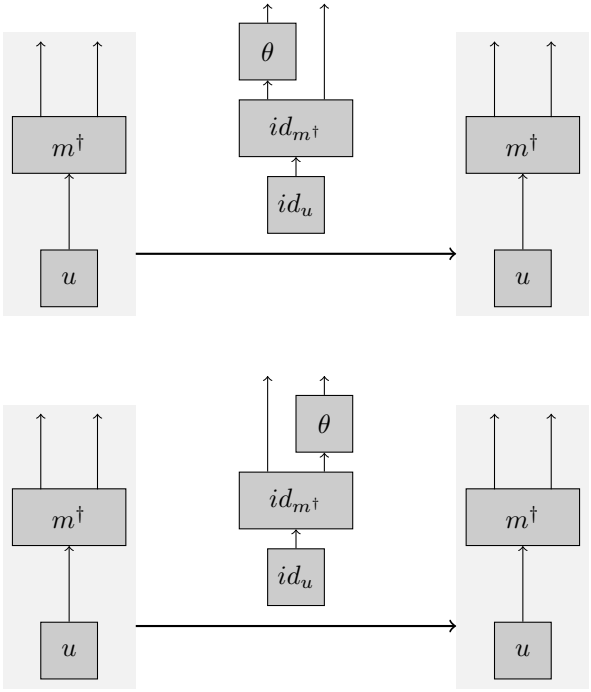


Figure 23: The braid and twist are compatible with rigidity if these 2-cells are equal.

If there were other simple objects that the object and its dual could not fuse to, we would have to remove them as well. Thus, it is this basis vector, u_σ, that we want to contract $\alpha_{\sigma,\sigma,\sigma}$ with, giving $u_\sigma^\dagger \alpha_{\sigma,\sigma,\sigma} u_\sigma = \frac{1}{\sqrt{2}}$, which is not equal to zero.

4.6.3 Testing Braid and Twist Coherence with Rigidity

We also need to test equation 9 to ensure that our twisted braided monoidal category is compatible with our rigid structure. However, rather than test each simple object and its dual individually, we can apply an elegant 2-cell method to test them all at once. If we start with our unit, u, and then compose with $m^\dagger$, we have a map, $m^\dagger \circ u : \mathbf{1} \to \mathbf{N} \boxtimes \mathbf{N}$, where N is the number of simple objects. This map will give all the pairs of simple objects and their duals. Thus, we can apply our twist, θ, to the left leg for a 2-cell representing one path and the right leg for a 2-cell representing the other path. These 2-cells are shown in Figure 23. Equating the two 2-cells will give equation 9.

4.7 Modularity

The diagrammatic form of the modularity equation, equation 10, proceeds exactly as the others. A category is modular when—for all simple objects that are not the unit—there exists another simple object such that the equation does not hold.

To test this property in our computer algebra system, we compute the 2-cell $(\beta \bullet id_{S_{\mathbf{N},\mathbf{N}}}) \circ \beta$, where $\mathbf{N}$ is the base 0-cell of the category. Now, each column in our 2-cell representation represents the fusion of two simple objects, and the row represents the different fusion paths. For each simple object, A that is not the unit (there are $N-1$ of them), we look at all the columns that represent the fusion $A \times B$, where B is any other simple object that is not the identity. If for each A, at least one row in one of the $A \times B$ fusion columns is not the identity—i.e. every nontrivial simple object braids nontrivially with at least one other nontrivial simple object (including, possibly itself)—then the category is modular.

5 Conclusion and Further Research

Having developed this computer algebra system for **2Vect** and modular tensor categories, the following directions are possible for further research:

1. Implement the ability to solve for the structures of the MTCs (as opposed to just verifying they satisfy the correct equations).

2. Determine *high-level* structures necessary for TQC using MTCs.

Both of these represent different research directions and can be implemented in either order or even in parallel.

5.1 Solver

The solver will allow users to propose fusion rules and have then have the system find structures that satisfy the categorical axioms with those rules. Thus, the program will be able to "solve" for these types of categories. Since it is interesting in its own right to find and classify these categories (specifically MTCs) [33], this part would be an end in itself in addition to the TQC application.

Currently, we have implemented a basic solver, but is not efficient. It can solve for the associator of certain small[64] monoidal categories. With certain trivial bits filled in, it can solve for slightly more complicated categories. Additionally, it is able to solve for the equivalence functor of the monoidal category **Fib** (i.e. **Fib** while forgetting about its braiding and twisting structures), showing that there are two inequivalent families of monoidal categories, **Fib**. One possible direction for this project is the optimization of the solver.

As an example, when given the fusion rules for **Fib** (described in Section 2.3.3 by equation 17), the unit, and the trivial elements of the associator, our code in Mathematica can solve the pentagon equation for $\alpha_\tau^{\tau\tau\tau}$.[65] Furthermore, it gives two one-parameter families of solutions

$$(\alpha_\tau^{\tau\tau\tau})_+ = \begin{bmatrix} \varphi^{-1} & a \\ (a\varphi)^{-1} & -\varphi^{-1} \end{bmatrix} \tag{94}$$

$$(\alpha_\tau^{\tau\tau\tau})_- = \begin{bmatrix} -\varphi & b \\ -\varphi b^{-1} & \varphi \end{bmatrix}$$

where a and b are arbitrary parameters. We recognize $(\alpha_\tau^{\tau\tau\tau})_+$ as the solution given in Section 2.3.3, with the substitution $a \to \varphi^{-\frac{1}{2}}$. The other solution represents a different, but equally valid, Fibonacci anyon model that makes use of the negative root (remember, the solution to the defining equation for the golden ratio, $\varphi + 1 = \varphi^2$, gives two solutions: $\frac{1\pm\sqrt{5}}{2}$) and corresponds to a model where the braiding phase for τ and I are effectively swapped—i.e. the eigenvalues of $\beta^{\tau\tau}$ defined in equation 29 are switched (modulo other minus signs that may crop up).[66]

However, the one-parameter family of solutions for each of these needs to be discussed. By *Ocneanu Rigidity*, up to equivalence, for a given set of fusion rules—i.e. a product functor for a monoidal category—there cannot be an infinite number of categories—there cannot be a one-parameter family of solutions to the pentagon equation [10]. The key in the definition is *up to equivalence.* Thus, we need to find a monoidal functor between different associators parametrized by a (and b) to show that these categories are equivalent. It is essential for us to show this as a

[64] i.e. few simple objects, for instance, the base 0-cell is **2**.

[65] For certain simpler categories, it can solve the entire associator—for **Fib** it will run out of memory unless the trivial elements are entered.

[66] Of course, there are also two different solutions of the hexagon equations for a given associator, corresponding to interchanging clockwise and counterclockwise braiding.

way of validating our solution given by equation 94.

In fact, our solver can handle this full problem; given a monoidal functor, $F : \mathbf{2} \to \mathbf{2}$, can we find a natural transformation $\phi : m \Rightarrow m$ (since the product functor in both categories is the same) and a 2-cell $\phi_u : u \Rightarrow u$ (where here $u : \mathbf{1} \to \mathbf{2}$ is the morphism that picks out the unit, and again, we have the same unit in both categories) that allow the functor to be coherent with the monoidal structures for both categories. For the functor, we choose the trivial functor, so that our anyons remain the same and do not transform into each other.

$$F = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{95}$$

Then, we attempt to solve the three coherence equations for monoidal functors (equations 11–13). First, we find no solution between categories given by the associators $(\alpha_\tau^{\tau\tau\tau})_+$ and $(\alpha_\tau^{\tau\tau\tau})_-$; these categories are inequivalent families, as expected. However, we *do* find the categories parametrized by a (or b) to be equivalent. For instance, for a, we find an equivalence between a category with $(\alpha_\tau^{\tau\tau\tau})_+$ and an arbitrary a, and a category with $(\alpha_\tau^{\tau\tau\tau})_+$ and $a = 1$. We find

$$\phi = \begin{bmatrix} [c] & \emptyset_{0\times 0} & \emptyset_{0\times 0} & \left[\frac{d^2}{ca}\right] \\ \emptyset_{0\times 0} & [c] & [c] & [d] \end{bmatrix} \tag{96}$$

$$\phi_u = \begin{bmatrix} \left[c^{-1}\right] \\ \emptyset_{0\times 0} \end{bmatrix}$$

where c and d are new arbitrary parameters. First, we see that ϕ *is* dependent on the choice of a. Second, for a given a and the trivial functor F, there is a *two*-parameter family of natural transformations that solve the coherence equations. The ability to quickly verify properties and structures of MTCs would be a huge advantage in categorical study of TQC, since MTCs are currently very difficult to work with. With this solver, we were easily able to identify equivalence between categories, rather than having to work through the three commuting diagrams of the

monoidal functor coherence equations by hand.

5.2 Higher-Level Structures

The ability to find higher-level structures in MTCs is a big motivation for this project. We are interested in the investigation of higher-level structures within MTCs in order to better understand the mathematical *why* behind topological quantum computation. We will be applying the same type of reasoning to anyons and MTCs that was applying to general quantum information principles and **Hilb** (i.e. **Vect** with some additional structure), as pioneered by Abramsky, Coecke, and collaborators [1, 11, 15, 16, 13].

Abramsky and Coecke [1] explain that by recasting quantum information in categorical terms, quantum compounds systems are naturally explained within the categorical structure. The monoidal structure of **Hilb** allows qubits to be combined, but does not allow for *cloning* or *deleting*. Furthermore, the fact that **Hilb** is rigid and symmetric allows for entangled pair creation and measurement. Adding to compact closure the fact that **Hilb** has *biproducts*, then we have an explanation for *quantum teleportation* and other similar protocols. Essentially, in diagrammatic language, the fact that a snake diagram can be pulled straight allows for teleportation. Coecke, Pavlovic and Vicary identify how the existence of a certain Frobenius algebra object in **Hilb** is the structure behind the existence of orthonormal bases—which is, in quantum protocols, essential to the preparation of states and measurement of outcomes [15]. Coecke and Perdrix find in [16] how classical channels and information exist within this structure and there interaction with quantum information. Essentially, the existence of Frobenius algebras allows *classical structures* and the interaction of quantum and classical information. From this, they work out graphical *spiders*, thus extending the graphical language to classical information and the environment.

We hope to apply similar reasoning to anyon models and further understand them within the setting of category theory (as MTCs). Unlike **Hilb**, all the additional structures of modular tensor categories have previously made them somewhat intractable to work with. The advent of our computer algebra system should alleviate much of the difficulty and allow more "high-level" categorical progress to be made.

References

[1] S. Abramsky and B. Coecke. A categorical semantics of quantum protocols. 2004.

[2] S. Abramsky and B. Coecke. Categorical quantum mechanics. *Handbook of quantum logic and quantum structures: quantum logic*, page 261, 2008.

[3] S. Abramsky and N. Tzevelekos. Introduction to categories and categorical logic. *New Structures for Physics*, pages 3–94, 2011.

[4] D. Adams. *The Hitchhiker's Guide to the Galaxy*. Pan Books, 1979.

[5] J. Baez and M. Stay. Physics, topology, logic and computation: a Rosetta Stone. *New structures for physics*, pages 95–172, 2011.

[6] J.C. Baez. Higher-dimensional algebra ii: 2-hilbert spaces. *Arxiv preprint q-alg/9609018*, 1996.

[7] B. Bakalov and A.A. Kirillov. *Lectures on tensor categories and modular functors*. American Mathematical Society, 2001.

[8] B. Bartlett, C. Douglas, C. Schommer-Pries, and J. Vicary. 123 TQFTs. *in preparation*, 2011.

[9] N.E. Bonesteel, L. Hormozi, G. Zikos, and S.H. Simon. Braid topologies for quantum computation. *Physical review letters*, 95(14):140503, 2005.

[10] D. Calaque and P. Etingof. Lectures on tensor categories. *Arxiv preprint math/0401246*, 2004.

[11] B. Coecke. Quantum information-flow, concretely, and axiomatically. *Arxiv preprint quant-ph/0506132*, 2005.

[12] B. Coecke. Quantum picturalism. *Arxiv preprint arXiv:0908.1787*, 2009.

[13] B. Coecke and R. Duncan. Interacting quantum observables. *Automata, Languages and Programming*, pages 298–310, 2010.

[14] B. Coecke and É.O. Paquette. Categories for the practising physicist. *New Structures for Physics*, pages 173–286, 2011.

[15] B. Coecke, D. Pavlovic, and J. Vicary. A new description of orthogonal bases. *Arxiv preprint arXiv:0810.0812*, 2008.

[16] B. Coecke and S. Perdrix. Environment and classical channels in categorical quantum mechanics. In *Computer Science Logic*, pages 230–244. Springer, 2010.

[17] B. Coecke, M. Sadrzadeh, and S. Clark. Mathematical Foundations for a Compositional Distributional Model of Meaning. *Arxiv preprint arXiv:1003.4394*, 2010.

[18] G. P. Collins. Computing with quantum knots. *Scientific American*, 294(4):56, 2006.

[19] J. Elgueta. A strict totally coordinatized version of Kapranov and Voevodsky's 2-category 2Vect. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 142, pages 407–428. Cambridge Univ Press, 2007.

[20] M.H. Freedman, A. Kitaev, M.J. Larsen, and Z. Wang. Topological quantum computation. *Bulletin of the American Mathematical Society*, 40(1):31–38, 2002.

[21] A. Joyal and R. Street. The geometry of tensor calculus. I. *Advances in Mathematics*, 88(1):55–112, 1991.

[22] M. Kapranov and V. Voevodsky. 2-categories and Zamolodchikov tetrahedra equations. In *Proc. Symp. Pure Math*, volume 56, pages 177–259, 1994.

[23] M. Kapranov and V. Voevodsky. Braided monoidal 2-categories and manin-schechtman higher braid groups. *Journal of Pure and Applied Algebra*, 92(3):241 – 267, 1994.

[24] A. Kitaev. Fault-tolerant quantum computation by anyons. *Annals of Physics*, 303(1):2–30, 2003.

[25] A. Kitaev. Anyons in an exactly solved model and beyond. *Annals of Physics*, 321(1):2–111, 2006.

[26] S. Mac Lane. *Categories for the working mathematician*. Springer, 1998.

[27] M. Müger. On the structure of modular categories. *Proceedings of the London Mathematical Society*, 87(2):291, 2003.

[28] C. Nayak, S. H. Simon, A. Stern, M. Freedman, and S. D. Sarma. Non-Abelian anyons and topological quantum computation. *Reviews of Modern Physics*, 80(3):1083, 2008.

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

[30] R. W. Ogburn and J. Preskill. Topological quantum computation. *Quantum Computing and Quantum Communications*, pages 341–356, 1999.

[31] P. Panangaden and É.O. Paquette. A categorical presentation of quantum computation with anyons. *New Structures for Physics*, pages 983–1025, 2011.

[32] J. Preskill. Lecture notes for physics 219: Quantum computation. Chapter 9: Topological quantum computation, 2004.

[33] E. Rowell, R. Stong, and Z. Wang. On classification of modular tensor categories. *Communications in Mathematical Physics*, 292(2):343–389, 2009.

[34] S.D. Sarma, M. Freedman, and C. Nayak. Topological quantum computation. *Physics Today*, 59(7):32–38, 2006.

[35] C. J. Schommer-Pries. *The Classification of Two-Dimensional Extended Topological Field Theories*. PhD thesis, University of California, Berkeley, 2009.

[36] V. G. Turaev. *Quantum invariants of knots and 3-manifolds*. Walter de Gruyter, 1994.

[37] S. Wolfram. *Mathematica: a system for doing mathematics by computer*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1988.