CATEGORIES IN CONTROL

JOHN C. BAEZ AND JASON ERBELE

Abstract. Control theory uses 'signal-flow diagrams' to describe processes where real-valued functions of time are added, multiplied by scalars, differentiated and integrated, duplicated and deleted. These diagrams can be seen as string diagrams for the symmetric monoidal category $\mathrm{FinVect}_k$ of finite-dimensional vector spaces over the field of rational functions $k = \mathbb{R}(s)$, where the variable s acts as differentiation and the monoidal structure is direct sum rather than the usual tensor product of vector spaces. For any field k we give a presentation of $\mathrm{FinVect}_k$ in terms of the generators used in signal-flow diagrams. A broader class of signal-flow diagrams also includes 'caps' and 'cups' to model feedback. We show these diagrams can be seen as string diagrams for the symmetric monoidal category FinRel_k, where objects are still finite-dimensional vector spaces but the morphisms are linear relations. We also give a presentation for FinRel_k. The relations say, among other things, that the 1-dimensional vector space k has two special commutative †-Frobenius structures, such that the multiplication and unit of either one and the comultiplication and counit of the other fit together to form a bimonoid. This sort of structure, but with tensor product replacing direct sum, is familiar from the 'ZX-calculus' obeyed by a finite-dimensional Hilbert space with two mutually unbiased bases.

1. Introduction

Control theory is the branch of engineering that focuses on manipulating 'open systems'—systems with inputs and outputs—to achieve desired goals. In control theory, 'signal-flow diagrams' are used to describe linear ways of manipulating signals, which we will take here to be smooth real-valued functions of time [10]. For a category theorist, at least, it is natural to treat signal-flow diagrams as string diagrams in a symmetric monoidal category [11, 12]. This forces some small changes of perspective, which we discuss below, but more important is the question: *which symmetric monoidal category?*

We shall argue that the answer is: the category FinRel_k of finite-dimensional vector spaces over a certain field k, but with *linear relations* rather than linear maps as morphisms, and *direct sum* rather than tensor product providing the symmetric monoidal structure. We use the field $k = \mathbb{R}(s)$ consisting of rational functions in one real variable s. This variable has the meaning of differentation. A linear relation from k^m to k^n is thus a system of linear constant-coefficient ordinary differential equations relating m 'input' signals and n 'output' signals.

Our main goal is to provide a complete 'generators and relations' picture of this symmetric monoidal category, with the generators being familiar components of signal-flow diagrams. It turns out that the answer has an intriguing but mysterious

Date: May 27, 2014 (revised May 20, 2015).

connection to ideas that are familiar in the diagrammatic approach to quantum theory. Quantum theory also involves linear algebra, but it uses linear maps between Hilbert spaces as morphisms, and the tensor product of Hilbert spaces provides the symmetric monoidal structure.

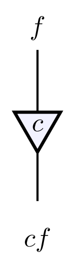
We hope that the category-theoretic viewpoint on signal-flow diagrams will shed new light on control theory. However, in this paper we only lay the groundwork. In Section 2 we introduce signal-flow diagrams and summarize our main results. In Section 3 we use signal-flow diagrams to give a presentation of $\mathrm{FinVect}_k$, the symmetric monoidal category of finite-dimensional vector spaces and linear maps. In Section 4 we use them to give a presentation of FinRel_k. In Section 5 we discuss a well-known example from control theory: an inverted pendulum on a cart. Finally, in Section 6 we compare our results to subsequent work of Bonchi–Sobociński–Zanasi [4, 5] and Wadsley–Woods [22].

2. Signal-flow diagrams

There are several basic operations that one wants to perform when manipulating signals. The simplest is multiplying a signal by a scalar. A signal can be amplified by a constant factor:

$$f \mapsto cf$$

where $c \in \mathbb{R}$. We can write this as a string diagram:

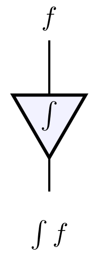


Here the labels f and cf on top and bottom are just for explanatory purposes and not really part of the diagram. Control theorists often draw arrows on the wires, but this is unnecessary from the string diagram perspective. Arrows on wires are useful to distinguish objects from their duals, but ultimately we will obtain a compact closed category where each object is its own dual, so the arrows can be dropped. What we really need is for the box denoting scalar multiplication to have a clearly defined input and output. This is why we draw it as a triangle. Control theorists often use a rectangle or circle, using arrows on wires to indicate which carries the input f and which the output cf.

A signal can also be integrated with respect to the time variable:

$$f \mapsto \int f.$$

Mathematicians typically take differentiation as fundamental, but engineers sometimes prefer integration, because it is more robust against small perturbations. In the end it will not matter much here. We can again draw integration as a string diagram:



Since this looks like the diagram for scalar multiplication, it is natural to extend $\mathbb{R}$ to $\mathbb{R}(s)$, the field of rational functions of a variable s which stands for differentiation. Then differentiation becomes a special case of scalar multiplication, namely multiplication by s, and integration becomes multiplication by $1/s$. Engineers accomplish the same effect with Laplace transforms, since differentiating a signal f is equivalent to multiplying its Laplace transform

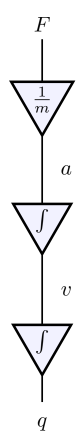
$$(\mathcal{L}f)(s) = \int_0^\infty f(t)e^{-st}\,dt$$

by the variable s. Another option is to use the Fourier transform: differentiating f is equivalent to multiplying its Fourier transform

$$(\mathcal{F}f)(\omega) = \int_{-\infty}^\infty f(t)e^{-i\omega t}\,dt$$

by $-i\omega$. Of course, the function f needs to be sufficiently well-behaved to justify calculations involving its Laplace or Fourier transform. At a more basic level, it also requires some work to treat integration as the two-sided inverse of differentiation. Engineers do this by considering signals that vanish for $t < 0$, and choosing the antiderivative that vanishes under the same condition. Luckily all these issues can be side-stepped in a formal treatment of signal-flow diagrams: we can simply treat signals as living in an unspecified vector space over the field $\mathbb{R}(s)$. The field $\mathbb{C}(s)$ would work just as well, and control theory relies heavily on complex analysis. In most of this paper we work over an arbitrary field k.

The simplest possible signal processor is a rock, which takes the 'input' given by the force F on the rock and produces as 'output' the rock's position q. Thanks to Newton's second law $F = ma$, we can describe this using a signal-flow diagram:



Here composition of morphisms is drawn in the usual way, by attaching the output wire of one morphism to the input wire of the next.

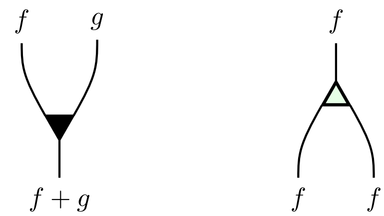
To build more interesting machines we need more building blocks, such as addition:

$$+\colon (f, g) \mapsto f + g$$

and duplication:

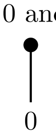
$$\Delta\colon f \mapsto (f, f)$$

When these linear maps are written as matrices, their matrices are transposes of each other. This is reflected in the string diagrams for addition and duplication:



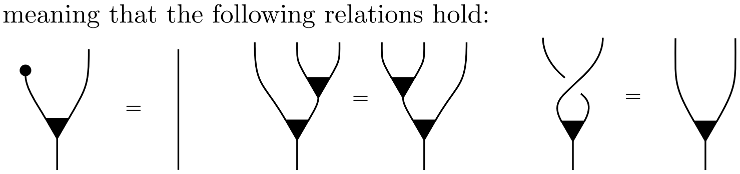
The second is essentially an upside-down version of the first. However, we draw addition as a dark triangle and duplication as a light one because we will later want another way to 'turn addition upside-down' that does *not* give duplication. As an added bonus, a light upside-down triangle resembles the Greek letter Δ, the usual symbol for duplication.

While they are typically not considered worthy of mention in control theory, for completeness we must include two other building blocks. One is the zero map from $\{0\}$ to our field k, which we denote as 0 and draw as follows:



The other is the zero map from k to $\{0\}$, sometimes called 'deletion', which we denote as ! and draw thus:

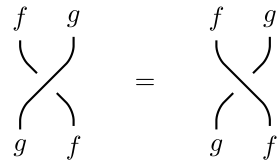
Just as the matrices for addition and duplication are transposes of each other, so are the matrices for zero and deletion, though they are rather degenerate, being 1×0 and 0×1 matrices, respectively. Addition and zero make k into a commutative monoid, meaning that the following relations hold:



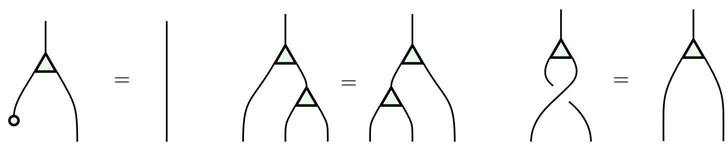
The equation at right is the commutative law, and the crossing of strands is the 'braiding'

$$B\colon (f, g) \mapsto (g, f)$$

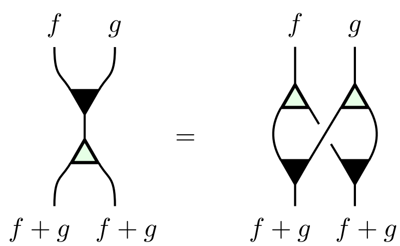
by which we switch two signals. In fact this braiding is a 'symmetry', so it does not matter which strand goes over which:



Dually, duplication and deletion make k into a cocommutative comonoid. This means that if we reflect the equations obeyed by addition and zero across the horizontal axis and turn dark operations into light ones, we obtain another set of valid equations:



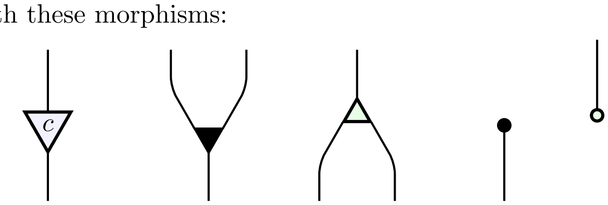
There are also relations between the monoid and comonoid operations. For example, adding two signals and then duplicating the result gives the same output as duplicating each signal and then adding the results:



This diagram is familiar in the theory of Hopf algebras, or more generally bialgebras. Here it is an example of the fact that the monoid operations on k are comonoid homomorphisms—or equivalently, the comonoid operations are monoid homomorphisms. We summarize this situation by saying that k is a **bimonoid**.

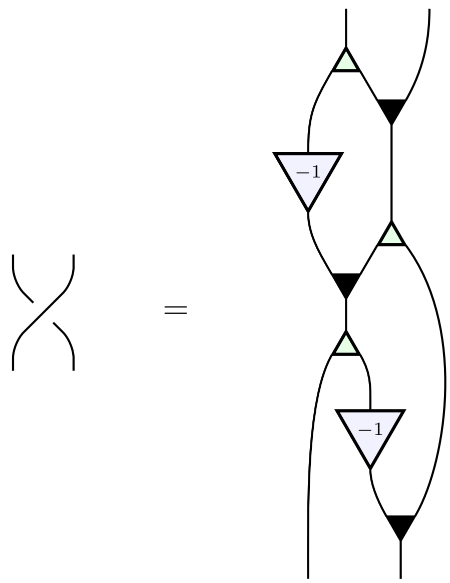
So far all our string diagrams denote linear maps. We can treat these as morphisms in the category $\mathrm{FinVect}_k$, where objects are finite-dimensional vector spaces over a field k and morphisms are linear maps. This category is equivalent to a skeleton where the only objects are vector spaces k^n for $n \geq 0$, and then morphisms can be seen as $n \times m$ matrices. The space of signals is a vector space V over k which may not be finite-dimensional, but this does not cause a problem: an $n \times m$ matrix with entries in k still defines a linear map from V^n to V^m in a functorial way.

In applications of string diagrams to quantum theory [3, 8], we make $\mathrm{FinVect}_k$ into a symmetric monoidal category using the tensor product of vector spaces. In control theory, we instead make $\mathrm{FinVect}_k$ into a symmetric monoidal category using the *direct sum* of vector spaces. In Lemma 1 we prove that for any field k, $\mathrm{FinVect}_k$ with direct sum is generated as a symmetric monoidal category by the one object k together with these morphisms:



where $c \in k$ is arbitrary.

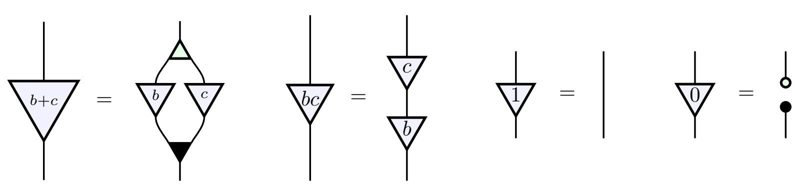
However, these generating morphisms obey some unexpected relations! For example, we have:



Thus, it is important to find a complete set of relations obeyed by these generating morphisms, thus obtaining a presentation of $\mathrm{FinVect}_k$ as a symmetric monoidal category. We do this in Theorem 2. In brief, these relations say:

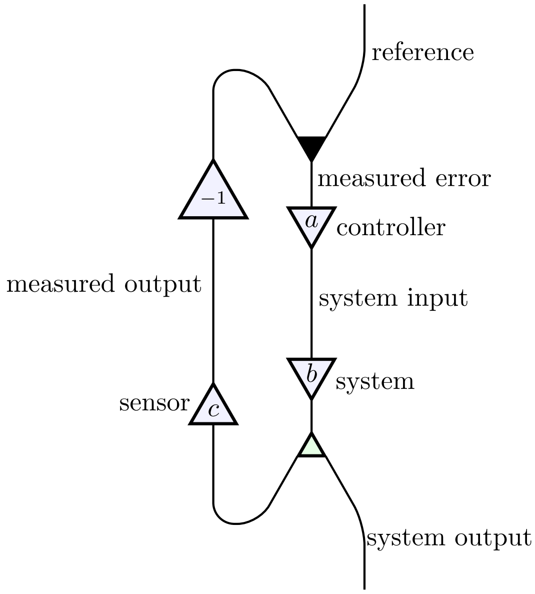
(1) $(k, +, 0, \Delta, !)$ is a bicommutative bimonoid;
(2) the rig operations of k can be recovered from the generating morphisms;
(3) all the generating morphisms commute with scalar multiplication.

Here item (2) means that $+$, $\cdot$, 0 and 1 in the field k can be expressed in terms of signal-flow diagrams as follows:



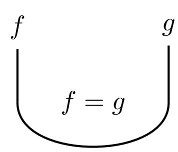
Multiplicative inverses cannot be so expressed, so our signal-flow diagrams so far do not know that k is a field. Additive inverses also cannot be expressed in this way. And indeed, a version of Theorem 2 holds whenever k is a commutative rig: that is, a commutative 'ring without negatives', such as $\mathbb{N}$. See Section 6 for details.

While Theorem 2 is a step towards understanding the category-theoretic underpinnings of control theory, it does not treat signal-flow diagrams that include 'feedback'. Feedback is one of the most fundamental concepts in control theory because a control system without feedback may be highly sensitive to disturbances or unmodeled behavior. Feedback allows these uncontrolled behaviors to be mollified. As a string diagram, a basic feedback system might look schematically like this:

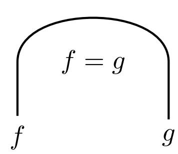


The user inputs a 'reference' signal, which is fed into a controller, whose output is fed into a system, or 'plant', which in turn produces its own output. But then the system's output is duplicated, and one copy is fed into a sensor, whose output is added (or if we prefer, subtracted) from the reference signal.

In string diagrams—unlike in the usual thinking on control theory—it is essential to be able to read any diagram from top to bottom as a composite of tensor products of generating morphisms. Thus, to incorporate the idea of feedback, we need two more generating morphisms. These are the 'cup':



and 'cap':



These are not maps: they are relations. The cup imposes the relation that its two inputs be equal, while the cap does the same for its two outputs. This is a way of describing how a signal flows around a bend in a wire.

To make this precise, we use a category called FinRel_k. An object of this category is a finite-dimensional vector space over k, while a morphism from U to V, denoted $L\colon U \nrightarrow V$, is a **linear relation**, meaning a linear subspace

$$L \subseteq U \oplus V.$$

In particular, when $k = \mathbb{R}(s)$, a linear relation $L\colon k^m \to k^n$ is just an arbitrary system of constant-coefficient linear ordinary differential equations relating m input variables and n output variables.

Since the direct sum $U \oplus V$ is also the cartesian product of U and V, a linear relation is indeed a relation in the usual sense, but with the property that if $u \in U$ is related to $v \in V$ and $u' \in U$ is related to $v' \in V$ then $cu + c'u'$ is related to $cv + c'v'$ whenever $c, c' \in k$. We compose linear relations $L\colon U \nrightarrow V$ and $L'\colon V \nrightarrow W$ as follows:

$$L'L = \{(u, w)\colon\ \exists\, v \in V \ \ (u, v) \in L \text{ and } (v, w) \in L'\}.$$

Any linear map $f\colon U \to V$ gives a linear relation $F\colon U \nrightarrow V$, namely the graph of that map:

$$F = \{(u, f(u)) : u \in U\}.$$

Composing linear maps thus becomes a special case of composing linear relations, so $\mathrm{FinVect}_k$ becomes a subcategory of FinRel_k. Furthermore, we can make FinRel_k into a monoidal category using direct sums, and it becomes symmetric monoidal using the braiding already present in $\mathrm{FinVect}_k$.

In these terms, the **cup** is the linear relation

$$\cup\colon k^2 \nrightarrow \{0\}$$

given by

$$\cup \ = \ \{(x, x, 0) : x \in k\} \ \subseteq \ k^2 \oplus \{0\},$$

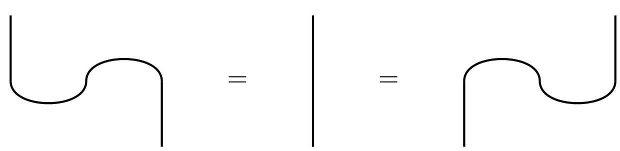
while the **cap** is the linear relation

$$\cap\colon \{0\} \nrightarrow k^2$$

given by

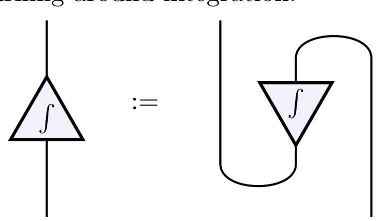
$$\cap \ = \ \{(0, x, x) : x \in k\} \ \subseteq \ \{0\} \oplus k^2.$$

These obey the **zigzag relations**:



Thus, they make FinRel_k into a compact closed category where k, and thus every object, is its own dual.

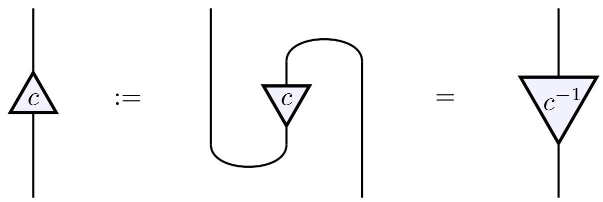
Besides feedback, one of the things that make the cap and cup useful is that they allow any morphism $L\colon U \nrightarrow V$ to be 'plugged in backwards' and thus 'turned around'. For instance, turning around integration:



we obtain differentiation. In general, using caps and cups we can turn around any linear relation $L\colon U \nrightarrow V$ and obtain a linear relation $L^\dagger\colon V \nrightarrow U$, called the **adjoint** of L, which turns out to given by

$$L^\dagger = \{(v,u) : (u,v) \in L\}.$$

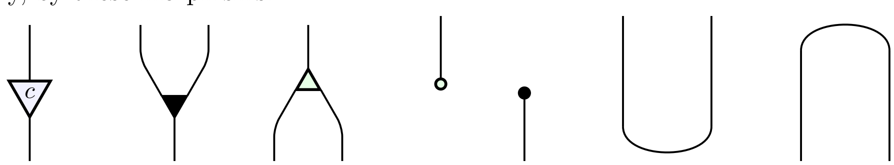
For example, if $c \in k$ is nonzero, the adjoint of scalar multiplication by c is multiplication by c^{-1}:



Thus, caps and cups allow us to express multiplicative inverses in terms of signal-flow diagrams! One might think that a problem arises when when $c = 0$, but no: the adjoint of scalar multiplication by 0 is

$$\{(0,x) : x \in k\} \subseteq k \oplus k.$$

In Lemma 3 we show that FinRel_k is generated, as a symmetric monoidal category, by these morphisms:



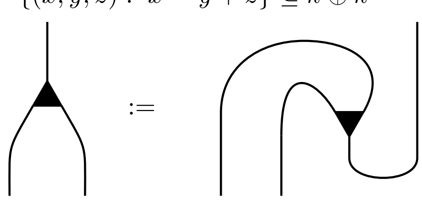
where $c \in k$ is arbitrary.

In Theorem 4 we find a complete set of relations obeyed by these generating morphisms, thus giving a presentation of FinRel_k as a symmetric monoidal category. To describe these relations, it is useful to work with adjoints of the generating morphisms. We have already seen that the adjoint of scalar multiplication by c is scalar multiplication by c^{-1}, except when $c = 0$. Taking adjoints of the other four generating morphisms of $\mathrm{FinVect}_k$, we obtain four important but perhaps unfamiliar linear relations. We draw these as 'turned around' versions of the original generating morphisms:

- **Coaddition** is a linear relation from k to k^2 that holds when the two outputs sum to the input:

$$+^\dagger\colon k \nrightarrow k^2$$

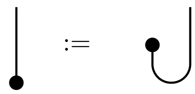
$$+^\dagger = \{(x,y,z) :\ x = y + z\} \subseteq k \oplus k^2$$



- **Cozero** is a linear relation from k to $\{0\}$ that holds when the input is zero:

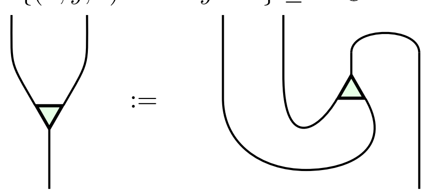
$$0^\dagger\colon k \nrightarrow \{0\}$$

$$0^\dagger = \{(0,0)\} \subseteq k \oplus \{0\}$$



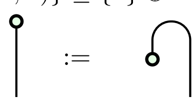
- **Coduplication** is a linear relation from k^2 to k that holds when the two inputs both equal the output:

$$\Delta^\dagger \colon k^2 \nrightarrow k$$
$$\Delta^\dagger = \{(x,y,z) : \; x = y = z\} \subseteq k^2 \oplus k$$



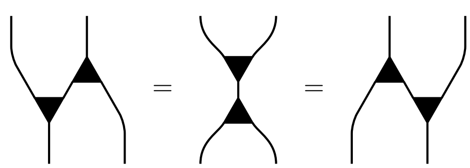
- **Codeletion** is a linear relation from $\{0\}$ to k that holds always:

$$!^\dagger \colon \{0\} \nrightarrow k$$
$$!^\dagger = \{(0,x)\} \subseteq \{0\} \oplus k$$



Since $+^\dagger, 0^\dagger, \Delta^\dagger$ and $!^\dagger$ automatically obey turned-around versions of the relations obeyed by $+, 0, \Delta$ and $!$, we see that k acquires a *second* bicommutative bimonoid structure when considered as an object in FinRel_k.

Moreover, the four dark operations make k into a **Frobenius monoid**. This means that $(k, +, 0)$ is a monoid, $(k, +^\dagger, 0^\dagger)$ is a comonoid, and the **Frobenius relation** holds:

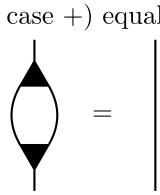


All three expressions in this equation are linear relations saying that the sum of the two inputs equal the sum of the two outputs.

The operation sending each linear relation to its adjoint extends to a contravariant functor

$$\dagger \colon \mathrm{FinRel}_k \to \mathrm{FinRel}_k,$$

which obeys a list of properties that are summarized by saying that FinRel_k is a '†-compact' category [1, 20]. Because two of the operations in the Frobenius monoid $(k, +, 0, +^\dagger, 0^\dagger)$ are adjoints of the other two, it is a **†-Frobenius monoid**. This Frobenius monoid is also **special**, meaning that comultiplication (in this case $+^\dagger$) followed by multiplication (in this case $+$) equals the identity:



This Frobenius monoid is also commutative—and cocommutative, but for Frobenius monoids this follows from commutativity.

Starting around 2008, commutative special †-Frobenius monoids have become important in the categorical foundations of quantum theory, where they can be understood as 'classical structures' for quantum systems [9, 21]. The category FinHilb of finite-dimensional Hilbert spaces and linear maps is a †-compact category, where any linear map $f\colon H \to K$ has an adjoint $f^\dagger\colon K \to H$ given by

$$\langle f^\dagger \phi, \psi \rangle = \langle \phi, f\psi \rangle$$

for all $\psi \in H, \phi \in K$. A commutative special †-Frobenius monoid in FinHilb is then the same as a Hilbert space with a chosen orthonormal basis. The reason is that given an orthonormal basis ψ_i for a finite-dimensional Hilbert space H, we can make H into a commutative special †-Frobenius monoid with multiplication $m\colon H \otimes H \to H$ given by

$$m(\psi_i \otimes \psi_j) = \left\{ \begin{array}{cc} \psi_i & i = j \\ 0 & i \neq j \end{array} \right.$$

and unit $i\colon \mathbb{C} \to H$ given by

$$i(1) = \sum_i \psi_i .$$

The comultiplication $m^\dagger$ duplicates basis states:

$$m^\dagger(\psi_i) = \psi_i \otimes \psi_i .$$

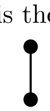
Conversely, any commutative special †-Frobenius monoid in FinHilb arises this way.

Considerably earlier, around 1995, commutative Frobenius monoids were recognized as important in topological quantum field theory. The reason, ultimately, is that the free symmetric monoidal category on a commutative Frobenius monoid is 2Cob, the category with 2-dimensional oriented cobordisms as morphisms: see Kock's textbook [13] and the many references therein. But the free symmetric monoidal category on a commutative *special* Frobenius monoid was worked out even earlier [6, 14, 19]: it is the category with finite sets as objects, where a morphism $f\colon X \to Y$ is an isomorphism class of cospans

$$X \longrightarrow S \longleftarrow Y.$$

This category can be made into a †-compact category in an obvious way, and then the 1-element set becomes a commutative special †-Frobenius monoid.

For all these reasons, it is interesting to find a commutative special †-Frobenius monoid lurking at the heart of control theory! However, the Frobenius monoid here has yet another property, which is more unusual. Namely, the unit $0\colon \{0\} \nrightarrow k$ followed by the counit $0^\dagger\colon k \nrightarrow \{0\}$ is the identity:

 =

We call a special Frobenius monoid that also obeys this extra law **extra-special**. One can check that the free symmetric monoidal category on a commutative extra-special Frobenius monoid is the category with finite sets as objects, where a morphism $f\colon X \to Y$ is an equivalence relation on the disjoint union $X \sqcup Y$, and we compose $f\colon X \to Y$ and $g\colon Y \to Z$ by letting f and g generate an equivalence relation on $X \sqcup Y \sqcup Z$ and then restricting this to $X \sqcup Z$.

As if this were not enough, the light operations share many properties with the dark ones. In particular, these operations make k into a commutative extra-special †-Frobenius monoid in a second way. In summary:

- $(k, +, 0, \Delta, !)$ is a bicommutative bimonoid;
- $(k, \Delta^\dagger, !^\dagger, +^\dagger, 0^\dagger)$ is a bicommutative bimonoid;
- $(k, +, 0, +^\dagger, 0^\dagger)$ is a commutative extra-special †-Frobenius monoid;
- $(k, \Delta^\dagger, !^\dagger, \Delta, !)$ is a commutative extra-special †-Frobenius monoid.

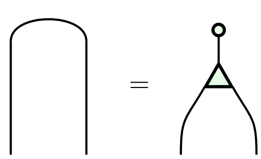
It should be no surprise that with all these structures built in, signal-flow diagrams are a powerful method of designing processes. However, it is surprising that most of these structures are present in a seemingly very different context: the so-called 'ZX calculus', a diagrammatic formalism for working with complementary observables in quantum theory [7]. This arises naturally when one has an n-dimensional Hilbert space H with two orthonormal bases ψ_i, ϕ_i that are 'mutually unbiased', meaning that

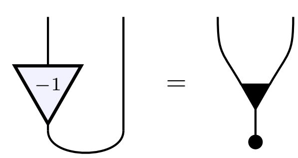
$$|\langle \psi_i, \phi_j \rangle|^2 = \frac{1}{n}$$

for all $1 \le i, j \le n$. Each orthonormal basis makes H into commutative special †-Frobenius monoid in FinHilb. Moreover, the multiplication and unit of either one of these Frobenius monoids fits together with the comultiplication and counit of the other to form a bicommutative bimonoid. So, we have all the structure present in the list above—except that these Frobenius monoids are only extra-special if H is 1-dimensional.

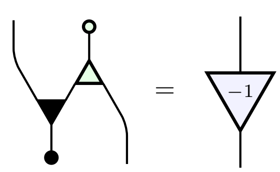
The field k is also a 1-dimensional vector space, but this is a red herring: in FinRel_k *every* finite-dimensional vector space naturally acquires all four structures listed above, since addition, zero, duplication and deletion are well-defined and obey all the relations we have discussed. We focus on k in this paper simply because it generates all the objects FinRel_k via direct sum.

Finally, in FinRel_k the cap and cup are related to the light and dark operations as follows:





Note the curious factor of -1 in the second equation, which breaks some of the symmetry we have seen so far. This equation says that two elements $x, y \in k$ sum to zero if and only if $-x = y$. Using the zigzag relations, the two equations above give



We thus see that in FinRel_k, both additive and multiplicative inverses can be expressed in terms of the generating morphisms used in signal-flow diagrams.

Theorem 4 gives a presentation of FinRel_k based on the ideas just discussed. Briefly, it says that FinRel_k is equivalent to the symmetric monoidal category generated by an object k and these morphisms:

(1) addition $+\colon k^2 \nrightarrow k$
(2) zero $0\colon \{0\} \nrightarrow k$
(3) duplication $\Delta\colon k \nrightarrow k^2$
(4) deletion $!\colon k \nrightarrow 0$
(5) scalar multiplication $c\colon k \nrightarrow k$ for any $c \in k$
(6) cup $\cup\colon k^2 \nrightarrow \{0\}$
(7) cap $\cap\colon \{0\} \nrightarrow k^2$

obeying these relations:

(1) $(k, +, 0, \Delta, !)$ is a bicommutative bimonoid;
(2) $\cap$ and $\cup$ obey the zigzag equations;
(3) $(k, +, 0, +^\dagger, 0^\dagger)$ is a commutative extra-special †-Frobenius monoid;
(4) $(k, \Delta^\dagger, !^\dagger, \Delta, !)$ is a commutative extra-special †-Frobenius monoid;
(5) the field operations of k can be recovered from the generating morphisms;
(6) the generating morphisms (1)-(4) commute with scalar multiplication.

Note that item (2) makes FinRel_k into a †-compact category, allowing us to mention the adjoints of generating morphisms in the subsequent relations. Item (5) means that $+, \cdot, 0, 1$ and also additive and multiplicative inverses in the field k can be expressed in terms of signal-flow diagrams in the manner we have explained.

3. A presentation of $\mathrm{FinVect}_k$

Our goal in this section is to find a presentation for the symmetric monoidal category $\mathrm{FinVect}_k$. To simplify some technicalities, we shall use Mac Lane's coherence theorem [17] to choose a symmetric monoidal equivalence $F\colon \mathrm{FinVect}'_k \to \mathrm{FinVect}_k$ where $\mathrm{FinVect}'_k$ is strict. This allows us to avoid mentioning associators and unitors, since in $\mathrm{FinVect}'_k$ these are identity morphisms. In what follows, we call $\mathrm{FinVect}'_k$ simply $\mathrm{FinVect}_k$, and call objects and morphisms in $\mathrm{FinVect}_k$ by the names of their images under F. Colloquially speaking, we 'work in a strict version' of $\mathrm{FinVect}_k$, and do not bother to indicate that this is a different (though equivalent) symmetric monoidal category.

We say a strict symmetric monoidal category C is **generated** by a set O of objects and a set M of morphisms going between tensor products of objects in O if the smallest subcategory C_0 of C containing:

- the objects in O,
- the morphisms in M,
- the tensor products of any objects or morphisms in C_0
- the braiding for any pair of objects in C_0

has the property that the inclusion $i\colon C_0 \to C$ is an equivalence of categories. It follows that i extends to an equivalence of symmetric monoidal categories. In this situation we call the elements of O **generating objects** for C, and call the elements of M **generating morphisms**.

Lemma 1. *For any field k, the object k together with the morphisms:*

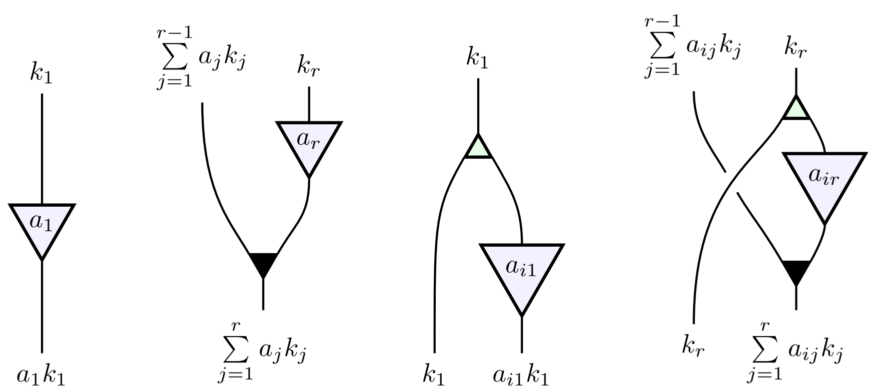
(1) scalar multiplication $c\colon k \to k$ for any $c \in k$
(2) addition $+\colon k \oplus k \to k$

(3) zero $0\colon \{0\} \to k$
(4) duplication $\Delta\colon k \to k \oplus k$
(5) deletion $!\colon k \to \{0\}$

generate $\mathrm{FinVect}_k$*, the category of finite-dimensional vector spaces over* k *and linear maps, as a symmetric monoidal category.*

Proof. It suffices to show that k together with the morphisms in (1)–(5) generate the full subcategory of $\mathrm{FinVect}_k$ containing only the iterated direct sums $k^n = k \oplus \cdots \oplus k$, since this is equivalent to $\mathrm{FinVect}_k$.

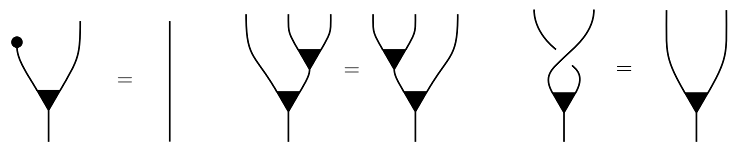
A linear map in $\mathrm{FinVect}_k$, $T\colon k^m \to k^n$ can be expressed as n k-linear combinations of m elements of k. That is, $T(k_1, \ldots, k_m) = (\sum_j a_{1j}k_j, \ldots, \sum_j a_{nj}k_j)$, $a_{ij} \in k$. Any k-linear combination of r elements can be constructed with only addition, multiplication, and zero, with zero only necessary when providing the unique k-linear combination for $r = 0$. When $r = 1$, $a_1(k_1)$ is an arbitrary k-linear combination. For $r > 1$, $+(S_{r-1}, a_r(k_r))$ yields an arbitrary k-linear combination on r elements, where S_{r-1} is an arbitrary k-linear combination of $r - 1$ elements. The inclusion of duplication allows process of forming k-linear combinations to be repeated an arbitrary (finite) positive number of times, and deletion allows the process to be repeated zero times. When n k-linear combinations are needed, each input may be duplicated $n - 1$ times. Because $\mathrm{FinVect}_k$ is being generated as a symmetric monoidal category, the mn outputs can then be permuted into n collections of m outputs: one output from each input for each collection. Each collection can then form a k-linear combination, as above. The following diagrams illustrate the pieces that form this inductive argument.



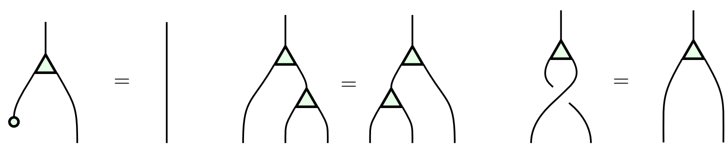
Since multiplication provides the map $k_1 \mapsto a_1k_1$, as in the far left diagram, the middle-left diagram can be used inductively to form a k-linear combination of any number of inputs. In particular, we have any linear map $S_r\colon k^m \to k$ given by $(k_1, \ldots, k_m) \mapsto (\sum_j a_{rj}k_j)$. Using duplication as in the middle-right diagram, one can produce the map $k_1 \mapsto (k_1, a_{i1}k_1)$, to which the right diagram can be inductively applied. Thus we can build any linear map, $T_j \in \mathrm{FinVect}_k$, $T_j\colon k^m \to k^{m+1}$ given by $(k_1, \ldots, k_m) \mapsto (k_1, \ldots, k_m, \sum_j a_{ij}k_j)$. If we represent the identity map on k^r as 1^r, the r-fold tensor product of the identity map on k, any linear map $T\colon k^m \to k^n$ can be given by $(k_1, \ldots, k_m) \mapsto (\sum_j a_{1j}k_j, \ldots, \sum_j a_{nj}k_j)$, which can be expressed as $T = (S_1 \oplus 1^{n-1})(T_2 \oplus 1^{n-2}) \cdots (T_{n-1} \oplus 1^1)T_n$. The above works as long as the vector spaces are not 0-dimensional. $f\colon k^m \to \{0\}$ can be written as an m-fold tensor product of deletion, $!^m$, and $f\colon \{0\} \to k^n$ can be written as an n-fold tensor product of zero, 0^n. $f\colon \{0\} \to \{0\}$ is the empty morphism, which has an empty diagram for its string diagram. $\square$

It is easy to see that the morphisms given in Lemma 1 obey the following 18 relations:

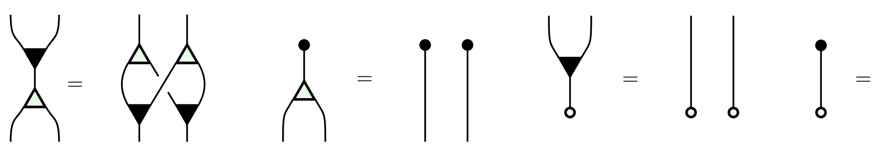
(1)–**(3)** Addition and zero make k into a commutative monoid:



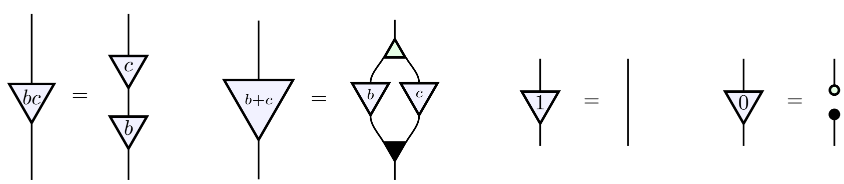
(4)–**(6)** Duplication and deletion make k into a cocommutative comonoid:



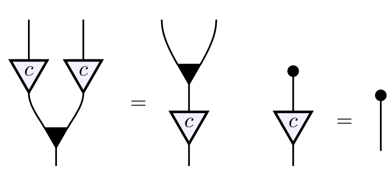
(7)–**(10)** The monoid and comonoid structures on k fit together to form a bimonoid:



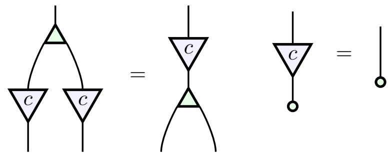
(11)–**(14)** The rig structure of k can be recovered from the generating morphisms:



(15)–**(16)** Scalar multiplication commutes with addition and zero:



(17)–**(18)** Scalar multiplication commutes with duplication and deletion:



In fact, these relations are enough. That is, together with the generating objects and morphisms, they give a 'presentation' of $\mathrm{FinVect}_k$ as a symmetric monoidal category. However, we need to make this concept precise.

Suppose C is generated by a set O of objects and a set M of morphisms going between tensor products of objects in O. Define a **formal morphism** to be a formal expression built from symbols for morphisms in M via composition, identity morphisms, tensor product, the unit object and the braiding. Any formal morphism f can be **evaluated** to obtain a morphism $\mathrm{ev}(f)$ in C, which actually lies in C_0.

Define a **relation** to be a pair f, g of formal morphisms. We say the relation **holds** in C if $\mathrm{ev}(f) = \mathrm{ev}(g)$. Suppose R is a set of relations that hold in C. We say (O, M, R) is a **presentation** of C if given any two formal morphisms j, k that evaluate to the same morphism, then we can go from j to k via a finite sequence of moves of these kinds:

(1) replacing an instance of a generating morphism f in a formal morphism by the generating morphism g, where $(f, g) \in R$,
(2) applying an equational law in the definition of strict symmetric monoidal category to a formal morphism.

In intuitive terms, this means that there are enough relations to prove all the equations that hold in C—or more precisely, in the equivalent category C_0.

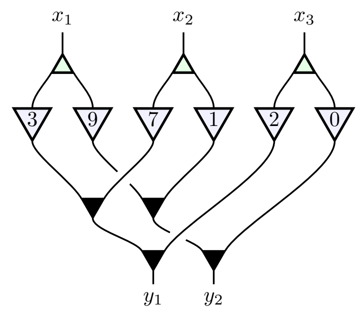
Theorem 2. *The symmetric monoidal category* $\mathrm{FinVect}_k$ *is presented by the object* k*, the morphisms given in Lemma 1, and relations* **(1)–(18)** *as listed above.*

Proof. To prove this, we show that these relations suffice to rewrite any formal morphism into a standard form, with all formal morphisms that evaluate to the same morphism $T\colon k^m \to k^n$ in $\mathrm{FinVect}_k$ having the same standard form. To deal with moves of type (2), we draw formal morphisms as string diagrams built from generating morphisms and the braiding. Two formal morphisms that differ only by equational laws in the definition of strict symmetric monoidal category will have topologically equivalent string diagrams. It suffices, then, to show that any string diagram built from generating morphisms and the braiding can be put into a standard form using topological equivalences and relations **(1)–(18)**.

A qualitative description of this standard form will be helpful for understanding how an arbitrary string diagram can be rewritten in this form. By way of example, consider the linear transformation $T\colon \mathbb{R}^3 \to \mathbb{R}^2$ given by

$$(x_1, x_2, x_3) \mapsto (y_1, y_2) = (3x_1 + 7x_2 + 2x_3, 9x_1 + x_2).$$

Its standard form looks like this:



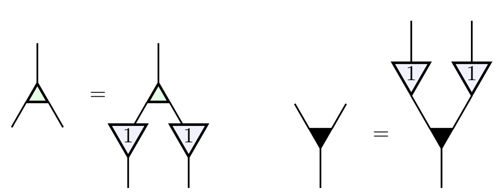
This is a string diagram picture of the following equation:

$$Tx = \begin{pmatrix} 3 & 7 & 2 \\ 9 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

In general, given a k-linear transformation $T \colon k^m \to k^n$, we can describe it using an $n \times m$ matrix with entries in k. The case where m and/or n is zero gives a matrix with no entries, so their standard form will be treated separately. For positive values of m and n, the standard form has three distinct layers. The top layer consists of m clusters of $n-1$ instances of Δ. The middle layer is mn multiplications. The n outputs of the jth cluster connect to the inputs of the multiplications $\{a_{1j}, \dots, a_{nj}\}$, where a_{ij} is the ij entry of A, the matrix for T. The bottom layer consists of n clusters of $m-1$ instances of $+$. There will generally be braiding in this layer as well, but since the category is being generated as symmetric monoidal, the locations of the braidings doesn't matter so long as the topology of the string diagram is preserved. The topology of the sum layer is that the ith sum cluster gets its m inputs from the outputs of the multiplications $\{a_{i1}, \dots, a_{im}\}$. The arrangement of the instances of Δ and $+$ within their respective clusters does not matter, due to the associativity of $+$ via relation **(2)** and coassociativity of Δ via relation **(5)**. For the sake of making the standard form explicit with respect to these relations, we may assume the right output of a Δ is always connected to a multiplication input, and the right input of a $+$ is always connected to a multiplication output. This gives a prescription for drawing the standard form of a string diagram with a corresponding matrix A.

The standard form for $T \colon k^0 \to k^n$ is n zeros $(0 \oplus \cdots \oplus 0)$, and the standard form for $T \colon k^m \to k^0$ is m deletions $(! \oplus \cdots \oplus !)$.

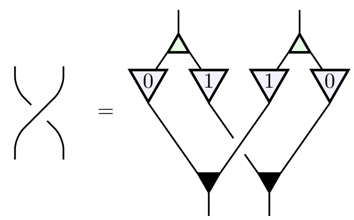
Each of the generating morphisms can easily be put into standard form: the string diagrams for zero, deletion, and multiplication are already in standard form. The string diagram for duplication (resp. addition) can be put into standard form by attaching a multiplication by 1, relation **(13)**, to each of the outputs (resp. inputs).



The braiding morphism is just as basic to our argument as the generating morphisms, so we will need to write the string diagram for B in standard form as well. The matrix corresponding to braiding is

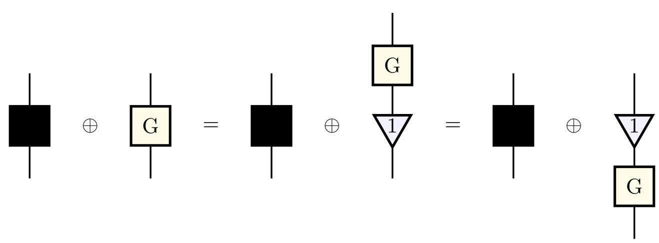
$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

so its standard form is as follows:



For $n > 1$, any morphism built from n copies of the **basic morphisms**—that is, generating morphisms and the braiding—can be built up from a morphism built from $n-1$ copies by composing or tensoring with one more basic morphism. Thus, to prove that any string diagram built from basic morphisms can be put into its standard form, we can proceed by induction on the number of basic morphisms.

Furthermore, because strings can be extended using the identity morphism, relation **(13)** can be used to show tensoring with any generating morphism is equivalent to tensoring with 1, followed by a composition: $\Delta = \Delta \circ 1$, $+ = 1 \circ +$, $c = 1 \circ c$, $! = ! \circ 1$, $0 = 1 \circ 0$. In the case of braiding, the step of tensoring with 1 is repeated once before making the composition: $B = (1 \oplus 1) \circ B$.

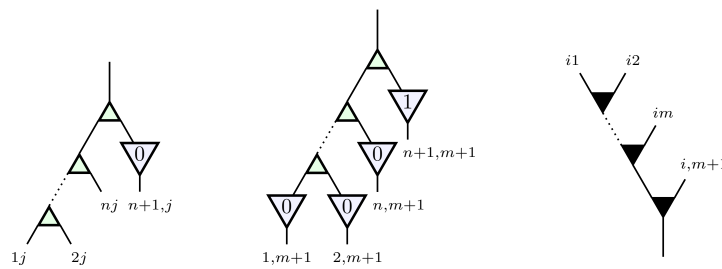


Thus there are 11 cases to consider for this induction: $\oplus 1$, $+\circ$, $\circ\Delta$, $\Delta\circ$, $\circ+$, $\circ c$, $c\circ$, $\circ 0$, $!\circ$, $B\circ$, $\circ B$. Without loss of generality, the string diagram S to which a generating morphism is added will be assumed to be in standard form already. Labels ij on diagrams illustrating these cases correspond to strings incident to the multiplications a_{ij}.

- **$\oplus$1**
  When tensoring morphisms together, the matrix corresponding to $C \oplus D$ is the block diagonal matrix

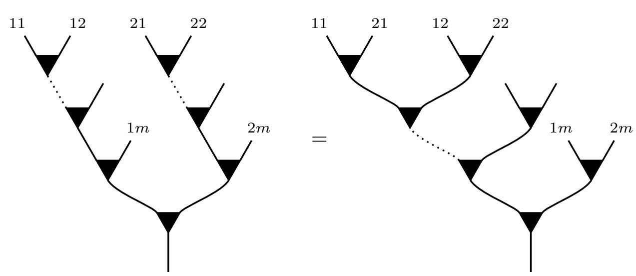
$$\begin{pmatrix} C & 0 \\ 0 & D \end{pmatrix},$$

  where, by abuse of notation, the block C is the matrix corresponding to morphism C, and respectively D with D. Thus, when tensoring S by 1, we write the matrix for S with one extra row and one extra column. Each of these new entries will be 0 with the exception of a 1 at the bottom of the extra column. The string diagram corresponding to the new matrix can be drawn in standard form as prescribed above. Using relations **(14)**, **(4)**, and **(1)**, the standard form reduces to $S \oplus 1$. The process is reversable ($\mathrm{ev}(f) = \mathrm{ev}(g)$ implies $\mathrm{ev}(g) = \mathrm{ev}(f)$), so if the string diagram S can be drawn in standard form, the string diagram $S \oplus 1$ can be drawn in standard form, too. The diagrams below show the relevant strings before they are reduced.

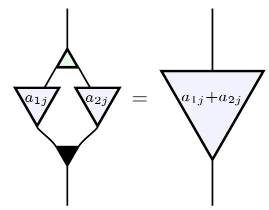


Note that for $i = n+1$ the multiplications $a_{i2}, \dots, a_{im}$ going to the sum cluster will be multiplication by zero, and $a_{i,m+1} = 1$. Otherwise $a_{i,m+1} = 0$, and the rest depend on the matrix corresponding to S. When $S = (! \oplus \cdots \oplus !)$, the matrix corresponding to $S \oplus 1$ has a single row, $(0 \cdots 0\ 1)$, and the standard form generated is just the middle diagram above. When the same simplifications are applied, no sum cluster exists to eliminate the zeros, so the standard form still simplifies to $S \oplus 1$. Dually, when $S = (0 \oplus \cdots \oplus 0)$, the matrix representation of $S \oplus 1$ is a column matrix. No duplication cluster exists in the standard form for this matrix, so the same simplifications again reduce to $S \oplus 1$.

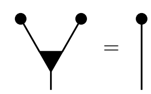
- $+\circ$
  If we compose the string diagram for addition with S, first consider only the affected clusters of additions: two clusters are combined into a larger cluster. Without loss of generality we can assume these are the first two clusters, or formally, $(+ \oplus 1^{n-2})(S)$. We can rearrange the sums using the associative law, relation **(2)**, and permute the inputs of this large cluster using the commutative law, relation **(3)**. After several iterations of these two relations, the desired result is obtained:



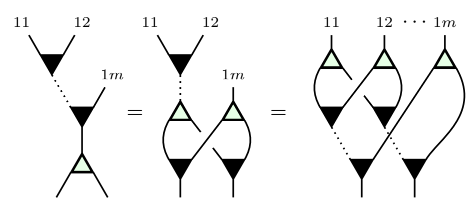
Now the right side of relation **(12)** appears in the diagram m times with a_{1j} and a_{2j} in place of b and c. Relation **(12)** can therefore be used to simplify to the multiplications $a_{1j} + a_{2j}$.



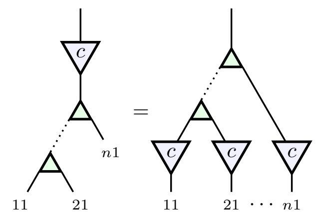
The simplification removes one instance of Δ from each of the m clusters of Δ and m instances of $+$ from the large addition cluster. There will remain $(m-1)+(m-1)+(1)-(m) = m-1$ instances of $+$, which is the correct number for the cluster. I.e. the composition has been reduced to standard form.
The argument is vastly simpler if $S = (0 \oplus \cdots \oplus 0)$. In that case relation **(1)** deletes the addition and one of the 0 morphisms, and S is still in the same form.



- $\circ\boldsymbol{\Delta}$
  The argument for $S \circ (\Delta \oplus 1^{m-2})$ is dual to the above argument, using the light relations **(4)**, **(5)** and **(6)** instead of the dark relations **(1)**, **(2)** and **(3)**.
- $\boldsymbol{\Delta}\circ$
  For $(\Delta \oplus 1^{n-1}) \circ S$, relation **(7)** can be used iteratively to "float" the Δ layer above each of the two $+$ clusters formed by the first iteration.

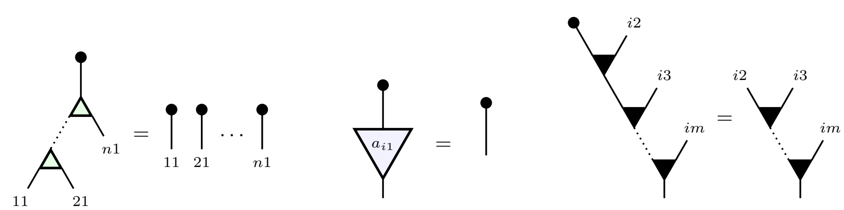


  Each of these instances of Δ can pass through the multiplication layer to Δ clusters using relation **(17)**.
  As before, we consider the subcase $S = (0 \oplus \cdots \oplus 0)$ separately. Relation **(8)** removes the duplication and creates a new zero, so S remains in the same form.
- $\circ +$
  For $S(+ \oplus 1^{m-1})$, the argument is dual to the previous one: relation **(7)** is used to "float" the additions down, relation **(15)** sends the additions through the multiplications, and relation **(9)** removes the addition and creates a new deletion in the subcase $S = (! \oplus \cdots \oplus !)$.
- $\circ \boldsymbol{c}$
  We can iterate relation **(17)** when a multiplication is composed on top, as in $S(c \oplus 1^{m-1})$.

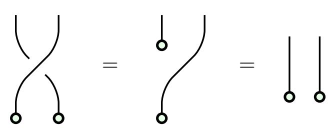


The double multiplications in the multiplication layer reduce to a single multiplication via relation **(11)**, $c \circ a_{ij} = ca_{ij}$, which leaves the diagram in standard form. The composition does nothing when $S = (! \oplus \cdots \oplus !)$, due to relation **(18)**.

- $\boldsymbol{c}\circ$
  A dual argument can be made for $(c \oplus 1^{n-1}) \circ S$ using relations **(15)**, **(11)** and **(16)**.
- $\circ\mathbf{0}$
  For $S(0 \oplus 1^{m-1})$, relations **(8)** and **(16)** eradicate the first Δ cluster and all the multiplications incident to it, leaving behind n zeros. Relation **(1)** erases each of these zeros along with one addition per addition cluster, leaving a diagram that is in standard form.



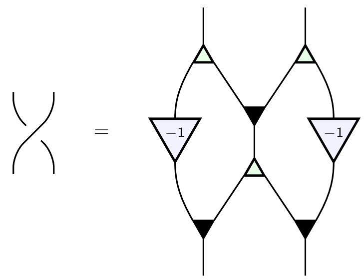
When $S = (! \oplus \cdots \oplus !)$, the zero annihilates one of the deletions via relation **(10)**.
- **!**$\circ$
  A dual argument erases the indicated output for the composition $(! \oplus 1^{n-1}) \circ S$ using relations **(9)**, **(18)**, and **(4)**. Again, relation **(10)** annihilates the deletion and one of the zeros if $S = (0 \oplus \cdots \oplus 0)$.
- $\boldsymbol{B}\circ$
  Since this category of string diagrams is symmetric monoidal, an appended braiding will naturally commute with the addition cluster morphisms. The principle that only the topology matters means the composition $(B \oplus 1^{n-2}) \circ S$ is in standard form. Braiding will similarly commute with deletion morphisms.



- $\circ\boldsymbol{B}$
  Composing with B on the top, braiding commutes with duplication, multiplication and zero, so $S \circ (B \oplus 1^{m-2})$ almost trivially comes into standard form.

□

An interesting exercise is to use these relations to derive a relation that expresses the braiding in terms of other basic morphisms. One example of such a relation appeared in Section 2. Here is another:



With a few more relations, $\mathrm{FinVect}_k$ can be presented as merely a monoidal category. Lafont [16] did this in the special case where k is the field with two elements.

4. A presentation of FinRel_k

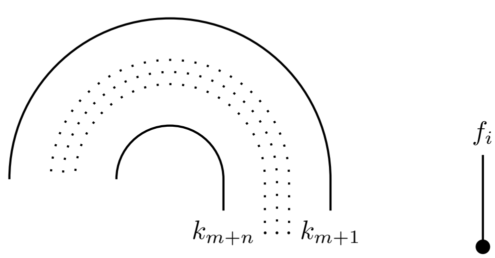
Now we give a presentation for the symmetric monoidal category FinRel_k. As we did in the previous section for $\mathrm{FinVect}_k$, we work in a strict version of the symmetric monoidal category FinRel_k.

Lemma 3. *For any field k, the object k together with the morphisms:*

- *addition* $+\colon k \oplus k \nrightarrow k$
- *zero* $0\colon \{0\} \nrightarrow k$
- *duplication* $\Delta\colon k \nrightarrow k \oplus k$
- *deletion* $!\colon k \nrightarrow \{0\}$
- *multiplication* $c\colon k \nrightarrow k$ *for any* $c \in k$
- *cup* $\cup\colon k \oplus k \nrightarrow \{0\}$
- *cap* $\cap\colon \{0\} \nrightarrow k \oplus k$

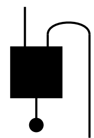
generate FinRel_k*, the category of finite-dimensional vector spaces over k and linear relations, as a symmetric monoidal category.*

Proof. A morphism of FinRel_k, $R\colon k^m \nrightarrow k^n$ is a subspace of $k^m \oplus k^n \cong k^{m+n}$. It can be expressed as a system of k-linear equations in k^{m+n}. Lemma 1 tells us any number of arbitrary k-linear combinations of the inputs may be generated. Any k-linear equation of those inputs can be formed by setting such a k-linear combination equal to zero. In particular, if caps are placed on each of the outputs to make them inputs and all the k-linear combinations are set equal to zero, any k-linear system of equations of the inputs and outputs can be formed. Expressed in terms of string diagrams,



The left diagram turns the n outputs into inputs by placing caps on all of them. The morphism zero gives the k-linear combination zero, so an arbitrary k-linear combination in k^{m+n} is set equal to zero ($f_i = 0$) via the cozero morphism. These elements can be combined with Lemma 1 to express any system of k-linear equations in k^{m+n}. □

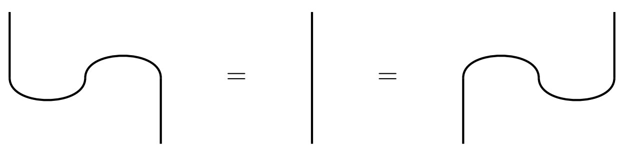
Putting these elements together, taking the $\mathrm{FinVect}_k$ portion as a black box and drawing a single string to denote zero or more copies of k, the picture is fairly simple:



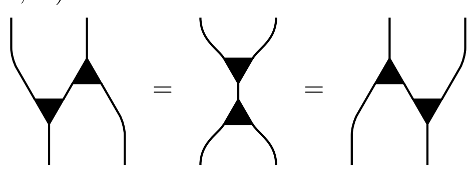
To obtain a presentation of FinRel_k as a symmetric monoidal category, we need to find enough relations obeyed by the generating morphisms listed in Lemma 3. Relations **(1)**–**(18)** from Theorem 2 still apply, but we need more.

For convenience, in the list below we draw the adjoint of any generating morphism by rotating it by 180°. It will follow from relations (19) and (20) that the cap is the adjoint of the cup, so this convenient trick is consistent even in that case, where *a priori* there might have been an ambiguity.

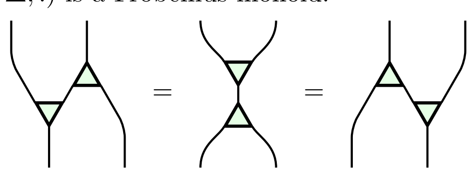
(19)–**(20)** $\cap$ and $\cup$ obey the zigzag relations, and thus give a †-compact category:



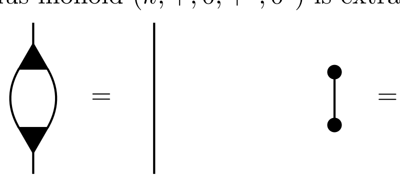
(21)–**(22)** $(k, +, 0, +^\dagger, 0^\dagger)$ is a Frobenius monoid:



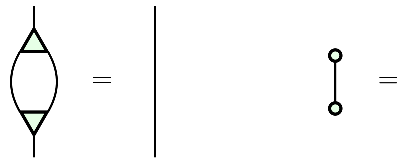
(23)–**(24)** $(k, \Delta^\dagger, !^\dagger, \Delta, !)$ is a Frobenius monoid:



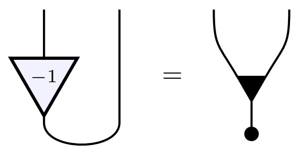
(25)–**(26)** The Frobenius monoid $(k, +, 0, +^\dagger, 0^\dagger)$ is extra-special:



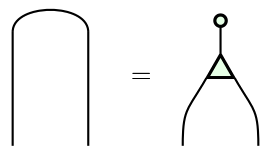
(27)–(28) The Frobenius monoid $(k, \Delta^\dagger, !^\dagger, \Delta, !)$ is extra-special:



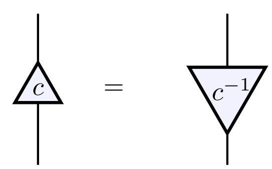
(29) $\cup$ with a factor of -1 inserted can be expressed in terms of $+$ and 0:



(30) $\cap$ can be expressed in terms of Δ and !:

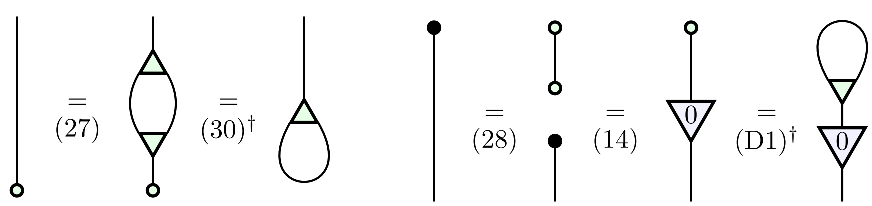


(31) For any $c \in k$ with $c \neq 0$, scalar multiplication by c^{-1} is the adjoint of scalar multiplication by c:



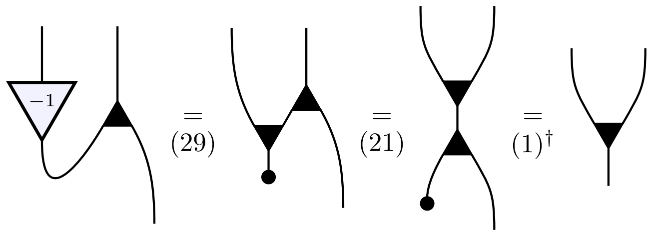
Some curious identities can be derived from relations **(1)–(31)**, beyond those already arising from **(1)–(18)**. For example:

(D1)–(D2) Deletion and zero can be expressed in terms of other generating morphisms:

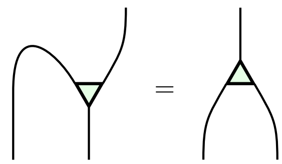


This does not diminish the role of deletion and zero. Indeed, regarding these generating morphisms as superfluous buries some of the structure of FinRel_k.

(D3) Addition can be expressed in terms of coaddition and scalar multiplication by -1, and the cup:

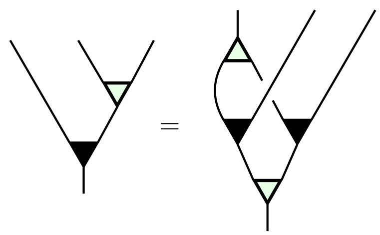


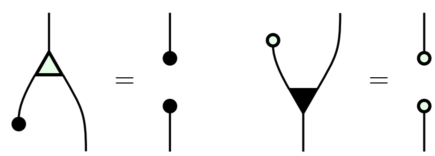
(D4) Duplication can be expressed in terms of coduplication and the cap:



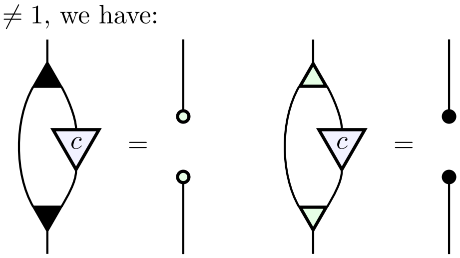
where the proof is similar to that of **(D3)**.

(D5)–**(D7)** We can reformulate the bimonoid relations (7)–(9) using adjoints:





(D8)–**(D9)** When $c \neq 1$, we have:



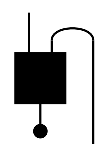
We leave the derivation of **(D5)–(D9)** as exercises for the reader.

Next we show that relations **(1)**–**(31)** are enough to give a presentation of FinRel_k as a symmetric monoidal category. As before, we do this by giving a standard form that any morphism can be written in and use induction to show that an arbitrary diagram can be rewritten in its standard form using the given relations.

Theorem 4. *The symmetric monoidal category* FinRel_k *is presented by the object* k*, the morphisms given in Lemma 3, and relations* **(1)**–**(31)** *as listed above.*

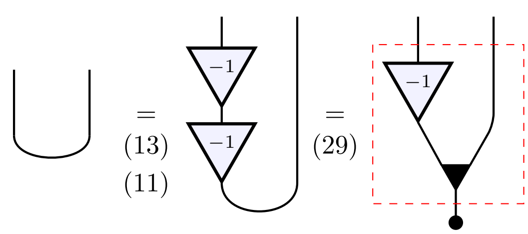
Proof. We prove this theorem by using the relations **(1)**–**(31)** to put any string diagram built from the generating morphisms and braiding into a standard form, so that any two string diagrams corresponding to the same morphism in FinRel_k have the same standard form.

As before, we induct on the number of **basic morphisms** involved in a string diagram, where the basic morphisms are the generating morphisms together with the braiding. If we let $R\colon k^m \nrightarrow k^n$ be a morphism in FinRel_k, we can build a string diagram S for R as in Lemma 3. Each output of S is capped, and, together with the inputs of S, form inputs for a $\mathrm{FinVect}_k$ block, T. For some $r \leq m+n$, there are r outputs of T–linear combinations of the $m+n$ inputs–each set equal to zero via $(0^\dagger)^r$. When T is in standard form for $\mathrm{FinVect}_k$, we say S is in **prestandard form**, and can be depicted as follows:

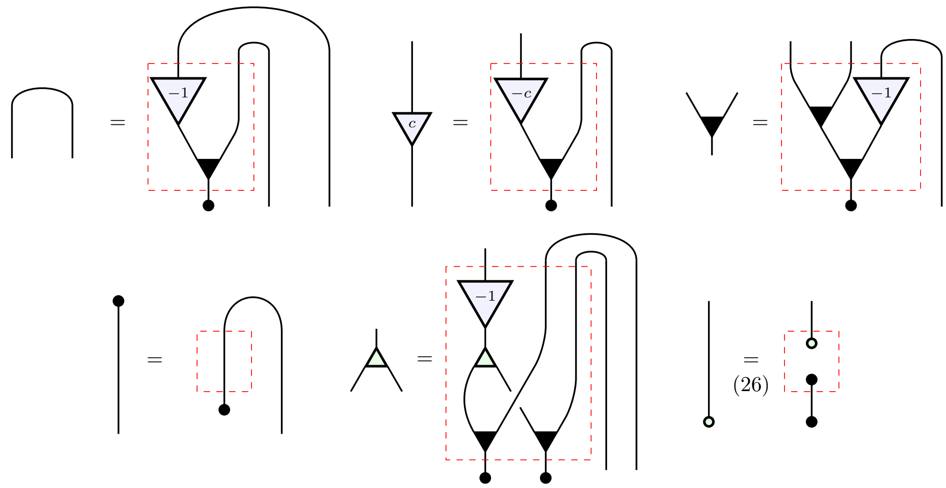


While the linear subspace of k^{m+n} defined by R is determined by a system of r linear equations, the converse is not true, meaning there may be multiple prestandard string diagrams for a single morphism R. The second stage of this proof collapses all the prestandard forms into a standard form using some basic linear algebra. The standard form will correspond to when the matrix representation of T is written in row-reduced echelon form. For this stage it will suffice to show all the elementary row operations correspond to relations that hold between diagrams. By Theorem 2, an arbitrary $\mathrm{FinVect}_k$ block can be rewritten in its standard form, so the $\mathrm{FinVect}_k$ blocks here need not be demonstrated in their standard form.
When there is one basic morphism, there are eight cases to consider, one per basic morphism. In each of these basic cases, the block of the diagram equivalent to a morphism in $\mathrm{FinVect}_k$ is denoted by a dashed rectangle. We first consider $\cup$.

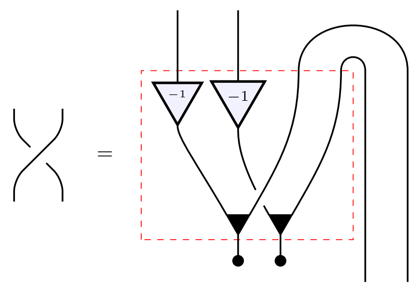
(D10)



Capping each of the inputs turns this into the standard form of $\cap$. Aside from deletion, the remaining generating morphisms can be formed by introducing a zigzag at each output and rewriting the resulting cups as above. The standard forms for 0 and ! have simpler expressions.

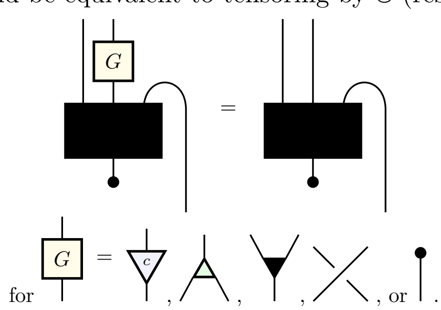


Braiding is two copies of multiplication by 1 that have been braided together.



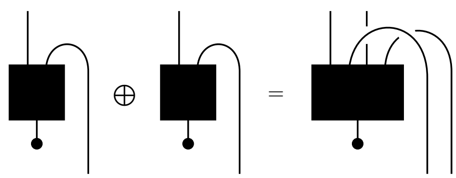
Assuming any string diagram with j basic morphisms can be written in prestandard form, we show an arbitrary diagram with $j+1$ basic morphisms can be written in prestandard form as well. Let S be a string diagram on j basic morphisms, rewritten into prestandard form, with a maximal $\mathrm{FinVect}_k$ subdiagram T. Several cases are considered: those putting a basic morphism above S, beside S, and below S.

- **$S \circ G$ for a basic morphism $G \neq \cap$**
  If a diagram G is composed above S, G can combine with T to make a larger $\mathrm{FinVect}_k$ subdiagram if G is c, Δ, $+$, B, or 0, as these are morphisms in $\mathrm{FinVect}_k$. The generating morphisms $\cap$, $\cup$ and ! are not on this list, though a composition with $\cup$ (resp. !) would be equivalent to tensoring by $\cup$ (resp. !).

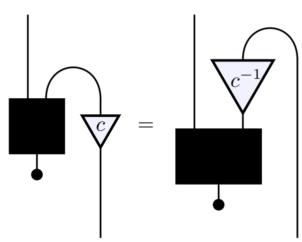


Putting these morphisms on top of S reduces to performing those compositions on T. The maximal $\mathrm{FinVect}_k$ subdiagram now includes T and G, with S unchanged outside the $\mathrm{FinVect}_k$ block.

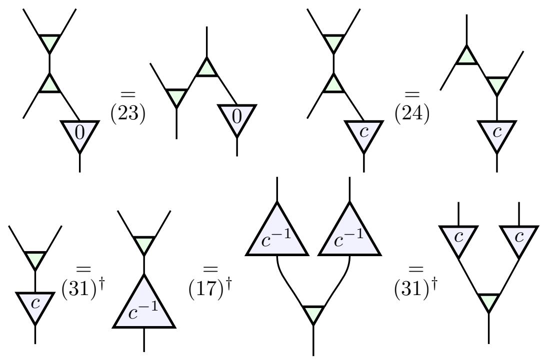
- $\boldsymbol{B \circ S}$
  B commutes with caps because the category is symmetric monoidal, so capping the braiding is equivalent to putting the braiding on top of T. B is "absorbed" into T, just as in the $S \circ G$ case.
- $\boldsymbol{S \oplus G}$ **for any basic morphism** $\boldsymbol{G}$
  If any two prestandard string diagrams S and S' are tensored together, the result combines into one prestandard diagram. This is evident because the category of string diagrams is symmetric monoidal, and the $\mathrm{FinVect}_k$ blocks can be placed next to each other as the tensor of two $\mathrm{FinVect}_k$ blocks. These combine into a single $\mathrm{FinVect}_k$ block, and absorbing all the braidings into this block as above brings the diagram into prestandard form. Since each basic morphism can be written as a prestandard diagram, the tensor $S \oplus G$ is a special case of this.



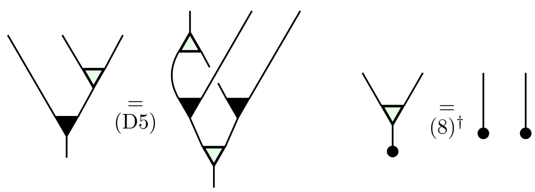
- $\boldsymbol{c \circ S}$ **for** $\boldsymbol{c \neq 0}$
  Because the outputs of S are capped, putting any morphism on the bottom of S is equivalent (via relations **(19)** and **(20)**) to putting its adjoint on top of T. Putting $c \neq 0$ below S reduces to putting c^{-1} on top of T by relation **(31)**. The case of $c = 0$ will be considered below. The other cases of adjoints of generating morphisms that need to be considered more carefully are the ones that put $\Delta^\dagger$, $+^\dagger$ and $\cap = \cup^\dagger$ on top of T.



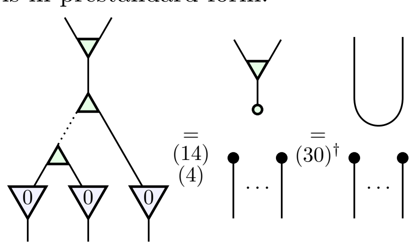
- $\boldsymbol{\Delta \circ S}$
  When putting $\Delta^\dagger$ on top of T, the idea is to make it "trickle down." If there is a nonzero multiplication incident to the Δ cluster, $\Delta^\dagger$ can slide through the Δs using relation **(23)** to the first nonzero multiplication, switching to relation **(24)**. When it encounters this c, relation **(31)** turns c into $(c^{-1})^\dagger$, relation $\mathbf{(17)}^\dagger$ allows $\Delta^\dagger$ to pass through $(c^{-1})^\dagger$. Both copies of $(c^{-1})^\dagger$ can return to being c by another application of relation **(31)**, and the $\Delta^\dagger$ moves on to the next layer.



When the codelta gets to a $+$ cluster, derived relation **(D5)** has a net effect of bringing it to the bottom of the subdiagram, as the other morphisms involved all belong to $\mathrm{FinVect}_k$. This allows the process to be repeated on the next addition until $\Delta^\dagger$ reaches the bottom of the $+$ cluster. Once there, codelta interacts with the cozero layer below T; relation **(8)**† reduces it to a pair of cozeros.

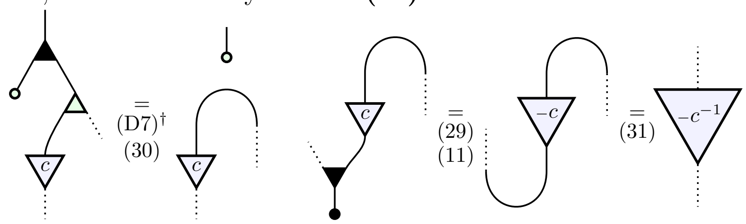


If all the multiplications incident to the Δ cluster are by 0, rather than trickling down, $\Delta^\dagger$ composes with ! (due to relation **(14)**), which gives $\cup$ by relation **(30)**†. By the zigzag identities, this cup becomes a cap that is tensored with a subdiagram of S that is in prestandard form.



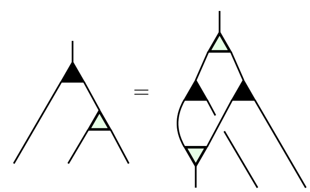
- **$+ \circ S$**
  There is a similar trickle down argument for $+^\dagger$. First rewriting all multiplications by zero via relation **(14)**, the two Δ clusters incident to the coaddition can either reduce to Δ clusters that are incident only to nonzero multiplications or reduce to a single deletion, as above, if none of the incident multiplications were nonzero. There are three cases of what can happen from here.
  - **Both Δ clusters were incident to only zero-multiplications**
    In the first case, as above, the Δ clusters will reduce to ! incident to the outputs of $+^\dagger$. Relations **(D7)** and **(28)** delete the coaddition.
  - **One Δ cluster was incident to only zero-multiplications**
    Without loss of generality, the ! incident to $+^\dagger$ is on the left. Relation **(D7)** replaces ! and $+^\dagger$ with $!^\dagger \circ !$, and relation **(30)** replaces Δ and $!^\dagger$ with a cap. The Δ was – and the cap is – incident to some multiplication by $c \neq 0$. Without

loss of generality, c is incident to the bottom addition in the cluster. Relation **(29)** replaces the addition and cozero with a cup and multiplication by -1, which combines with c by relation **(11)**. The cup and cap turn $-c$ around to its adjoint, which is $-c^{-1}$ by relation **(31)**.

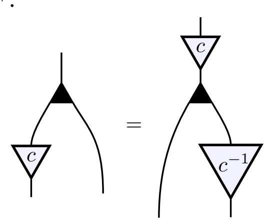


An addition cluster is above $-c^{-1}$ and a duplication cluster is below, but because those clusters are not otherwise connected to each other, there is a vertical arrangement of the morphisms in the $\mathrm{FinVect}_k$ block of the string diagram such that no cups or caps are present.

– **Both $\boldsymbol{\Delta}$ clusters are incident to at least one nonzero multiplication** Using relation $\mathbf{(D5)}^\dagger$, a $+^\dagger$ will pass through one Δ at a time. A new $\Delta^\dagger$ is created each time, but this can trickle down as before.

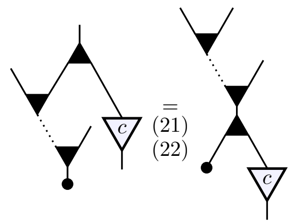


Once the $\Delta^\dagger$ trickles down, there are two possibilities for what is directly beneath each $+^\dagger$: either the same scenario will recur with a Δ connected to one or both outputs, which can only happen finitely many times, or two nonzero multiplications will be below the $+^\dagger$. A multiplication by any unit in k, $c \neq 0$, can move through a coaddition by inserting cc^{-1} on the top branch and applying relation $\mathbf{(15)}^\dagger$:



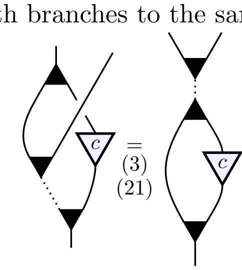
This allows one of the outputs of the coaddition to connect directly to a $+$ cluster.

∗ **If both branches go to different + clusters**, Frobenius relations **(21)**–**(22)** slide the $+^\dagger$ down the $+$ cluster on one side until it gets to the end of that cluster.

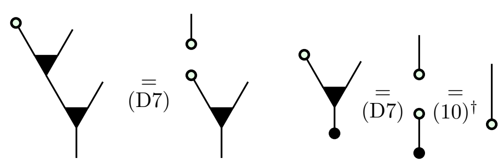


The only morphisms added to the $\mathrm{FinVect}_k$ block that are not from $\mathrm{FinVect}_k$ were the coaddition and the cozero. Since these reduce to an identity morphism string by relation **(1)**†, the $\mathrm{FinVect}_k$ block is truly a $\mathrm{FinVect}_k$ block again.

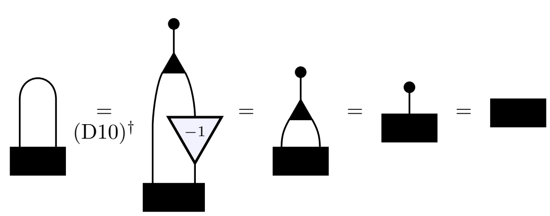
$*$ **If both branches go to the same + cluster**, relation **(3)** and the Frobenius relation **(21)** take both branches to the same addition.



Depending on whether the remaining multiplication is by 1, either relation **(25)** reduces the coaddition and the given addition to an identity string or relation **(D8)** applies. In the former case we are done, and in the latter case relations **(D7)** and **(10)**† remove the $!^\dagger$ introduced by applying relation (D8).



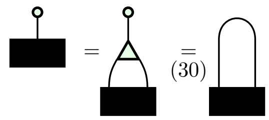
- **$\cup \circ S$ and $S \circ \cap$**
  Composing with a cup below S is equivalent to composing with cap above T, since $\cap = \cup^\dagger$. Using relation **(D10)**†, this cap can be replaced by multiplication by -1, coaddition, and zero. By the arguments above, -1, $+^\dagger$, and 0 can each be absorbed into the $\mathrm{FinVect}_k$ block.



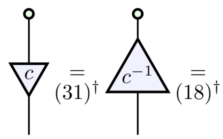
The compositions with zero and multiplication by -1 expand the $\mathrm{FinVect}_k$ block, thus have no effect on whether the diagram can be written in prestandard form.

- **$! \circ S$**
  When composing $!^\dagger$ above T, two possibilities arise, depending on whether there is a layer of Δs in the $\mathrm{FinVect}_k$ block. If there is such a layer, relation **(30)**

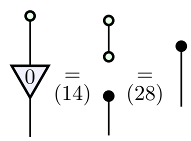
combines the $!^\dagger$ with a Δ, making a cap on top of T. As we have just seen, this can be rewritten in prestandard form.



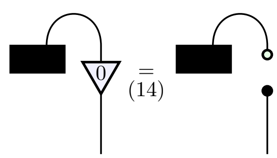
If no layer of Δs exists, relations **(31)**† and **(18)**† pass the codeletion through a nonzero multiplication. Then relations **(D7)** and **(10)**† can be used to remove $!^\dagger$, as we have already seen. This leaves only the basic morphisms of $\mathrm{FinVect}_k$ within the $\mathrm{FinVect}_k$ block.



If the multiplication is $c = 0$, relation **(14)** converts $c = 0$ to $0 \circ !$, allowing relation **(28)** to remove the $!^\dagger$, with the same conclusion.



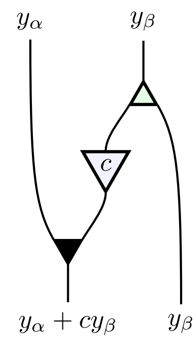
- $\boldsymbol{c \circ S}$ **for** $\boldsymbol{c = 0}$
  Composing with multiplication by $c = 0$ below S is equivalent to composing with codeletion, followed by tensoring with zero. Codeletion is the $! \circ S$ case, and zero can be written in a prestandard form, so this reduces to tensoring two diagrams that are in prestandard form.



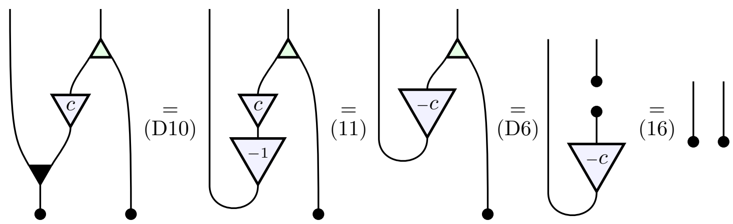
Finally, we need to show the prestandard forms can be rewritten in standard form. We need to show what elementary row operations look like in terms of string diagrams. We also need to show for an arbitrary prestandard string diagram S with $\mathrm{FinVect}_k$ block T that if T is replaced with T', the diagram where an elementary row operation has been performed on T, the resulting diagram S' can be built from S using relations **(1)**–**(31)**.

Because the ith output of a $\mathrm{FinVect}_k$ diagram is a linear combinations of the inputs, with the coefficients coming from the ith row of its matrix, rows of the matrix correspond to outputs of the $\mathrm{FinVect}_k$ block. Because of this, the row operation subdiagrams in S' will have $0^\dagger$s immediately beneath them. Showing S' can be built from S reduces to showing composition of row operations with $0^\dagger$s builds the same number of $0^\dagger$s.

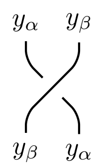
- Add a multiple c of one row to another row:
  If we want to add a multiple of the β row to the α row, we need a map $(y_\alpha, y_\beta) \mapsto (y_\alpha + c y_\beta, y_\beta)$. By the naturality of the braiding in a symmetric monoidal category, we can ignore any intermediate outputs:



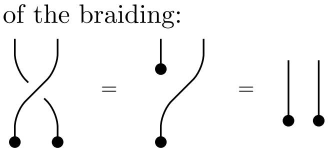
When two cozeros are composed on the bottom of this diagram, the result is two cozeros:



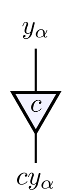
- Swap rows:
  If we want to swap the β row with the α row, we need a map $(y_\alpha, y_\beta) \mapsto (y_\beta, y_\alpha)$, which is the braiding of two outputs. Again, intermediate outputs may be ignored:



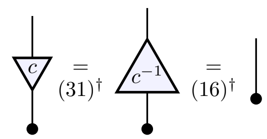
  When two cozeros are composed at the bottom of this diagram, the cut strings untwist by the naturality of the braiding:



- Multiply a row by $c \neq 0$:
  The third row operation is multiplying an arbitrary row by a unit, but since k is a field, that means any $c \neq 0$. This is just the multiplication map on one of the outputs:



  Because c is a unit, $c^{-1} \in k$, so the multiplication by c can be replaced by the adjoint of multiplication by c^{-1}.



□

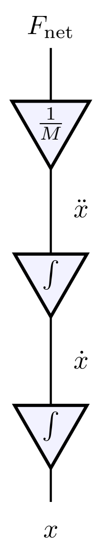
5. An example

A famous example in control theory is the 'inverted pendulum': an upside-down pendulum on a cart [10]. The pendulum naturally tends to fall over, but we can stabilize it by setting up a feedback loop where we observe its position and move the cart back and forth in a suitable way based on this observation. Without introducing this feedback loop, let us see how signal-flow diagrams can be used to describe the pendulum and the cart. We shall see that the diagram for a system made of parts is built from the diagrams for the parts, not merely by composing and tensoring, but also with the help of duplication and coduplication, which give additional ways to set variables equal to one another.

Suppose the cart has mass M and can only move back and forth in one direction, so its position is described by a function $x(t)$. If it is acted on by a total force $F_{\mathrm{net}}(t)$ then Newton's second law says

$$F_{\mathrm{net}}(t) = M\ddot{x}(t).$$

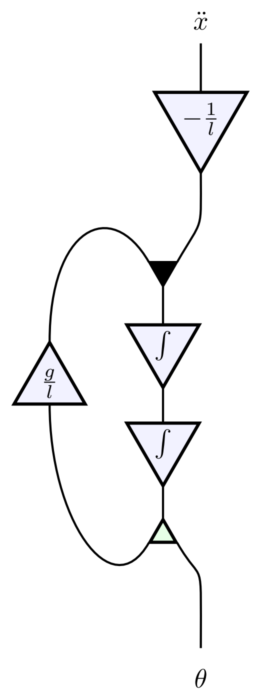
We can thus write a signal-flow diagram with the force as input and the cart's position as output:



The inverted pendulum is a rod of length ℓ with a mass m at its end, mounted on the cart and only able to swing back and forth in one direction, parallel to the cart's movement. If its angle from vertical, $\theta(t)$, is small, then its equation of motion is approximately linear:

$$\ell\ddot{\theta}(t) = g\theta(t) - \ddot{x}(t)$$

where g is the gravitational constant. We can turn this equation into a signal-flow diagram with $\ddot{x}$ as input and θ as output:

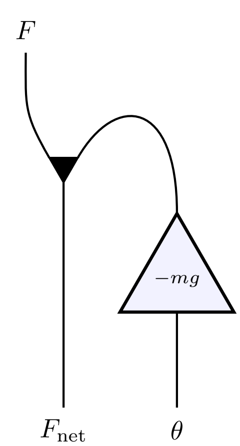


Note that this already includes a kind of feedback loop, since the pendulum's angle affects the force on the pendulum.

Finally, there is an equation describing the total force on the cart:

$$F_{\text{net}}(t) = F(t) - mg\theta(t)$$

where $F(t)$ is an externally applied force and $-mg\theta(t)$ is the force due to the pendulum. It will be useful to express this as follows:



Here we are treating θ as an output rather than an input, with the help of a cap.

The three signal-flow diagrams above describe the following linear relations:

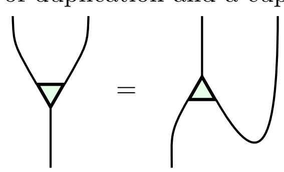
$$x = \int\int \frac{1}{M} F_{\text{net}} \tag{1}$$

$$\theta = \int\int \left(\frac{g}{l}\,\theta - \frac{1}{l}\,\ddot{x}\right) \tag{2}$$

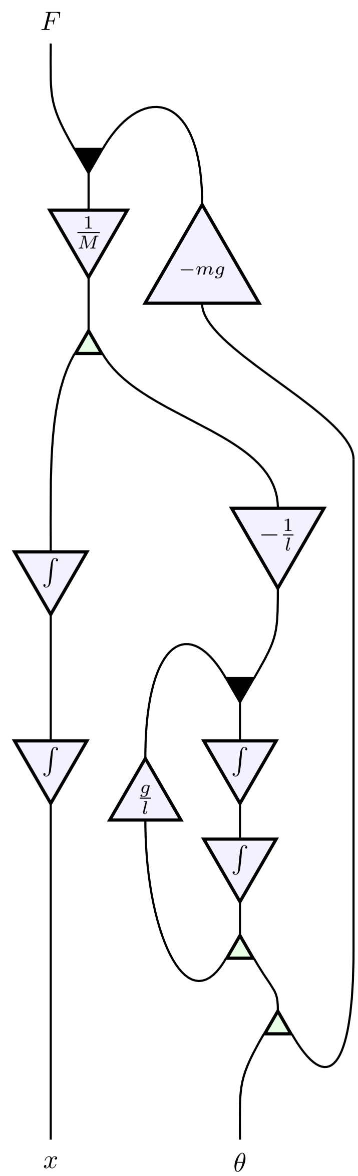
$$F_{\text{net}} + mg\theta = F \tag{3}$$

where we treat (1) as a relation with F_{net} as input and x as output, (2) as a relation with $\ddot{x}$ as input and θ as output, and (3) as a relation with F as input and (F_{net}, θ) as output.

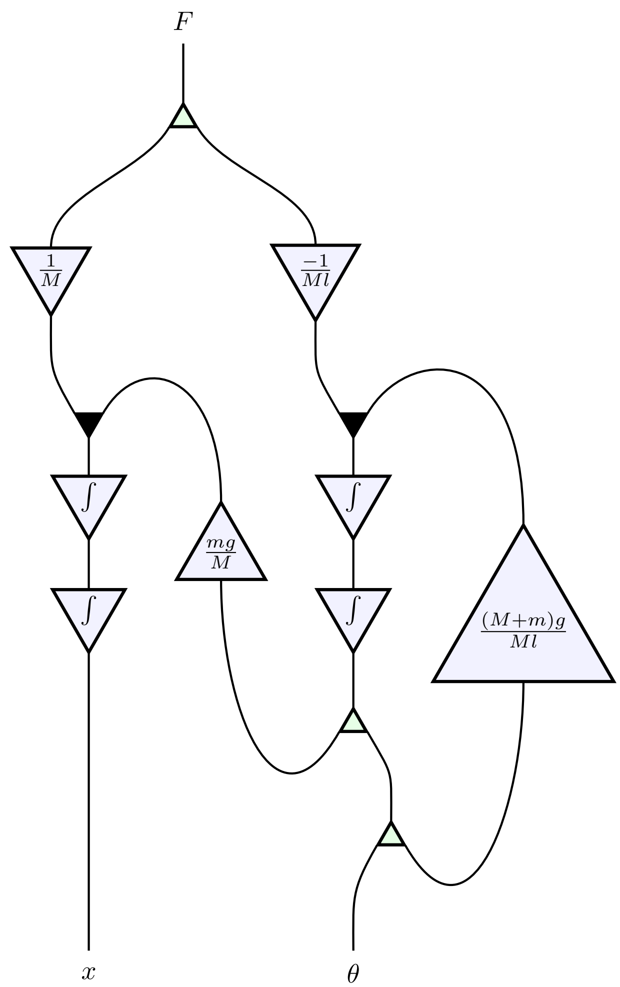
To understand how the external force affects the position of the cart and the angle of the pendulum, we wish to combine all three diagrams to form a signal-flow diagram that has the external force F as input and the pair (x, θ) as output. This is not just a simple matter of composing and tensoring the three diagrams. We can take F_{net}, which is an output of (3), and use it as an input for (1). But we also need to duplicate $\ddot{x}$, which appears as an intermediate variable in (1) since $\ddot{x} = \frac{1}{M} F_{\text{net}}$, and use it as an input for (2). Finally, we need to take the variable θ, which appears as an output of both (2) and (3), and identify the two copies of this variable using coduplication. Following traditional engineering practice, we shall write coduplication in terms of duplication and a cup, as follows:



The result is this signal-flow diagram:



This is not the signal-flow diagram for the inverted pendulum that one sees in Friedland's textbook on control theory [10]. We leave it as an exercise to the reader to rewrite the above diagram using the rules given in this paper, obtaining Friedland's diagram:



As a start, one can use Theorem 4 to prove that it is indeed possible to do this rewriting. To do this, simply check that both signal-flow diagrams define the same linear relation. The proof of the theorem gives a method to actually do the rewriting—but not necessarily the fastest method.

6. Conclusions

We conclude with some remarks aimed at setting our work in context. In particular, we would like to compare it to some other recent papers. On April 30th, 2014, after most of this paper was written, Sobociński told the first author about some closely related papers that he wrote with Bonchi and Zanasi [4, 5]. These provide interesting characterizations of symmetric monoidal categories equivalent to $\mathrm{FinVect}_k$ and FinRel_k. Later, while this paper was being refereed, Wadsley and Woods [22] generalized the first of these results to the case where k is any commutative rig. We discuss Wadsley and Woods' work first, since doing so makes the exposition simpler.

A particularly tractable sort of symmetric monoidal category is a PROP: that is, a strict symmetric monoidal category where the objects are natural numbers and the tensor product of objects is given by ordinary addition. The symmetric monoidal category $\mathrm{FinVect}_k$ is equivalent to the PROP $\mathrm{Mat}(k)$, where a morphism $f\colon m \to n$ is an $n \times m$ matrix with entries in k, composition of morphisms is given by matrix multiplication, and the tensor product of morphisms is the direct sum of matrices.

Wadsley and Woods gave an elegant description of the algebras of $\mathrm{Mat}(k)$. Suppose C is a PROP and D is a strict symmetric monoidal category. Then the **category of algebras** of C in D is the category of strict symmetric monoidal functors $F\colon C \to D$ and natural transformations between these. If for every choice of D the category of algebras of C in D is equivalent to the category of algebraic structures of some kind in D, we say C is the PROP for structures of that kind.

In this language, Wadsley and Woods proved that $\mathrm{Mat}(k)$ is the PROP for 'bicommutative bimonoids over k'. To understand this, first note that for any bicommutative bimonoid A in D, the bimonoid endomorphisms of A can be added and composed, giving a rig $\mathrm{End}(A)$. A bicommutative bimonoid **over *k*** in D is one equipped with a rig homomorphism $\Phi_A\colon k \to \mathrm{End}(A)$. Bicommutative bimonoids over k form a category where a morphism $f\colon A \to B$ is a bimonoid homomorphism compatible with this extra structure, meaning that for each $c \in k$ the square

$$\begin{tikzcd}
A \arrow[r, "\Phi_A(c)"] \arrow[d, "f"'] & A \arrow[d, "f"] \\
B \arrow[r, "\Phi_B(c)"'] & B
\end{tikzcd}$$

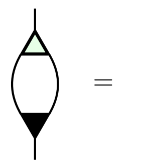
commutes. Wadsley and Woods proved that this category is equivalent to the category of algebras of $\mathrm{Mat}(k)$ in D.

This result amounts to a succinct restatement of Theorem 2, though technically the result is a bit different, and the style of proof much more so. The fact that an algebra of $\mathrm{Mat}(k)$ is a bicommutative bimonoid is equivalent to our relations **(1)**–**(10)**. The fact that $\Phi_A(c)$ is a bimonoid homomorphism for all $c \in k$ is equivalent to relations **(15)**–**(18)**, and the fact that Φ is a rig homomorphism is equivalent to relations **(11)**–**(14)**.

Even better, Wadsley and Woods showed that $\mathrm{Mat}(k)$ is the PROP for bicommutative bimonoids over k whenever k is a commutative rig. Subtraction and division are not required to define the PROP $\mathrm{Mat}(k)$, nor are they relevant to the definition of bicommutative bimonoids over k. Working with commutative rigs is not just generalization for the sake of generalization: it clarifies some interesting facts.

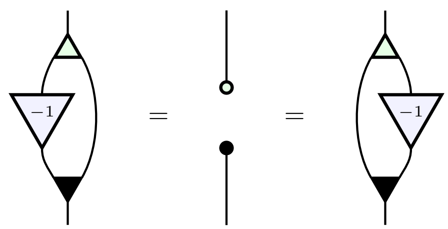
For example, the commutative rig of natural numbers gives a PROP $\mathrm{Mat}(\mathbb{N})$. This is equivalent to the symmetric monoidal category where morphisms are isomorphism classes of spans of finite sets, with disjoint union as the tensor product. Lack [15, Ex. 5.4] had already shown that this is the PROP for bicommutative bimonoids. But this also follows from the result of Wadsley and Woods, since every bicommutative bimonoid A is automatically equipped with a unique rig homomorphism $\Phi_A\colon \mathbb{N} \to \mathrm{End}(A)$.

Similarly, the commutative rig of booleans $\mathbb{B} = \{F, T\}$, with 'or' as addition and 'and' as multiplication, gives a PROP $\mathrm{Mat}(\mathbb{B})$. This is equivalent to the symmetric monoidal category where morphisms are relations between finite sets, with disjoint union as the tensor product. Mimram [18, Thm. 16] had already shown this is the PROP for **special** bicommutative bimonoids, meaning those where comultiplication followed by multiplication is the identity:



But again, this follows from the general result of Wadsley and Woods.

Finally, taking the commutative ring of integers $\mathbb{Z}$, Wadsley and Woods showed that $\mathrm{Mat}(\mathbb{Z})$ is the PROP for bicommutative Hopf monoids. The key here is that scalar multiplication by -1 obeys the axioms for an antipode, namely:



More generally, whenever k is a commutative ring, the presence of $-1 \in k$ guarantees that a bimonoid over k is automatically a Hopf monoid over k. So, when k is a commutative ring, Wadsley and Woods' result implies that $\mathrm{Mat}(k)$ is the PROP for Hopf monoids over k.

Earlier, Bonchi, Sobociński and Zanasi gave an elegant and very different proof that $\mathrm{Mat}(R)$ is the PROP for Hopf monoids over R when R is a principal ideal domain [4, Prop. 3.7]. The advantage of their argument is that they build up the PROP for Hopf monoids over R from smaller pieces, using some ideas developed by [15].

These authors also described a PROP that is equivalent to FinRel_k as a symmetric monoidal category whenever k is a field. In this PROP, which they call $\mathbb{SV}_k$, a morphism $f\colon m \to n$ is a linear relation from k^m to k^n. They proved that $\mathbb{SV}_k$ is a pushout in the category of PROPs and strict symmetric monoidal functors:

$$\begin{tikzcd}
\mathrm{Mat}(R) + \mathrm{Mat}(R)^{\mathrm{op}} \arrow[r] \arrow[d] & \mathrm{Span}(\mathrm{Mat}(R)) \arrow[d] \\
\mathrm{Cospan}(\mathrm{Mat}(R)) \arrow[r] & \mathbb{SV}_k
\end{tikzcd}$$

This pushout square requires a bit of explanation. Here R is any principal ideal domain whose field of fractions is k. For example, we could take $R = k$, though Bonchi, Sobociński and Zanasi are more interested in the example where $R = \mathbb{R}[s]$ and $k = \mathbb{R}(s)$. A morphism in $\mathrm{Span}(\mathrm{Mat}(R))$ is an isomorphism class of spans in

Mat(R). There is a covariant functor

$$\begin{array}{ccc} \mathrm{Mat}(R) & \to & \mathrm{Span}(\mathrm{Mat}(R)) \\ m \xrightarrow{f} n & \mapsto & m \xleftarrow{1} m \xrightarrow{f} n \end{array}$$

and also a contravariant functor

$$\begin{array}{ccc} \mathrm{Mat}(R) & \to & \mathrm{Span}(\mathrm{Mat}(R)) \\ m \xrightarrow{f} n & \mapsto & n \xleftarrow{f} m \xrightarrow{1} m. \end{array}$$

Putting these together we get the functor from $\mathrm{Mat}(R)+\mathrm{Mat}(R)^{\mathrm{op}}$ to Span(Mat(R)) that gives the top edge of the square. Similarly, a morphism in Cospan(Mat(R)) is an isomorphism class of cospans in Mat(R), and we have both a covariant functor

$$\begin{array}{ccc} \mathrm{Mat}(R) & \to & \mathrm{Cospan}(\mathrm{Mat}(R)) \\ m \xrightarrow{f} n & \mapsto & m \xrightarrow{f} n \xleftarrow{1} n \end{array}$$

and a contravariant functor

$$\begin{array}{ccc} \mathrm{Mat}(R) & \to & \mathrm{Cospan}(\mathrm{Mat}(R)) \\ m \xrightarrow{f} n & \mapsto & n \xrightarrow{1} n \xleftarrow{f} m. \end{array}$$

Putting these together we get the functor from $\mathrm{Mat}(R)+\mathrm{Mat}(R)^{\mathrm{op}}$ to Cospan(Mat(R)) that gives the left edge of the square.

Bonchi, Sobociński and Zanasi analyze this pushout square in detail, giving explicit presentations for each of the PROPs involved, all based on their presentation of Mat(R). The upshot is a presentation of $\mathbb{SV}_k$ which is very similar to our presentation of the equivalent symmetric monoidal category FinRel_k. Their methods allow them to avoid many, though not all, of the lengthy arguments that involve putting morphisms in 'normal form'.

Acknowledgements. We thank Jamie Vicary for pointing out the relevance of the ZX calculus when the first author gave a talk on this material at Oxford in February 2014 [2]. Discussions with Brendan Fong have also been very useful.

References

[1] Samson Abramsky and Bob Coecke, A categorical semantics of quantum protocols, *Proceedings of the 19th IEEE Conference on Logic in Computer Science (LiCS'04)*, IEEE Computer Science Press, 2004, pp. 415–425. Also available as arXiv:quant-ph/0402130. (Referred to on page 10.)

[2] John Baez, Network theory I: electrical circuits and signal-flow graphs, lecture at the Department of Computer Science, University of Oxford, February 25, 2014. Slides and video available at http://math.ucr.edu/home/baez/networks_oxford/. (Referred to on page 41.)

[3] John Baez and Mike Stay, Physics, topology, logic and computation: a Rosetta Stone, in *New Structures for Physics*, ed. Bob Coecke, *Lecture Notes in Physics* vol. 813, Springer, Berlin, 2011, pp. 95–172. Also available as arXiv:0903.0340. (Referred to on page 5.)

[4] Filippo Bonchi, Paweł Sobociński and Fabio Zanasi, Interacting Hopf algebras. Available as arXiv:1403.7048. (Referred to on page 2, 38, 40.)

[5] Filippo Bonchi, Paweł Sobociński and Fabio Zanasi, A categorical semantics of signal flow graphs, in *CONCUR 2014–Concurrency Theory*, eds. P. Baldan and D. Gorla, *Lecture Notes in Computer Science* vol. 8704, Springer, Berlin, 2014, pp. 435–450. Also available at http://users.ecs.soton.ac.uk/ps/papers/sfg.pdf. (Referred to on page 2, 38.)

[6] Aurelio Carboni and Robert F. C. Walters, Cartesian bicategories I, *J. Pure Appl. Alg.* **49** (1987), 11–32. (Referred to on page 11.)

[7] Bob Coecke and Ross Duncan, Interacting quantum observables: categorical algebra and diagrammatics, *New J. Phys.* **13** (2011), 043016. Also available as arXiv:0906.4725. (Referred to on page 12.)

[8] Bob Coecke and Eric Oliver Paquette, Categories for the practising physicist, in *New Structures for Physics*, ed. Bob Coecke, *Lecture Notes in Physics* vol. 813, Springer, Berlin, 2011, pp. 173–286. Also available as arXiv:0905.3010. (Referred to on page 5.)

[9] Bob Coecke, Dusko Pavlovic and Jamie Vicary, A new description of orthogonal bases, *Math. Str. Comp. Sci.* **23** (2013), 555–567. Also available as arXiv:0810.0812. (Referred to on page 11.)

[10] Bernard Friedland, *Control System Design: An Introduction to State-Space Methods*, Courier Dover Publications, 2012. (Referred to on page 1, 34, 37.)

[11] André Joyal and Ross Street, The geometry of tensor calculus I, *Adv. Math.* **88** (1991), 55-113. (Referred to on page 1.)

[12] André Joyal and Ross Street, The geometry of tensor calculus II. Draft available at http://maths.mq.edu.au/∼street/GTCII.pdf. (Referred to on page 1.)

[13] Joachim Kock, *Frobenius Algebras and 2D Topological Quantum Field Theories*, Cambridge U. Press, Cambridge, 2003. Short version available at http://mat.uab.es/∼kock/TQFT/FS.pdf. (Referred to on page 11.)

[14] Joachim Kock, Remarks on the origin of the Frobenius equation, available at http://mat.uab.es/∼kock/TQFT.html#history. (Referred to on page 11.)

[15] Stephen Lack, Composing PROPs, *Th. Appl. Cat.* **13** (2004), 147–163. Available at http://www.tac.mta.ca/tac/volumes/13/9/13-09abs.html. (Referred to on page 39, 40.)

[16] Yves Lafont, Towards an algebraic theory of Boolean circuits, *J. Pure Appl. Alg.* **184** (2003), 257–310. Also available at http://iml.univ-mrs.fr/∼lafont/pub/circuits.pdf. (Referred to on page 22.)

[17] Saunders Mac Lane, Natural associativity and commutativity, *Rice U. Studies* **49** (1963), 28—46. (Referred to on page 13.)

[18] Samuel Mimram, Presentation of a game semantics for first-order propositional logic. Available as arXiv:0805.0845. (Referred to on page 40.)

[19] Robert Rosebrugh, Nicoletta Sabadini and Robert F. C. Walters, Generic commutative separable algebras and cospans of graphs, *Th. Appl. Cat.* **15** (2005), 264–277. Available at http://www.tac.mta.ca/tac/volumes/15/6/15-06abs.html. (Referred to on page 11.)

[20] Peter Selinger, Dagger compact closed categories and completely positive maps, *Elec. Notes Theor. Comp. Sci.* **170** (2007), 139–163. (Referred to on page 10.)

[21] Jamie Vicary, Categorical formulations of finite-dimensional quantum algebras, *Comm. Math. Phys.* **304** (2011), 765–796. Also available as arXiv:0805.0432. (Referred to on page 11.)

[22] Simon Wadsley and Nick Woods, PROPs for linear systems. Available as arXiv:1505.00048. (Referred to on page 2, 38.)

Department of Mathematics, University of California, Riverside CA 92521, USA, and Centre for Quantum Technologies, National University of Singapore, Singapore 117543

E-mail address: baez@math.ucr.edu

Department of Mathematics, University of California, Riverside CA 92521, USA

E-mail address: erbele@math.ucr.edu