A PRACTICAL TYPE THEORY FOR SYMMETRIC MONOIDAL CATEGORIES

MICHAEL SHULMAN

ABSTRACT. We give a natural-deduction-style type theory for symmetric monoidal categories whose judgmental structure directly represents morphisms with tensor products in their codomain as well as their domain. The syntax is inspired by Sweedler notation for coalgebras, with variables associated to types in the domain and terms associated to types in the codomain, allowing types to be treated informally like "sets with elements" subject to global linearity-like restrictions. We illustrate the usefulness of this type theory with various applications to duality, traces, Frobenius monoids, and (weak) Hopf monoids.

Contents

1. Introduction

1.1. TYPE THEORIES FOR MONOIDAL CATEGORIES. Type theories are a powerful tool for reasoning about categorical structures. This is best-known in the case of the internal language of a topos, which is a higher-order intuitionistic logic. But there are also weaker type theories that correspond to less highly-structured categories, such as regular logic for regular categories, simply typed λ-calculus for cartesian closed categories, typed algebraic

theories for categories with finite products, and so on (a good overview can be found in [Joh02, Part D]).

However, type theories seem to be only rarely used to reason about *non-cartesian monoidal* categories. Such categories are of course highly important in both pure mathematics and its applications (such as quantum mechanics, network theory, etc.), but usually they are studied using either traditional arrow-theoretic syntax or string diagram calculus [JS91, Sel11].

Type theories corresponding to non-cartesian monoidal categories do certainly exist — intuitionistic linear logic for closed symmetric monoidal categories, ordered logic for non-symmetric monoidal categories, classical linear logic for $*$-autonomous categories — but they are not widely used for reasoning about monoidal categories. I believe this is largely because their convenience for practical category-theoretic arguments does not approach that of cartesian type theories. The basic problem is that most nontrivial arguments in monoidal category theory involve tensor product objects in both the *domain* and the *codomain* of morphisms.

Existing type theories for non-cartesian monoidal categories fall into two groups, neither of which deals satisfactorily with this issue. The first is exemplified by intuitionistic linear logic, whose judgments have many variables as inputs but only a single term as output:

$$x : A, y : B, z : C \vdash f(x, g(y, z)) : D.$$

This allows the terms (such as $f(x, g(y, z))$) to intuitively "treat types as if they were sets with elements" (one of the big advantages of type-theoretic syntax). But it privileges tensor products in the domain (here the domain is semantically $A \otimes B \otimes C$) over the codomain. The only way to talk about morphisms into tensor products is to use the tensor product type constructor:

$$x : A, y : B \vdash (y, x) : B \otimes A.$$

This is asymmetric, but more importantly its term syntax is heavy and unintuitive: in the cartesian case we can use an element $p : A \times B$ by simply projecting out its components $\pi_1(p) : A$ and $\pi_2(p) : B$, but in the non-cartesian case we have to "match" it and bind both components as new variables:

$$p : A \otimes B \vdash \text{let } (x, y) := p \text{ in } (y, x) : B \otimes A.$$

The second group of non-cartesian type theories is exemplified by classical linear logic, whose judgments have multiple inputs as well as multiple outputs:

$$A, B, C \vdash D, E.$$

This eliminates the asymmetry, but at the expense of no longer having a concise and intuitive term syntax. Most commonly such type theories are presented as sequent calculi without any terms at all. Term calculi do exist (e.g. [Red93, BCST96, Ong96, CPT16]) but usually involve some kind of "covariables", which complicate the interpretation of types as

"sets with elements". Moreover, in type theories of this sort the tensor product appearing in codomains is usually a *different* one from the one appearing in domains (the "cotensor product" of a $*$-autonomous category), whereas in practical applications it happens very frequently that they are the same. The only type theories with the same tensor product appearing in domains and codomains that I know of in the literature are [Shi99, Dun06], which are sequent calculi without a real term syntax, and [Has97], which does have a term syntax but still decomposes tensor products by matching. (The referee has pointed out that [PLC17] is in a similar spirit to our work, though not directly comparable.)

The purpose of this paper is to fill this gap, by describing a type theory for symmetric monoidal categories in which:

- Judgments have multiple inputs as well as multiple outputs.

- The tensor products appearing in domains and codomains are the same.

- There is a convenient and intuitive term syntax without the need for "covariables".

- Tensor product types, though rarely needed, have a convenient term syntax involving "projections", as in the cartesian case, rather than "matches".

This type theory seems to be very convenient in practice for reasoning about symmetric monoidal categories. We will show this in section 7 with a number of examples; here we sketch two of them to whet the reader's appetite.

For the first, recall that in a compact closed category, the **trace** $\mathrm{tr}(f)$ of an endomorphism $f : A \to A$ is the composite

$$ I \xrightarrow{\eta} A \otimes A^* \xrightarrow{f \otimes \mathsf{id}} A \otimes A^* \xrightarrow{\sim} A^* \otimes A \xrightarrow{\varepsilon} I $$

where I is the unit object, A^* is the dual of A, and η and ε are the unit and counit of the duality (also called the coevaluation and evaluation, respectively). A fundamental property of trace is *cyclicity*, i.e. $\mathrm{tr}(fg) = \mathrm{tr}(gf)$ for any $f : A \to B$ and $g : B \to A$. The proof of this is essentially incomprehensible when written with composites of morphisms:

$$ \mathrm{tr}(fg) = \varepsilon\mathsf{s}(fg \otimes \mathsf{id})\eta = \varepsilon\mathsf{s}(f \otimes \mathsf{id})(g \otimes \mathsf{id})\eta = \varepsilon\mathsf{s}(f \otimes \mathsf{id})(\mathsf{id} \otimes \varepsilon \otimes \mathsf{id})(\eta \otimes \mathsf{id} \otimes \mathsf{id})(g \otimes \mathsf{id})\eta = $$

$$ \varepsilon\mathsf{s}(\mathsf{id} \otimes \varepsilon \otimes \mathsf{id})(f \otimes \mathsf{id} \otimes \mathsf{id} \otimes \mathsf{id})(\mathsf{id} \otimes \mathsf{id} \otimes g \otimes \mathsf{id})(\eta \otimes \eta) = (\varepsilon \otimes \varepsilon)\mathsf{s}(\mathsf{id} \otimes \mathsf{id} \otimes g \otimes \mathsf{id})(f \otimes \mathsf{id} \otimes \mathsf{id} \otimes \mathsf{id})(\eta \otimes \eta) = $$

$$ (\varepsilon \otimes \varepsilon)\mathsf{s}(\mathsf{id} \otimes \mathsf{id} \otimes g \otimes \mathsf{id})(\mathsf{id} \otimes \mathsf{id} \otimes \eta)(f \otimes \mathsf{id})\eta = \varepsilon\mathsf{s}(\mathsf{id} \otimes \varepsilon \otimes \mathsf{id})(g \otimes \mathsf{id} \otimes \mathsf{id} \otimes \mathsf{id})(\eta \otimes \mathsf{id} \otimes \mathsf{id})(f \otimes \mathsf{id})\eta = $$

$$ \varepsilon\mathsf{s}(g \otimes \mathsf{id})(\mathsf{id} \otimes \varepsilon \otimes \mathsf{id})(\eta \otimes \mathsf{id} \otimes \mathsf{id})(f \otimes \mathsf{id})\eta = \varepsilon\mathsf{s}(g \otimes \mathsf{id})(f \otimes \mathsf{id})\eta = \varepsilon\mathsf{s}(gf \otimes \mathsf{id})\eta = \mathrm{tr}(gf). $$

It becomes much more understandable when written in string diagram calculus, as in Figure 1. But in our type theory, the proof is one line of algebra:

$$ \mathrm{tr}(fg) \overset{\text{def}}{=} (\,|\, \lambda^B y \triangleleft fgy) = (\,|\, \lambda^B y \triangleleft fx, \lambda^A x \triangleleft gy) = (\,|\, \lambda^A x \triangleleft gfx) \overset{\text{def}}{=} \mathrm{tr}(gf). $$

The reader is not expected to understand this proof yet, but to give some flavor we explain that the two non-definitional equalities are "β-reductions for duality". The binary operator
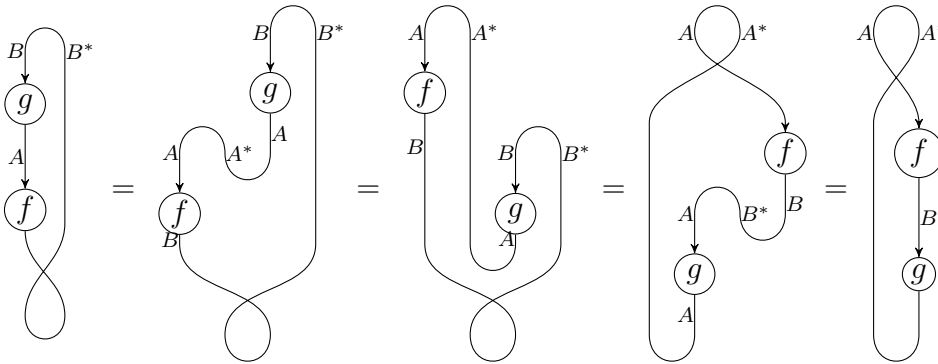
Figure 1: Cyclicity of trace with string diagrams

$\lhd$ denotes the counit ε, while a variable/abstraction pair $(x, \lambda^A x)$ denotes the unit η. A term of the form $\lambda^A x \lhd t$, where x does not occur in t, is a β-redex for duality, and in the resulting reduction this term is eliminated and the occurrence of x elsewhere[1] is substituted by t. (Semantically, this means applying a zigzag identity, which in string diagram notation is "straightening a bent string".) Thus the middle term $(\mid \lambda^B y \lhd fx, \lambda^A x \lhd gy)$ contains two β-redexes, and the resulting two reductions yield the two terms on either side.

For our second example, recall that if x is an element of a monoid M, and $\overline{x}$ and $\widehat{x}$ are both two-sided inverses[2] of x, then $\overline{x} = \widehat{x}$ by the following calculation:

$$\overline{x} = \overline{x}e = \overline{x}(x\widehat{x}) = (\overline{x}x)\widehat{x} = e\widehat{x} = \widehat{x}. \tag{1.2}$$

This same proof can be reproduced in cartesian type theory, and therefore holds for monoid objects in any cartesian monoidal category. It follows that if a monoid object M has an "inversion" morphism $i : M \to M$ such that

$$\begin{array}{ccc}
M \times M & \xrightarrow{i \times 1} & M \times M \\
{\scriptstyle \triangle}\nearrow & & \searrow{\scriptstyle m} \\
M \xrightarrow{\;\;!\;\;} 1 \xrightarrow{\;\;e\;\;} M \\
{\scriptstyle \triangle}\searrow & & \nearrow{\scriptstyle m} \\
M \times M & \xrightarrow[1 \times i]{} & M \times M
\end{array} \tag{1.3}$$

commutes (thereby making it a group object), then i is the unique such morphism.

An interesting application of this relies on the fact that the category $\mathrm{CComon}(\mathcal{C})$ of *cocommutative comonoids* in any symmetric monoidal category $\mathcal{C}$ is in fact *cartesian monoidal*, with the cartesian product being the monoidal structure of $\mathcal{C}$. A monoid object

---

[1]This "elsewhere" could be anywhere at all. Thus although $\lambda^A x$ does "bind" the variable x, it does not delimit its scope.

[2]In fact, of course, it suffices for $\overline{x}$ to be a left inverse and $\widehat{x}$ a right inverse.

in $\mathrm{CComon}(\mathcal{C})$ is a *cocommutative bimonoid object* in $\mathcal{C}$: an object with both a monoid and a cocommutative comonoid structure, such that the monoid structure maps are comonoid morphisms (or equivalently the comonoid structure maps are monoid morphisms). For any bimonoid object, a morphism $i : M \to M$ satisfying the analogue of (1.3):

$$
\begin{array}{ccc}
M \otimes M & \xrightarrow{\;i \otimes 1\;} & M \otimes M \\
{\scriptstyle\triangle}\nearrow & & \searrow{\scriptstyle m} \\
M \xrightarrow{\;\;\varepsilon\;\;} & I \xrightarrow{\;\;e\;\;} & M \\
{\scriptstyle\triangle}\searrow & & \nearrow{\scriptstyle m} \\
M \otimes M & \xrightarrow[\;1 \otimes i\;]{} & M \otimes M
\end{array}
\tag{1.4}
$$

(where $\triangle, \varepsilon$ are the comonoid structure) is called an *antipode*, and a bimonoid object equipped with an antipode is called a *Hopf monoid*. Thus, the classical argument for uniqueness of inverses, phrased in cartesian type theory, implies that a cocommutative bimonoid object has at most one antipode. Dualizing, we can conclude that a *commutative* bimonoid object also has at most one antipode.

For bimonoids that are neither commutative nor cocommutative, cartesian type theory cannot help us. We can manually translate the classical proof (1.2) into commutative diagrams and then replace all the $\times$'s by $\otimes$'s, but this is tedious and one feels that there should be a better way. And indeed, in our type theory for symmetric monoidal categories, we can write a very close analogue of (1.2) that applies to arbitrary bimonoids:

$$
\overline{x} = \overline{x}\,e = (\overline{x_{(1)}}\,e \mid \cancel{x_{(2)}}) = \overline{x_{(1)}}\,x_{(2)}\,\widehat{x_{(3)}} = (e\,\widehat{x_{(2)}} \mid \cancel{x_{(1)}}) = e\,\widehat{x} = \widehat{x}.
\tag{1.5}
$$

Again, the reader is not expected to understand this completely, but the resemblance to (1.2) should be clear. The main differences are the subscripts on the x's and the insertion of the steps $(\overline{x_{(1)}}\,e \mid \cancel{x_{(2)}})$ and $(e\,\widehat{x_{(2)}} \mid \cancel{x_{(1)}})$. The subscripts are used to track applications of the comultiplication $\triangle : M \to M \otimes M$. This is necessary because (unlike in the cartesian case) not all morphisms are comonoid morphisms (so that $f(x)_{(1)}$ might differ from $f(x_{(1)})$), and also because in the absence of cocommutativity the resulting "copies" of x must be distinguished (so that $(x_{(1)}, x_{(2)})$ is different from $(x_{(2)}, x_{(1)})$). Similarly, the canceled terms such as $\cancel{x_{(2)}}$ track applications of the counit $\varepsilon : M \to I$, and the steps involving these are inserted in order to match the middle composite $M \xrightarrow{\varepsilon} I \xrightarrow{e} M$ in (1.4).

With this as preview and motivation, we now move on to a somewhat more detailed description of our type theory.

1.6. GENERALIZED SWEEDLER NOTATION. The idea behind our type theory is to formalize and generalize the informal *Sweedler notation* that is common in coalgebra theory. A coalgebra is a comonoid in the category of vector spaces, i.e. a vector space C with a comultiplication $\triangle : C \to C \otimes C$ that is coassociative and counital. Since the elements of the tensor product $C \otimes C$ are finite sums of generating tensors $c_{(1)} \otimes c_{(2)}$, we can write

$$
\triangle(c) = \sum_i c^i_{(1)} \otimes c^i_{(2)}.
$$

Sweedler's notation is to omit the index i, and sometimes also the symbol $\sum$, obtaining

$$\triangle(c) = c_{(1)} \otimes c_{(2)}.$$

(This is therefore a sort of "dual" to the "Einstein summation convention" for products of tensors, which also has a type-theoretic formalization known to physicists as "abstract index notation" [PR84].) For instance, coassociativity can then be expressed as

$$c_{(1)} \otimes c_{(2)(1)} \otimes c_{(2)(2)} = c_{(1)(1)} \otimes c_{(1)(2)} \otimes c_{(2)}$$

(which is then often written as $c_{(1)} \otimes c_{(2)} \otimes c_{(3)}$), and counitality, for a counit ε, can be expressed as

$$c_{(1)} \cdot \varepsilon(c_{(2)}) = c = \varepsilon(c_{(1)}) \cdot c_{(2)}$$

where $\cdot$ denotes scalar multiplication.

We generalize this notation to apply to any morphism $f : A \to B \otimes C$ in any monoidal category whose codomain is a tensor product. Of course we now have to notate *which* morphism we are talking about, so instead of $c_{(1)} \otimes c_{(2)}$ we write $f_{(1)}(x) \otimes f_{(2)}(x) : B \otimes C$, where $x : A$ is a formal variable. Note that in Sweedler's original notation for coalgebras *in vector spaces*, the element $\triangle(c)$ really is a sum of generating tensors $c_{(1)} \otimes c_{(2)}$, whereas now we are in an arbitrary monoidal category so that this doesn't even make sense. Nevertheless, we can still use a similar *formal* notation governed by appropriate rules, just as in type theory we use variables and terms assigned to objects of the category as "types" even though the objects of an arbitrary category may not actually have "elements".

However, as a nod towards this extra formality (and also an extra step in the "types are like sets" direction), we stop using the symbol $\otimes$ and write instead a pair $(f_{(1)}(x), f_{(2)}(x)) : (B, C)$ (although see subsection 1.9). We also have to incorporate an analogue of "scalar multiplication", which in a general monoidal category means the unit isomorphism $C \otimes I \cong C$; for consistency we collect all the "scalars" in a separate tuple separated by a bar. Thus, for instance, the counitality of a comultiplication $\triangle : C \to C \otimes C$ with counit $\varepsilon : C \to I$ in an arbitrary monoidal category will be expressed as

$$(\triangle_{(1)}(c) \mid \varepsilon(\triangle_{(2)}(c))) = c = (\triangle_{(2)}(c) \mid \varepsilon(\triangle_{(1)}(c))).$$

And the composite of $f : A \otimes B \to C \otimes D$ with $g : E \otimes D \to F \otimes G$ along the object D is

$$\frac{\begin{array}{c} x : A, y : B \vdash (f_{(1)}(x, y), f_{(2)}(x, y)) : (C, D) \\ z : E, w : D \vdash (g_{(1)}(z, w), g_{(2)}(z, w)) : (F, G) \end{array}}{x : A, y : B, z : E \vdash (f_{(1)}(x, y), g_{(1)}(z, f_{(2)}(x, y)), g_{(2)}(z, f_{(2)}(x, y))) : (C, F, G)}$$

Note that in contrast to intuitionistic linear logic, the variables in a context are not literally treated "linearly" in our terms, since they can occur multiple times in the multiple "components" of a map f. Instead, the "usages" of a variable are controlled by the codomain arity of the morphisms applied to them.

One further technical device is required to deal with morphisms having nullary domain. Suppose we have $f : I \to B \otimes C$, written in our type theory as $\vdash (f_{(1)}, f_{(2)}) : (B, C)$, and we compose/tensor it with itself (and apply a symmetry) to get a morphism $I \to B \otimes B \otimes C \otimes C$. We would naïvely write this as $\vdash (f_{(1)}, f_{(1)}, f_{(2)}, f_{(2)}) : (B, B, C, C)$, but this is ambiguous since we can't tell which $f_{(1)}$ matches which $f_{(2)}$. Thus, we disambiguate the possibilities by annotating the terms in pairs, such as $() \vdash (f_{(1)}, f'_{(1)}, f_{(2)}, f'_{(2)})$ or $() \vdash (f_{(1)}, f'_{(1)}, f'_{(2)}, f_{(2)})$.

The "variable-binding notation" $(x, \lambda^A x)$ for duality that we mentioned earlier is just syntactic sugar for a special case of this: we use variables like x as the "labels" for unit morphisms $\eta_A : () \to (A, A^*)$ in place of primes, writing $(x, \lambda^A x)$ instead of $((\eta_A)^x_{(1)}, (\eta_A)^x_{(2)})$. Of course, this requires that the "variables" used as labels of this sort are disjoint from those occurring in the actual context. The operator $\triangleleft$ is just an infix notation for the counit $\varepsilon : (A^*, A) \to ()$.

Another bit of syntactic sugar is that if each type is equipped with at most one specified comonoid structure, then there is no ambiguity in reverting back to Sweedler's original notation $(x_{(1)}, x_{(2)})$ instead of $(\triangle_{(1)}(x), \triangle_{(2)}(x))$. For instance, the axiom of a Frobenius algebra can be written as

$$(x_{(1)}, x_{(2)}y) = ((xy)_{(1)}, (xy)_{(2)}) = (xy_{(1)}, y_{(2)}) \tag{1.7}$$

while the principal axiom of a bialgebra is

$$((xy)_{(1)}, (xy)_{(2)}) = (x_{(1)}y_{(1)}, x_{(2)}y_{(2)}). \tag{1.8}$$

Note that if all types have such a comonoid structure *and* all morphisms are comonoid morphisms, which in type-theoretic notation means $f(x)_{(i)} = f(x_{(i)})$ and $\varepsilon(f(x)) = \varepsilon(x)$, then the monoidal structure automatically becomes cartesian. In ordinary type theory for cartesian monoidal categories, the diagonal (i.e. the comultiplication) is represented by literally duplicating a variable (x, x); thus our subscripting of variables $(x_{(1)}, x_{(2)})$ can be viewed as a minimal modification of this to deal with situations where the comultiplication is not literally a cartesian diagonal. (Indeed, the comultiplication of a coalgebra is often viewed as a non-cartesian substitute for the diagonal, e.g. in the theory of quantum groups.) We can similarly regard the counit ε as a "discarding" operation, in which case I like to notate $\varepsilon(x)$ by χ (as was done in (1.5)).

1.9. AN ALTERNATIVE NOTATION FOR COALGEBRAISTS. The particular symbols we choose in the syntax of our type theory are, of course, immaterial. I personally find the pair notation $(f_{(1)}(x), f_{(2)}(x))$ evocative and helpful, since I am more used to thinking of sets than coalgebras. However, a reader who is more used to ordinary coalgebras in vector spaces may prefer to retain $\otimes$ as a formal symbol in place of a comma, and use a symbol like $\cdot$ for "scalar multiplication" in place of the separating bar $|$. For instance, in this notation the axioms (1.7) and (1.8) become

$$x_{(1)} \otimes x_{(2)}y = (xy)_{(1)} \otimes (xy)_{(2)} = xy_{(1)} \otimes y_{(2)} \qquad (xy)_{(1)} \otimes (xy)_{(2)} = x_{(1)}y_{(1)} \otimes x_{(2)}y_{(2)}.$$

while the calculation (1.5) becomes

$$\overline{x} = \overline{x}\, e = \overline{x_{(1)}}\, e \cdot \varepsilon(x_{(2)}) = \overline{x_{(1)}}\, x_{(2)}\, \widehat{x_{(3)}} = e\, \widehat{x_{(2)}} \cdot \varepsilon(x_{(1)}) = e\, \widehat{x} = \widehat{x}.$$

A reader who prefers this notation is free to use it instead. (However, one should probably then choose a different notation for the tensor product *types* discussed in Remark 2.6.)

1.10. SOME PLURALISTIC REMARKS. Since many readers will be more familiar with other syntaxes for monoidal categories such as intuitionistic linear logic or string diagrams, it is worth saying a few words about their complementary relationship to our type theory, and in particular how and why they can all coexist.

Intuitionistic linear logic was discussed briefly above; from our present point of view, it has the disadvantages of a heavy and unintuitive "match" syntax for decomposing elements of tensor products. However, it also has definite advantages,[3] such as an asymptotically more concise syntax: the present type theory involves much duplication of terms, since the arguments of a function are repeated in every component of its output. This matters little for the examples we consider here, but might become more important if it were ever implemented at scale in a computer proof assistant. Intuitionistic linear logic also cleanly extends to a syntax for *closed* monoidal categories, whereas with our current type theory this seems difficult.

String diagrams (a.k.a. graphical calculus) are another well-known and very powerful tool for working with structures in many kinds of monoidal categories, which also have both strengths and weaknesses. String diagrams are at their best when dealing with structures whose axioms "don't change the topology", such as dual pairs and Frobenius algebras, since in this case many proofs are simply topological deformations. For structures whose axioms do change the topology, such as bialgebras and Hopf algebras, string diagrams are still useful, but in such cases only some of the proof steps can be reduced to simple topological deformations: generally those steps involving pure *naturality* conditions.

By contrast, our type theory does not "see" the topological nature of dual pairs and Frobenius algebras. However, it represents most naturality conditions as *syntactic identities*, making them completely invisible; for instance, the two middle steps in Figure 1 disappear entirely in (1.5). Moreover, it leverages a different intuition, allowing us to think of objects of an arbitrary monoidal category as "sets" with "elements" (modulo certain "linearity" restrictions). This seems to be particularly useful for coalgebraic and Hopf-type structures — perhaps unsurprisingly, given the origin of our syntax in Sweedler notation.

Another difference is that string diagrams incarnate categorical duality in an obvious way — by simply rotating or reflecting a string diagram — while our type theory breaks this duality, treating inputs and outputs differently. One might consider this an advantage of string diagrams. But breaking duality in the syntax is part of what gives us the above advantages (notably the view of types as sets), and it also means that duality becomes a nontrivial and useful technique: in a situation that is not self-dual, we can choose which orientation of the type theory is most convenient. For instance, in subsection 7.9 we will

---

[3]I am indebted to Matt Oliveri for these points.

use our type theory to show that the antipode of a (weak) Hopf monoid is a monoid anti-homomorphism; it then follows by duality that it is also a comonoid anti-homomorphism, although a direct proof of the latter in our syntax would be less intuitive.

There is no reason to expect any one notation for symmetric monoidal categories to be the best in all situations: each has applications to which it is better-suited and others to which it is not as well suited. The new type theory presented here makes no claim to supplant string diagrams or intuitionistic linear logic; rather it is an alternative that may be more convenient in *some* applications. Only time and experience can render a final verdict on the usefulness of a notation, as well as its generalizations and limitations — e.g. to what extent can the type theory presented in this paper be generalized to other kinds of monoidal categories, such as closed, ∗-autonomous, braided, ribbon, and planar monoidal categories, indexed monoidal categories, (symmetric) monoidal bicategories, and so on? My primary hope for this paper is to begin a conversation about type theories in this family and their potential uses.

1.11. ACKNOWLEDGEMENTS. I would like to thank Dan Licata, Robin Cockett, Peter LeFanu Lumsdaine, and Matt Oliveri for useful conversations.

2. Props

To simplify and clarify the semantics of our type theory, we will interpret it in a categorical structure that reflects the type-theoretic distinction between judgmental and type operations. To explain what this means, recall that ordinary cartesian type theory is often interpreted in categories with finite products; but in this case both the judgmental comma (in a context $(x : A, y : B, z : C)$) and the product type operation (in a product type $(A \times B) \times C$) are interpreted by the same categorical operation (cartesian product types), which can cause confusion. A more direct semantics of cartesian type theory maintains this distinction by using a "cartesian multicategory", allowing the judgmental comma to be interpreted by the "categorical comma", i.e. the concatenation that forms a list of objects to be the domain of a morphism in a multicategory.

For type theories like classical linear logic, which allow multiple types in both domain and codomain but use commas on the left and right of the turnstile to represent different tensor products, the appropriate "multicategorical" structure is called a "polycategory" [Sza75] (the corresponding "monoidal" version being a linearly distributive [CS97, BCST96] or ∗-autonomous [Bar79, Bar91] category). In our case, where the two tensor products are the same, the appropriate structure is called a *prop*.

2.1. DEFINITION. A **prop** $\mathcal{P}$ consists of

(i) A set $\mathrm{ob}(\mathcal{P})$ of **objects**, and

(ii) A symmetric strict monoidal category (that is, a symmetric monoidal category whose associators and unitors are identities) whose underlying monoid of objects is freely generated by $\mathrm{ob}(\mathcal{P})$.

(The original Adams–MacLane [Mac65] definition of prop had only one object; thus our "props" are sometimes called "colored props".)

We write the objects of the monoidal category in (ii) as finite lists $(A, B, \ldots, Z)$ of objects of the prop (i.e. $A, B, \ldots, Z \in \mathrm{ob}(\mathcal{P})$). The monoidal structure is given by concatenation of lists; the unit object is the empty list ().

We now summarize the relationship between props and symmetric monoidal categories.

2.2. DEFINITION. A **tensor product** $A \otimes B$ of objects in a prop is an object together with an isomorphism

$$(A, B) \xrightarrow{\sim} (A \otimes B)$$

in the monoidal category of Definition 2.1(ii). Similarly, a **unit** is an object I with an isomorphism

$$() \xrightarrow{\sim} (I).$$

A prop is called **representable** if it has a unit and any pair of objects has a tensor product.

A **morphism of props** $\omega : \mathcal{P} \to \mathcal{Q}$ is a function $\omega_0 : \mathrm{ob}(\mathcal{P}) \to \mathrm{ob}(\mathcal{Q})$ together with a *strict* symmetric monoidal functor whose action on objects is obtained by applying ω_0 elementwise to lists.

2.3. THEOREM. *The category of symmetric monoidal categories and strong symmetric monoidal functors is equivalent to the subcategory of representable props (and all morphisms between them).*

SKETCH OF PROOF. Every symmetric monoidal category $\mathcal{C}$ has an underlying prop $U\mathcal{C}$ with the same objects, and in which a morphism $(A, \ldots, B) \to (C, \ldots, D)$ is a morphism $A \otimes \cdots \otimes B \to C \otimes \cdots \otimes D$ in $\mathcal{C}$. This construction U is functorial on strong symmetric monoidal functors, and the props in its image are representable. (In fact, it is the functorial strictification.) The functor U is faithful since the action of $f : \mathcal{C} \to \mathcal{D}$ on objects and arrows is preserved in Uf, while the coherence constraints of f are recorded in the action of Uf on the isomorphisms from Definition 2.2. Similarly, the functor U is full, since the action of a morphism $U\mathcal{C} \to U\mathcal{D}$ on the isomorphisms from Definition 2.2 induces symmetric monoidal constraints on its underlying ordinary functor. Finally, every representable prop induces (by choosing tensor products and a unit) a symmetric monoidal structure on its category of unary and co-unary morphisms, and it is isomorphic to the underlying prop thereof. ∎

We can say more: this subcategory is an injectivity-class, and the corresponding category of *algebraic* injectives (objects equipped with chosen lifts against the generating morphisms, and maps that preserve the chosen lifts) is equivalent to the category of symmetric monoidal categories and *strict* symmetric monoidal functors. In particular, the latter is monadic over the category of props.

The fundamental "initiality theorem" for semantics of our type theory, which we prove in section 5, is that the "term model" is the prop freely generated by some input data. Following [BCR18], we will call this input data a *signature*.

2.4. DEFINITION. A **signature** $\mathcal{G}$ is a set of objects together with a set of arrows, each assigned a domain and codomain that are both finite lists of objects.

2.5. THEOREM. *The category of props is monadic over the category of signatures.*

PROOF. Just like the proof in [BCR18, Appendix A.2] for the one-object case. ∎

Thus, every prop has a *presentation* as a coequalizer of a pair of maps between free props. In section 6 we will extend our type theory to construct "presented props" as well, allowing equational reasoning as in the examples from subsection 1.1.

2.6. REMARK. The fact that props allow multiple objects in both domains and codomains means that we rarely need to talk about actual tensor product types $A \otimes B$ with semantics in tensor product objects (Definition 2.2). For this reason, we will not include such types in our formal system. However, we note that if necessary, they can be added quite easily: because Definition 2.2 simply asserts objects, morphisms, and equations (rather than a unique factorization property), it can essentially be ensured as a special case of a prop presentation. Thus we need only add some generating terms and axioms to our type theory, for any pair of objects A, B whose tensor product we need to talk about:

$$x : A, y : B \vdash \langle x, y \rangle : A \otimes B \qquad\qquad p : A \otimes B \vdash (\pi_{(1)}(p), \pi_{(2)}(p)) : (A, B)$$

$$(x, y) = (\pi_{(1)}\langle x, y \rangle, \pi_{(2)}\langle x, y \rangle) \qquad\qquad p = \langle \pi_{(1)}(p), \pi_{(2)}(p) \rangle.$$

Note how similar this is to the treatment of cartesian products in cartesian type theory with pairing and projection operations.

3. On the admissibility of structural rules

In general, type theories consist of *rules* for deriving *judgments* about *terms*. The most common judgments are *typing judgments* (that a term belongs to a type, or in our case a tuple of terms belong to a tuple of types) and *equality judgments* (that two terms — or tuples of terms — are equal). A tree of rules ending with a judgment is called a *derivation* of that judgment, and the categorical structure presented by a type theory is built out of the *derivable* (or *valid*) judgments.

Now in proving that this categorical structure does in fact have the desired universal property, it is very useful if we arrange the type theory so that every derivable judgment has a *unique* derivation. The reason for this is that we want a morphism in our categorical semantics to be determined by a *term itself*, not by a choice of derivation of that term; but the natural way to prove the desired universal property (a.k.a. the "initiality theorem" for that type theory) is by induction *over derivations*. Thus, if the same term can arise from multiple derivations, proving this universal property requires an extra step of proving that this choice is immaterial (i.e. that any two derivations of the same term determine the same morphism in the semantics). This step is tricky and often omitted in the literature, leading to incomplete proofs. It becomes even trickier when considering higher-categorical

semantics, in which the morphisms determined by two derivations of the same term may be only *isomorphic* rather than equal.

The choice that terms should have unique derivations essentially requires that nearly all structural rules should be admissible rather than primitive. (Recall that an *admissible rule* is one that is *not* asserted as part of the specification of the type theory, but for which we can prove after the fact that whenever we have derivations of its premises we can construct a derivation of its conclusion — usually by inductively traversing and modifying the given derivations of its premises. The *structural rules* are, roughly speaking, those that correspond to the operations of the categorical structure used as the semantics: composition in a category, permutation of domain lists in a symmetric multicategory, and permutation of both domains and codomains in a polycategory or prop.)

The reason this requirement arises is that structural rules generally have to satisfy equations that are "tautological" in their action on terms. For instance, composing $f : A \to B$ with $g : B \to C$ and $h : C \to D$ in the two possible ways (semantically, $h \circ (g \circ f)$ and $(h \circ g) \circ f$) produces the same term:

$$\frac{\dfrac{x : A \vdash f(x) : B \qquad y : B \vdash g(y) : C}{x : A \vdash g(f(x)) : C} \qquad z : C \vdash h(z) : D}{x : A \vdash h(g(f(x))) : D}$$

$$\frac{x : A \vdash f(x) : B \qquad \dfrac{y : B \vdash g(y) : C \qquad z : C \vdash h(z) : D}{y : B \vdash h(g(y)) : D}}{x : A \vdash h(g(f(x))) : D} \ .$$

Thus, if composition were a primitive rule, this term would have two distinct derivations. But if composition (i.e. substitution) is an admissible rule, then we can *prove* that the derivations *constructed* by applying it in these two different ways *turn out to be* the same. Similarly, if permutation were primitive, then for any $f : (A, B) \to C$ we would have two (in fact, infinitely many) derivations of the same term:

$$\frac{}{x : A, y : B \vdash f(x, y) : C} \qquad\qquad \frac{\dfrac{\dfrac{}{x : A, y : B \vdash f(x, y) : C}}{y : B, x : A \vdash f(x, y) : C}}{x : A, y : B \vdash f(x, y) : C}$$

whereas if permutation (a.k.a. "exchange") is admissible, then its functoriality as an operation on derivations can be proven.

Type theorists know how to make a rule admissible: we build "just enough" of it into the primitive rules. For instance, if we introduce generating morphisms such as $f : (A, B) \to C$ and $g : C \to D$ as simple axioms (i.e. rules with no premises):

$$\frac{}{x : A, y : B \vdash f(x, y) : C} \qquad\qquad\qquad \frac{}{z : C \vdash g(z) : D}$$

then there would be no way to construct derivations of composites such as

$$\frac{x : A, y : B \vdash f(x,y) : C \qquad z : C \vdash g(z) : D}{x : A, y : B \vdash g(f(x,y)) : D} \tag{3.1}$$

except by using a *primitive* composition/substitution rule. Therefore, we instead introduce each generating morphism in "Yoneda style" by allowing ourselves to *postcompose* any given term(s) with it. For instance, in cartesian type theory we introduce generators with rules such as

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash f(a,b) : C} \qquad\qquad \frac{\Gamma \vdash c : C}{\Gamma \vdash g(c) : D}$$

for arbitrary contexts Γ and terms a, b, c. Now (3.1) can be obtained without a primitive substitution rule:

$$\frac{\dfrac{\overline{x : A, y : B \vdash x : A} \qquad \overline{x : A, y : B \vdash y : B}}{x : A, y : B \vdash f(x,y) : C}}{x : A, y : B \vdash g(f(x,y)) : D} \tag{3.2}$$

In categorical terms, the point is that we can build the free category on a graph either by freely adding all binary (and nullary) composites and then quotienting by the relation of associativity, or we can avoid the need to quotient at all by defining the morphisms as composable lists of generating arrows, where a "list" is defined inductively as either an empty list or a list postcomposed by a generating arrow — that is, we enforce right-associated composites $k \circ (h \circ (g \circ (f \circ \mathsf{id})))$.

   This technique is trickier in non-cartesian type theories, since we cannot keep the same context all the way through the derivation. That is, in a cartesian monoidal category we can start (3.2) with the "identity" or "axiom" rules $x : A, y : B \vdash x : A$ and $x : A, y : B \vdash y : B$, corresponding categorically to the projections $A \times B \to A$ and $A \times B \to B$; but in a non-cartesian monoidal category such projections do not exist. Thus, we need to concatenate contexts as we go down the derivation tree. For instance, the generator rule for $f : (A, B) \to C$ must be something like

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B}{\Gamma, \Delta \vdash f(a,b) : C}. \tag{3.3}$$

   However, now we have a new problem: the exchange rule (permutations of the domain). In the cartesian case, we can make this admissible by "propagating it up" the entire derivation since the context remains the same. For instance, if we permute the inputs of (3.2) we would simply get

$$\frac{\dfrac{\overline{y : B, x : A \vdash x : A} \qquad \overline{y : B, x : A \vdash y : B}}{y : B, x : A \vdash f(x,y) : C}}{y : B, x : A \vdash g(f(x,y)) : D} \tag{3.4}$$

$$x_1 : A_1, \ldots, x_m : A_m \vdash (M_1, \ldots, M_n \mid Y_1, \ldots, Y_p) : (B_1{}^{\star?}, \ldots, B_n{}^{\star?}).$$
$$x_1 : A_1, \ldots, x_m : A_m \vdash (M_1, \ldots, M_n \mid Y_1, \ldots, Y_p) = (N_1, \ldots, N_n \mid Z_1, \ldots, Z_q) : (B_1, \ldots, B_n).$$

Figure 2: Judgments

$$\Gamma \vdash (\vec{M} \mid \vec{Y}) : \vec{B}^{\star?} \qquad\qquad \Gamma \vdash (\vec{M} \mid \vec{Y}) = (\vec{N} \mid \vec{Z}) : \vec{B}.$$

Figure 3: Judgments, abbreviated

But in the non-cartesian case this doesn't work. We don't want to assert a primitive exchange rule as this would break the "terms have unique derivations" principle, so instead we build exchange into the generator rule (3.3). Our first try might be something like

$$\frac{\Gamma \vdash a : A \qquad \Delta \vdash b : B \qquad \sigma : \Gamma, \Delta \cong \Phi}{\Phi \vdash f(a, b) : C}$$

where σ is an arbitrary permutation. But this too breaks the "terms have unique derivations" principle, since a permutation of types *within* Γ or Δ could be obtained either as part of σ or by operating on the input derivations of $\Gamma \vdash a : A$ and $\Delta \vdash b : B$. Instead we have to build in "just enough" exchange but not too much, by requiring σ to be not an arbitrary permutation but a *shuffle*: a permutation of (Γ, Δ) that preserves the *relative* order of the types in Γ and in Δ. We will write $\mathrm{Shuf}(\Gamma; \Delta)$ for the set of such shuffles.

This may seem overly technical, but the presence of such things as shuffles need never be seen by the *user* of the type theory. Indeed, maintaining the "terms have unique derivations" principle is precisely what *allows* the user to work only with terms, ignoring the details of the derivations.

4. The type theory for free props

Let $\mathcal{G}$ be a signature; we will define a type theory that presents the free prop on $\mathcal{G}$. Our general typing judgment will be of the form shown in Figure 2. For reasons to be explained later, we annotate some of the types in the consequent of each judgment with a superscript star, $B^\star$, and call them *active*; we write $B^{\star?}$ to mean that B might be active.

We use vector notation $\vec{M}, \vec{A}$, etc. to indicate a list of terms or types, so the judgments can be abbreviated as in Figure 3, although this omits the information that $\vec{M}$ and $\vec{N}$ must have the same length as $\vec{B}$ (while the length of $\vec{Y}$ and $\vec{Z}$ is unrestricted). If $\vec{Y}$ is empty, we write $(\vec{M} \mid)$ as simply $(\vec{M})$. If furthermore $\vec{M}$ and $\vec{B}$ have length 1, we omit the parentheses, writing simply $M : B$. When all the lists are empty, we have $\vdash () : ()$, which will be valid (it represents the identity morphism of the unit object).

$$\frac{(x : A) \in \Gamma}{\Gamma \vdash x \text{ term}} \qquad \frac{f \in \mathcal{G}(; B_1, \dots, B_n) \qquad \mathfrak{a} \in \mathfrak{A} \qquad n \geq 2 \qquad 1 \leq k \leq n}{\Gamma \vdash f^{\mathfrak{a}}_{(k)} \text{ term}}$$

$$\frac{f \in \mathcal{G}(; B) \qquad \mathfrak{a} \in \mathfrak{A}}{\Gamma \vdash f^{\mathfrak{a}} \text{ term}} \qquad\qquad \frac{f \in \mathcal{G}(;)}{\Gamma \vdash f \text{ term}}$$

$$\frac{\Gamma \vdash M_1 \text{ term} \quad \dots \quad \Gamma \vdash M_m \text{ term}}{f \in \mathcal{G}(A_1, \dots, A_m; B_1, \dots, B_n) \qquad m \geq 1 \qquad n \geq 2 \qquad 1 \leq k \leq n}{\Gamma \vdash f_{(k)}(M_1, \dots, M_m) \text{ term}}$$

$$\frac{\Gamma \vdash M_1 \text{ term} \quad \dots \quad \Gamma \vdash M_m \text{ term}}{f \in \mathcal{G}(A_1, \dots, A_m; B_1, \dots, B_n) \qquad m \geq 1 \qquad n \leq 1}{\Gamma \vdash f(M_1, \dots, M_m) \text{ term}}$$

Figure 4: Terms

In these judgments A_i, B_j are types, x_i are variables, and M_j, N_j, Y_k, Z_ℓ are terms. Here by a *type* we simply mean an object of our generating signature $\mathcal{G}$, since there are no type-forming operations in the theory. There is nothing new in our *variables*; the reader who prefers de Bruijn indices is free to think of them in that way, although since our syntax has no variable binding[4] the usual subtleties of capture-avoidance are irrelevant.

The *terms* are defined inductively by the rules shown in Figure 4. Recall that in general our typing judgments involve a *list* of such terms, one for each type in the codomain, together with a list of scalar terms. The terms defined in Figure 4 do not yet have any types, but they are "well-scoped" by definition: they come with a context and only use variables occurring in that context. We subscript only applications of functions with greater than unary codomain, and as noted in section 1, we annotate each occurrence of a morphism with nullary domain and positive-ary codomain with some element of an infinite alphabet of symbols $\mathfrak{A}$ (such as $',\!'',\!''',\dots$, or $1, 2, 3, \dots$).

We will write $\vec{f}(\vec{M})$ for the list of all subscriptings of the application of f to the arguments $\vec{M}$. For instance, if $f : (A, B, C) \to (D, E)$ then $\vec{f}(\vec{M})$ would be

$$(f_{(1)}(M_1, M_2, M_3), f_{(2)}(M_1, M_2, M_3)).$$

---

[4]Recall that in the notation $(x, \lambda^A x)$ for the unit of a duality, x is not actually a variable in this sense but rather a "label"; we will introduce labels formally in a moment.

$$x_k[M_1, \ldots, M_n/x_1, \ldots, x_n] = M_k$$
$$y[\vec{M}/\vec{x}] = y \qquad\qquad (y \notin \vec{x})$$
$$f^{\mathfrak{a}}_{(k)}[\vec{M}/\vec{x}] = f^{\mathfrak{a}}_{(k)}$$
$$f^{\mathfrak{a}}[\vec{M}/\vec{x}] = f^{\mathfrak{a}}$$
$$f[\vec{M}/\vec{x}] = f$$
$$f_{(k)}(N, \ldots, P)[\vec{M}/\vec{x}] = f_{(k)}(N[\vec{M}/\vec{x}], \ldots, P[\vec{M}/\vec{x}])$$
$$f(N, \ldots, P)[\vec{M}/\vec{x}] = f(N[\vec{M}/\vec{x}], \ldots, P[\vec{M}/\vec{x}])$$

Figure 5: Simultaneous substitution into terms

More generally, for $f \in \mathcal{G}(A_1, \ldots, A_m; B_1, \ldots, B_n)$ we have

$$\vec{f}(\vec{M}) = \begin{cases} (f^{\mathfrak{a}}_{(1)}, \ldots, f^{\mathfrak{a}}_{(n)}) & m = 0,\ n \geq 2 \\ f^{\mathfrak{a}} & m = 0,\ n = 1 \\ f & m = 0,\ n = 0 \\ (f_{(1)}(\vec{M}), \ldots, f_{(n)}(\vec{M})) & m \geq 1,\ n \geq 2 \\ f(\vec{M}) & m \geq 1,\ n \leq 1. \end{cases}$$

In the first and second case, we also write $\vec{f}^{\mathfrak{a}}$ to notate the label $\mathfrak{a}$. In the third case, we allow ourselves to write $\vec{f}^{\mathfrak{a}}$ to mean simply f (discarding the label). Finally, when $n \leq 1$ we allow ourselves to write $f_{(1)}$ or $f_{(1)}(\vec{M})$ to mean f or $f(\vec{M})$ respectively.

We define the "simultaneous substitution" of a list of terms $\vec{M}$ for a list of variables $\vec{x}$ in the usual way, as shown in Figure 5.

We now move on to the rules governing our typing judgment. In section 3 we argued that by incorporating Yoneda-style generator rules and shuffles, we can make composition and exchange admissible and thereby ensure that any judgment has a unique derivation. In the case of props, we also want to make the monoidal structure admissible (since it satisfies strict associativity and interchange laws that we would otherwise have to assert as judgmental equalities). In particular, for morphisms $f : A \to B$ and $g : C \to D$ we would like the judgment

$$x : A, y : C \vdash (f(x), g(y)) : (B, D)$$

to have a unique derivation. Symmetry suggests that this unique derivation cannot apply f first and then g or vice versa. Thus, we replace the generator rule by a "multi-generator" rule allowing only a one-step derivation

$$\frac{x : A, y : C \vdash (x, y) : (A, C)}{x : A, y : C \vdash (f(x), g(y)) : (B, D)}$$

A first approximation to the general form of this rule is

$$\frac{\Gamma \vdash (\vec{M}, \ldots, \vec{N}, \vec{P} \mid \vec{Z}) : (\vec{A}, \ldots, \vec{B}, \vec{C}) \qquad f \in \mathcal{G}(\vec{A}; \vec{D}) \qquad \cdots \qquad g \in \mathcal{G}(\vec{B}; \vec{E})}{\Gamma \vdash (\vec{f}(\vec{M}), \ldots, \vec{g}(\vec{N}), \vec{P} \mid \vec{Z}) : (\vec{D}, \ldots, \vec{E}, \vec{C}).}$$

However, if there are generators with nullary codomain, we need to collect them into the scalar terms $\vec{Z}$. Thus a second approximation is

$$\frac{\begin{array}{c} \Gamma \vdash (\vec{M}, \ldots, \vec{N}, \vec{P}, \ldots, \vec{Q}, \vec{R} \mid \vec{Z}) : (\vec{A}, \ldots, \vec{B}, \vec{C}, \ldots, \vec{D}, \vec{E}) \\ f \in \mathcal{G}(\vec{A}; \vec{F}_{\geq 1}) \qquad \cdots \qquad g \in \mathcal{G}(\vec{B}; \vec{G}_{\geq 1}) \\ h \in \mathcal{G}(\vec{C};) \qquad \cdots \qquad k \in \mathcal{G}(\vec{D};) \end{array}}{\Gamma \vdash \left(\vec{f}(\vec{M}), \ldots, \vec{g}(\vec{N}), \vec{R} \mid h(\vec{P}), \ldots, k(\vec{Q}), \vec{Z} \right) : (\vec{F}, \ldots, \vec{G}, \vec{E}).}$$

(Here $\vec{F}_{\geq 1}$ means that $\vec{F}$ contains at least one type.) Eventually we will also incorporate shuffles, but we postpone that for now. Let us consider instead how to prevent duplication of derivations. In addition to our desired term

$$x : A, y : C \vdash (f(x), g(y)) : (B, D) \tag{4.1}$$

we must also be able to write both of the following:

$$x : A, y : C \vdash (f(x), y) : (B, C) \tag{4.2}$$
$$x : A, y : C \vdash (x, g(y)) : (A, D). \tag{4.3}$$

So how do we prevent ourselves from being able to apply the generator rule again to the latter two, obtaining two more derivations of the same morphism as (4.1)?

There are different possible choices that one could make here, potentially leading to different type theories and different "normal forms" for the morphisms in a free prop. The choice we will make is to force ourselves to "apply all functions as soon as possible". Thus, for instance, we forbid ourselves from applying g to y in (4.2) because we *could have* already applied it to produce (4.1). On the other hand, we will still allow ourselves to apply $h : (B, C) \to E$ in (4.2) to get

$$x : A, y : C \vdash (h(f(x), y)) : E$$

because h has f as an input, hence could not have been applied at the same time as f.

Making this precise is the purpose of designating some of the types in the consequent as **active**; recall that we denote the active types by $A^\star$. If $\vec{A}$ is a list of types, we write $\vec{A}^{\star \geq 1}$ to mean that at least one of the types in $\vec{A}$ is active, $\vec{A}^\star$ to mean that they are all active, and $\vec{A}^{\star = 0}$ to mean that none of them are active. As noted previously, we write $\vec{A}^{\star ?}$ to avoid specifying whether or not any of the types are active.

The identity rule will make all types active, while the generator rule makes only the outputs of the generators active. We then restrict the generator rule to require that at

least one of the *inputs* of each generator being applied must be active in the premise; this means that none of them could have been applied any sooner, since at least one of their arguments was just introduced by the previous rule. Thus, our desired derivation

$$\frac{\overline{x : A, y : C \vdash (x, y) : (A^\star, C^\star)}}{x : A, y : C \vdash (f(x), g(x)) : (B^\star, D^\star)}$$

is allowed, while the undesired one

$$\require{enclose} \quad \text{¿} \quad \frac{\dfrac{\overline{x : A, y : C \vdash (x, y) : (A^\star, C^\star)}}{x : A, y : C \vdash (f(x), y) : (B^\star, C)}}{x : A, y : C \vdash (f(x), g(y)) : (B, D)} \quad \text{?}$$

is not allowed, since in the attempted application of g the input type C is not active. Thus our generator rule now becomes

$$\frac{\Gamma \vdash (\vec{M}, \ldots, \vec{N}, \vec{P}, \ldots, \vec{Q}, \vec{R} \mid \vec{Z}) : (\vec{A}^{\star \geq 1}, \ldots, \vec{B}^{\star \geq 1}, \vec{C}^{\star \geq 1}, \ldots, \vec{D}^{\star \geq 1}, \vec{E}^{\star ?}) \quad \begin{matrix} f \in \mathcal{G}(\vec{A}; \vec{F}_{\geq 1}) & \cdots & g \in \mathcal{G}(\vec{B}; \vec{G}_{\geq 1}) \\ h \in \mathcal{G}(\vec{C};) & \cdots & k \in \mathcal{G}(\vec{D};) \end{matrix}}{\Gamma \vdash \left(\vec{f}(\vec{M}), \ldots, \vec{g}(\vec{N}), \vec{R} \mid \vec{h}(\vec{P}), \ldots, \vec{k}(\vec{Q}), \vec{Z} \right) : (\vec{F}^\star, \ldots, \vec{G}^\star, \vec{E}^{\star = 0}).}$$

Of course, this rule can now never apply to generators with nullary domain. Since these can always be applied at the very beginning, we incorporate them into the identity rule. Thus the identity rule is now

$$\frac{\begin{matrix} f \in \mathcal{G}(; \vec{B}_{\geq 1}) & \cdots & g \in \mathcal{G}(; \vec{C}_{\geq 1}) \\ h \in \mathcal{G}(;) & \cdots & k \in \mathcal{G}(;) \\ \mathfrak{a}, \ldots, \mathfrak{b} \in \mathfrak{A} \text{ and pairwise distinct} \end{matrix}}{\vec{x} : \vec{A} \vdash \left(\vec{x}, \vec{f}^{\mathfrak{a}}, \ldots, \vec{g}^{\mathfrak{b}} \mid h, \ldots, k \right) : (\vec{A}^\star, \vec{B}^\star, \ldots, \vec{C}^\star).}$$

Note the labels on the terms with nullary domain and positive-ary codomain, as promised.

Finally, if we want to make the exchange rule admissible, we have to build permutations into the rules as well. Each rule should add exactly the part of a permutation that can't be "pushed into the premises". Because we've formulated the generator rule so that the premise and conclusion have the same context, any desired permutation in the domain can be pushed all the way up to the identity rule. Thus, for the generator rule it remains to deal with permutation in the codomain.

The freedom we have in the premises of the generator rule is to (inductively) permute the types *within* each list $\vec{A}, \vec{B}, \vec{C}, \vec{D}, \vec{E}$, and also to block-permute the lists $\vec{A}, \ldots, \vec{B}$ and separately the lists $\vec{C}, \ldots, \vec{D}$ (with a corresponding permutation of the generators $f, \ldots, g$ and $h, \ldots, k$). (If we permuted the main premise any more than this, it would

$$\Gamma \vdash (\vec{M}, \ldots, \vec{N}, \vec{P}, \ldots, \vec{Q}, \vec{R} \mid \vec{Z}) : (\vec{A}^{\star \geq 1}, \ldots, \vec{B}^{\star \geq 1}, \vec{C}^{\star \geq 1}, \ldots, \vec{D}^{\star \geq 1}, \vec{E}^{\star ?})$$

$$\frac{\begin{array}{c} f \in \mathcal{G}(\vec{A}; \vec{F}_{\geq 1}) \quad \cdots \quad g \in \mathcal{G}(\vec{B}; \vec{G}_{\geq 1}) \\ h \in \mathcal{G}(\vec{C};) \quad \cdots \quad k \in \mathcal{G}(\vec{D};) \\ \sigma : (\vec{F}^{\star}, \ldots, \vec{G}^{\star}, \vec{E}^{\star = 0}) \xrightarrow{\sim} \Delta \text{ preserving activeness} \\ \sigma \text{ preserves the relative order of types in } \vec{E} \\ \sigma \text{ preserves the relative order of } F_1, \ldots, G_1 \quad \tau \in \text{Shuf}(h, \ldots, k; \vec{Z}) \end{array}}{\Gamma \vdash \left(\sigma \left(\vec{f}(\vec{M}), \ldots, \vec{g}(\vec{N}), \vec{R} \right) \mid \tau \left(h(\vec{P}), \ldots, k(\vec{Q}), \vec{Z} \right) \right) : \Delta}$$

$$\frac{\begin{array}{c} f \in \mathcal{G}(; \vec{B}_{\geq 1}) \quad \cdots \quad g \in \mathcal{G}(; \vec{C}_{\geq 1}) \\ h \in \mathcal{G}(;) \quad \cdots \quad k \in \mathcal{G}(;) \\ \mathfrak{a}, \ldots, \mathfrak{b} \in \mathfrak{A} \text{ and pairwise distinct} \\ \sigma : (\vec{A}^{\star}, \vec{B}^{\star}, \ldots, \vec{C}^{\star}) \xrightarrow{\sim} \Delta \text{ preserving activeness} \\ \sigma \text{ preserves the relative order of } B_1, \ldots, C_1 \end{array}}{\vec{x} : \vec{A} \vdash \left(\sigma \left(\vec{x}, \vec{f}^{\mathfrak{a}}, \ldots, \vec{g}^{\mathfrak{b}} \right) \mid h, \ldots, k \right) : \Delta}$$

Figure 6: Rules of the typing judgment

no longer have the requisite shape to apply the rule to.) Permutations of $\vec{C}, \ldots, \vec{D}$ don't do us any good in terms of permuting the codomain of the conclusion, but we can push permutations of $\vec{E}$ directly into the premise, and also a block-permutation of $\vec{F}, \ldots, \vec{G}$ into a block-permutation of $\vec{A}, \ldots, \vec{B}$.

What remains that we have to build into the rule can be described precisely by a permutation of $\vec{F}, \ldots, \vec{G}, \vec{E}$ that (1) preserves the relative order of the types in $\vec{E}$, and (2) preserves the relative order of the *first* types $F_1, \ldots, G_1$ in the lists $\vec{F}, \ldots, \vec{G}$. That is, any permutation of $\vec{F}, \ldots, \vec{G}, \vec{E}$ can be factored uniquely as one with these two properties followed by a block sum of a block-permutation of $\vec{F}, \ldots, \vec{G}$ with a permutation of $\vec{E}$. (The choice of the first types is arbitrary; we could just as well use the last types, etc.)

There is no real need to allow ourselves to permute the scalar terms, since semantically their order doesn't matter anyway. But it is convenient to allow ourselves to write the scalar terms in any order, so we incorporate permutations there too. The freedom in the premises allows us to permute the term in $\vec{Z}$ arbitrarily, and also to permute the terms $h, \ldots, k$ among themselves; thus what remains is precisely a shuffle. The final generator rule is therefore the first rule shown in Figure 6.

In the identity rule, the only useful freedom in the premises is to block-permute the $\vec{B}, \ldots, \vec{C}$. Thus what remains is a permutation that preserves the relative order of the first types $B_1, \ldots, C_1$. Any permutation in the scalar terms can be pushed into the premises, so we have the final rule shown second in Figure 6. Note that this also allows us to incorporate an arbitrary permutation in the domain.

Having introduced the auxiliary notion of "active types" to ensure that typing judgments have unique derivations, we now proceed to eliminate it. We start with the following observation. The notion of *subterm* is defined as usual; we write $M \equiv N$ to mean that M and N are syntactically the same term.

4.4. LEMMA. *If $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$ is derivable, and some M_i is a subterm of some M_j, then $i = j$ (hence $M_i \equiv M_j$).*

PROOF. By induction on the derivation. In an application of the identity rule, the non-scalar terms have no proper subterms, so the assumption means that $M_i \equiv M_j$. And since these terms are also uniquely identified by their label and subscript, we have $i = j$.

For an application of the generator rule, all the non-scalar terms either occur verbatim in the main premise, or are of the form $f_{(k)}(\vec{M})$ (including $f(\vec{M})$ when $k = 1$) where each M_i is a non-scalar term in the main premise and $|\vec{M}| \geq 1$. These two possibilities are mutually exclusive, and hence partition the terms into two classes that we call *old* and *new* respectively.

The inductive hypothesis takes care of the case when both terms are old. If a new term $f_{(k)}(\vec{M})$ is a subterm of an old term N, then each M_i is a proper subterm of N, contradicting the inductive hypothesis. This includes the case when an old term equals a new term; while if an old term N is a proper subterm of a new term $f_{(k)}(\vec{M})$, then it must be a subterm of some M_i, also contradicting the inductive hypothesis.

If one new term $f_{(k)}(\vec{M})$ is a proper subterm of another $g_{(\ell)}(\vec{N})$, then it must be a subterm of some N_j. Hence each M_i must be a proper subterm of N_j, contradicting the inductive hypothesis.

Finally, suppose two new terms are syntactically equal, $f_{(k)}(\vec{M}) \equiv g_{(\ell)}(\vec{N})$. Then we must have $f = g$, $k = \ell$, and $\vec{M} \equiv \vec{N}$. Note that $f = g$ means that f and g are the same function symbol (morphism in $\mathcal{G}$), but not *a priori* that they arise from the same generator *application* in the rule (a given instance of the generator rule could apply the same generator more than once). However, $\vec{M} \equiv \vec{N}$ and the inductive hypothesis ensure that they *do* arise from the same generator application. Together with $k = \ell$, this implies that they have the same place in the given judgment as well. ∎

We are primarily interested in applying Lemma 4.4 in the case of an "improper subterm" $M_i \equiv M_j$, but the stronger hypothesis makes the induction go through more easily.

4.5. REMARK. Semantically, it is not really necessary to label the morphisms in $\mathcal{G}(; B)$ with a symbol $\mathfrak{a} \in \mathfrak{A}$, since tensor products of such morphisms are invariant under permutation (because the swap on the unit object is the identity morphism). However, it would be significantly trickier to omit such labels syntactically. In practice, we can leave them off informally and trust the reader to put them back if needed.

4.6. THEOREM. *If there is some assignment of activeness to the types in Δ such that $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$ is derivable, then that assignment is unique, as is the derivation.*

PROOF. By the *height* of an occurrence of a variable or function symbol in a term, we mean its maximum distance to a leaf node in the abstract syntax tree representing the term. Thus the height of a variable is 0, and the height of an occurrence of a function symbol is the least natural number strictly greater than the heights of the head symbols of all its arguments. Note that a nullary function symbol always has height 0, while a function symbol applied to a positive number of variables alone has height 1.

We claim that in any derivable typing judgment, the terms associated to active types are precisely those non-scalar ones whose head symbol has maximum height. The proof is by induction on derivations. In the identity rule, all terms have height 0 and all types are active. Now consider the generator rule, and suppose inductively that the claim is true for the main premise, with maximal height n, say. Then since each of the new function symbols introduced by the rule is applied to at least one term from an active type, which therefore has the maximal height n, it must have height $n + 1$. It follows that the new maximum height is $n + 1$, and that these new symbols are precisely those of maximum height; but they are also precisely those associated to active types. This proves the claim.

It follows immediately that the terms uniquely determine the activeness of the types, since height is a syntactic invariant of the terms. Moreover, we can tell from the terms which rule must have been applied last (if the maximum height is 0, it must come from the identity rule; otherwise it must come from the generator rule) and which function symbols that rule must have introduced (those of maximum height).

In the case of the generator rule, we can also conclude that the new terms must be precisely those whose head generator symbol has maximum height. We divide these new terms into subsets that have the same generator symbol and arguments, so that each subset consists of terms $f_{(i)}(\vec{M})$ for fixed f and $\vec{M}$ but varying i. By Lemma 4.4, there can be no more than one occurrence of a particular term $f_{(i)}(\vec{M})$, so the subset of new terms determined by f and $\vec{M}$ must consist of exactly the m terms $f_{(i)}(\vec{M})$ for $1 \leq i \leq m$, where f has m outputs.

Now the ordering of these function symbol applications as $f, \ldots, g$ and $h, \ldots, k$ must be the order in which the corresponding $f_{(1)}, \ldots, g_{(1)}$ and $h, \ldots, k$ appear in the term list, since the permutations σ and τ preserve those orders. Then σ^{-1} is uniquely determined by the fact that it must place all the outputs of f first, and so on until all the outputs of g, then all the terms of non-maximum height in the same order that they were given in the conclusion. Similarly, τ^{-1} is uniquely determined by the fact that it has to place $h, \ldots, k$ first and the scalar terms of non-maximum height last, preserving internal order in each group. Finally, this determines the main premise uniquely as well.

The argument for the identity rule is similar, with no τ and with σ^{-1} placing all the variables first in the order of the context. Inductively, therefore, the entire derivation is uniquely determined. ∎

Note that we can regard this proof as an algorithm for "type-checking" a judgment without activeness annotations: first we use the recursive height function on terms to calculate the activeness, then we proceed as usual to recursively match against the generator or identity rules. Because of this theorem, in the future we will omit the activeness labels.
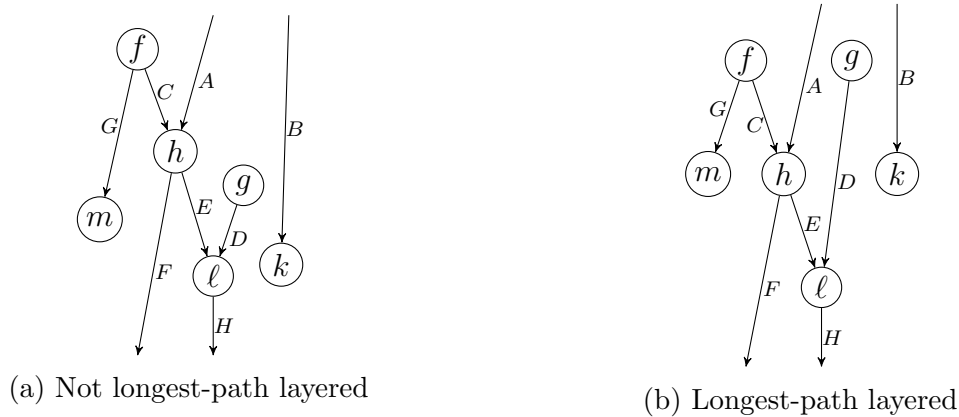
(a) Not longest-path layered   (b) Longest-path layered

Figure 7: String diagrams for a morphism in a free prop

Theorem 4.6 can be seen as giving a "normal form" for morphisms in a free prop. Intuitively, a composite of generating morphisms and permutations is in normal form if each generator or permutation is applied "as soon as possible"; the inductive definition of judgments with activeness makes precise exactly what that means.

Another perspective on this involves string diagrams. From the latter perspective, a morphism in the free prop generated by $\mathcal{G}$ can be represented by an *acyclic directed graph* whose vertices are labeled by generators and whose edges are labeled by objects. We allow "free edges" one or both of whose ends do not connect to any vertex, corresponding to the input and output objects of the morphism. For instance, such a graph based on generators $f \in \mathcal{G}(; G, C)$, $g \in \mathcal{G}(; D)$, $h \in \mathcal{G}(C, A; F, E)$, $k \in \mathcal{G}(B;)$, $\ell \in \mathcal{G}(E, D; H)$, and $m(G;)$ is shown in Figure 7a. The corresponding term in our type theory is

$$x : A, y : B \vdash (h_{(1)}(f_{(2)}, x), \ell(h_{(2)}(f_{(2)}, x), g) \mid m(f_{(1)}), k(y)) : (F, H)$$

Applying the algorithm of Theorem 4.6, we first calculate the heights of each generator:

$$f : 0 \qquad g : 0 \qquad h : 1 \qquad k : 1 \qquad \ell : 2 \qquad m : 1$$

Therefore, H is the only active type, and the final rule application must have been a generator rule applying only ℓ:

$$\frac{x : A, y : B \vdash (h_{(2)}(f_{(2)}, x), g, h_{(1)}(f_{(2)}, x) \mid m(f_{(1)}), k(y)) : (E^\star, D, F^\star) \qquad \ell \in \mathcal{G}(E, D; H)}{x : A, y : B \vdash (h_{(1)}(f_{(2)}, x), \ell(h_{(2)}(f_{(2)}, x), g) \mid m(f_{(1)}), k(y)) : (F, H^\star)}. \tag{4.7}$$

In the premise of this rule, E and F are the active types since h has height 1. But k and m also have height 1, so they must also be applied by the generator rule leading to this judgment:

$$\frac{\begin{array}{c} x : A, y : B \vdash (f_{(2)}, x, f_{(1)}, y, g) : (C^\star, A^\star, G^\star, B^\star, D^\star) \\ h \in \mathcal{G}(C, A; F, E) \\ m \in \mathcal{G}(G;) \qquad k \in \mathcal{G}(B;) \end{array}}{x : A, y : B \vdash (h_{(2)}(f_{(2)}, x), g, h_{(1)}(f_{(2)}, x) \mid m(f_{(1)}), k(y)) : (E^\star, D, F^\star)}. \tag{4.8}$$

Now in the premise all the types are active and all terms have height 0, so it must have arisen from the identity rule:

$$\frac{f \in \mathcal{G}(;G,C) \qquad g \in \mathcal{G}(;D)}{x : A, y : B \vdash (f_{(2)}, x, f_{(1)}, y, g) : (C^\star, A^\star, G^\star, B^\star, D^\star)}. \tag{4.9}$$

Thus we have a complete typing derivation:

$$\frac{\dfrac{f \in \mathcal{G}(;G,C) \qquad g \in \mathcal{G}(;D)}{x : A, y : B \vdash (f_{(2)}, x, f_{(1)}, y, g) : (C^\star, A^\star, G^\star, B^\star, D^\star)} \quad h \in \mathcal{G}(C, A; F, E) \qquad m \in \mathcal{G}(G;) \qquad k \in \mathcal{G}(B;)}{\dfrac{x : A, y : B \vdash (h_{(2)}(f_{(2)}, x), g, h_{(1)}(f_{(2)}, x) \mid m(f_{(1)}), k(y)) : (E^\star, D, F^\star) \qquad \ell \in \mathcal{G}(E, D; H)}{x : A, y : B \vdash (h_{(1)}(f_{(2)}, x), \ell(h_{(2)}(f_{(2)}, x), g) \mid m(f_{(1)}), k(y)) : (F, H^\star)}}.$$

This derivation corresponds to the redrawing of the string diagram shown in Figure 7b, in which the vertices appear in horizontal layers, with the edges always pointing down the page, and each vertex placed in the highest layer possible. The topmost layer with f and g corresponds to the identity rule application (4.9), the next layer with h, m, k corresponds to the generator rule application (4.8), and the final layer with ℓ corresponds to (4.7).

The referee has pointed out that this is essentially the "longest path layering" [EL89] of an acyclic directed graph (modulo a suitable correction to deal with vertexless half-edges). The proof of Theorem 4.6 can thus equivalently be interpreted as computing a normal form for a morphism in a free prop by expressing it as an acyclic directed graph and rearranging it according to its longest path layering, and also making certain canonical choices regarding the ordering of vertices along each layer. It is natural to expect that other canonical ways of drawing a directed graph might correspond to other normal forms and hence other type theories for free props, but we will not investigate this here.

It remains to consider the equality judgment. We don't have a traditional form of α-conversion since we have no bound variables as such, but as remarked in section 1 the labels $\mathfrak{a} \in \mathfrak{A}$ can sometimes be regarded as playing a similar role, and in particular the choice of concrete labels must not matter. We also need to impose invariance under permutation of the scalar terms. It may seem silly to have incorporated permutations in the scalar terms earlier and yet quotient out by that freedom now, but such an equality rule would be necessary even if we hadn't incorporated any permutations to start with. The paradigmatic case is when we have two nullary scalar generators $f : () \to ()$ and $g : () \to ()$, leading unavoidably to two distinct valid terms

$$\vdash (\mid f, g) : () \qquad\qquad \vdash (\mid g, f) : ()$$

that must be equal in a monoidal category (since the monoid of endomorphisms of the unit object is commutative). It is probably no coincidence that this is also the case where interesting things happen upon categorification.

Combining these two permutation rules, we obtain the rule shown in Figure 8. When we consider *presented* props in section 6, with equational theories, we will need the usual reflexivity, symmetry, transitivity, and congruence rules for equality, but with only this rule we can omit them since permutations are already a group. And since we have no type forming operations, there are no β or η rules.

This completes our **type theory for the free prop generated by** $\mathcal{G}$.

$$\frac{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) : \Delta \qquad \rho \in S_n \qquad \sigma : \mathfrak{A} \cong \mathfrak{A}}{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) = (\vec{M}^\sigma \mid Z^\sigma_{\rho 1}, \ldots, Z^\sigma_{\rho n}) : \Delta}$$

Figure 8: Rule of the axiom-free equality judgment

5. Constructing free props from type theory

We now proceed to show that our type theory has the structure of a prop, beginning with the admissibility of exchange on the right.

5.1. PROPOSITION. *If $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$ is derivable and ρ is a permutation of Δ, then $\Gamma \vdash (\rho \vec{M} \mid \vec{Z}) : \rho \Delta$ is also derivable. Moreover, this action is functorial.*

PROOF. This essentially follows from how we built the rules. If the derivation ends with the identity rule, then we can compose ρ with the specified permutation σ from that rule, and reorder the generators $f, \ldots, g$ in the rule according to the order that $\rho \sigma$ puts them in. If the derivation ends with the generator rule, then we similarly compose ρ with σ, reorder the generators $f, \ldots, g$, and inductively push the remaining part of the permutation (that acting on the non-active terms) into the main premise. Functoriality follows immediately. ∎

For admissibility of composition/substitution, it seems helpful to first prove the admissibility of a single-generator rule. Note that we formulate it with the domain of the generator at the *end* of the given codomain context.

5.2. PROPOSITION. *If $\Gamma \vdash (\vec{M}, \vec{N} \mid \vec{Z}) : \Delta, \vec{A}$ is derivable and $f \in \mathcal{G}(\vec{A}; \vec{B})$, then $\Gamma \vdash (\vec{M}, \vec{f}(\vec{N}) \mid \vec{Z}) : \Delta, \vec{B}$ is derivable. Moreover, if none of the types in $\vec{A}$ are active in the given derivation, then all of the types in Δ that are active in the given derivation are still active in the result.*

PROOF. If any of the types in $\vec{A}$ are active, we can simply apply the generator rule with f as the only generator. Otherwise, none of them were introduced by the final rule in the given derivation. If that final rule was the identity rule, then $\vec{A}$ must be empty (since all types in the conclusion of the identity rule are active), so f has nullary domain and we can just add it to that application of the identity rule.

If the final rule in the given derivation was the generator rule, then $\vec{A}$ must also appear at the end of its main premise. If none of the types in $\vec{A}$ are active therein, then we can inductively apply f to that premise; by the second clause of the inductive hypothesis, this does not alter the activeness of the other types in the premise, so we can re-apply the generator rule. Finally, if at least one of the types in $\vec{A}$ *is* active in the main premise, then we can add f to the generator rule, applying it alongside all the other generators, since it satisfies the condition that at least one of its arguments be active. (Technically, this may require us to first permute the consequent of the main premise so that $\vec{A}$ appears before all the other non-inputs to the generator rule. This is not a problem for the induction

since in this case we are not actually using the inductive hypothesis at all.) In all cases, the second claim of the lemma is obvious. ∎

Now by combining Proposition 5.1 and Proposition 5.2, we can postcompose with a generator $f \in \mathcal{G}(\vec{A}; \vec{B})$ whose domain types $\vec{A}$ appear anywhere in the consequent of a judgment $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$, in any order.

5.3. PROPOSITION. *Substitution is admissible: if* $\Gamma \vdash (\vec{M} \mid \vec{Y}) : \Delta$ *and* $\Delta \vdash (\vec{N} \mid \vec{Z}) : \Phi$ *are derivable, then so is* $\Gamma \vdash (\vec{N}[\vec{M}/\Delta] \mid \vec{Z}[\vec{M}/\Delta], \vec{Y}) : \Phi$.

PROOF. We induct on the derivation of $\Delta \vdash (\vec{N} \mid \vec{Z}) : \Phi$. If it comes from the identity rule, then we just have to compose $\Gamma \vdash (\vec{M} \mid \vec{Y}) : \Delta$ with some number of nullary-domain generators and permute its codomain; we do this one by one using Proposition 5.2 and then Proposition 5.1. Similarly, if it comes from the generator rule, we inductively compose with its main premise, then apply all of the new generators one by one using Proposition 5.2. ∎

As an example, suppose we want to compose the following terms:

$$x : A, y : B \vdash (f_{(1)}(y), k(g, f_{(3)}(y)), f_{(2)}(y) \mid h(x)) : (C, D, E) \tag{5.4}$$

$$u : C, v : D, w : E \vdash (m(u, \ell_{(2)}(w)), s, \ell_{(1)}(w) \mid n(v)) : (F, G, H) \tag{5.5}$$

Here the generators are

$$f : B \to (C, E, P) \qquad g : () \to Q \qquad h : A \to () \qquad k : (Q, P) \to D \qquad \ell : E \to (H, R)$$

$$m : (C, R) \to F \qquad\qquad n : D \to () \qquad\qquad s : () \to G$$

The heights are

$$f = 1 \qquad g = 0 \qquad h = 1 \qquad k = 2 \qquad \ell = 1 \qquad m = 2 \qquad n = 1 \qquad s = 0$$

Thus, the final rule of the second derivation must apply m only, so our inductive job is to compose (5.4) with

$$u : C, v : D, w : E \vdash (u, \ell_{(2)}(w), s, \ell_{(1)}(w) \mid n(v)) : (C, R, G, H) \tag{5.6}$$

Now the final rule of the second derivation must apply ℓ and n together, so our inductive job is to compose (5.4) with

$$u : C, v : D, w : E \vdash (w, v, u, s \mid) : (E, D, C, G) \tag{5.7}$$

The latter is obtained from the identity rule, so our task is now to apply Proposition 5.2 to the former and the single generator $s : () \to G$. Peeling down the derivation of the former, we obtain

$$x : A, y : B \vdash (g, f_{(3)}(y), f_{(1)}(y), f_{(2)}(y) \mid h(x)) : (Q, P, C, E)$$

and then

$$x : A, y : B \vdash (y, x, g \mid) : (B, A, Q)$$

which is also obtained from the identity rule. The identity rule can therefore also give us

$$x : A, y : B \vdash (y, x, g, s \mid) : (B, A, Q, G).$$

Re-applying f, h and then k, we obtain

$$x : A, y : B \vdash (g, f_{(3)}(y), f_{(1)}(y), f_{(2)}(y), s \mid h(x)) : (Q, P, C, E, G)$$

and then

$$x : A, y : B \vdash (f_{(1)}(y), k(g, f_{(3)}(y)), f_{(2)}(y), s \mid h(x)) : (C, D, E, G).$$

Permuting this, we obtain

$$x : A, y : B \vdash (f_{(2)}(y), f_{(1)}(y), k(g, f_{(3)}(y)), s \mid h(x)) : (E, C, D, G).$$

as the result of composing (5.4) and (5.7).

Backing out the induction one more step, we must apply ℓ and n to this using Proposition 5.2. We cannot apply ℓ directly since its domain E is not active (its term $f_{(2)}(y)$ has height 1 while the maximum height is 2). Thus, we back up to the main premise

$$x : A, y : B \vdash (g, f_{(3)}(y), f_{(1)}(y), f_{(2)}(y), s \mid h(x)) : (Q, P, C, E, G)$$

in which E is active. Thus, we can apply ℓ in the same generator rule as k, obtaining

$$x : A, y : B \vdash (\ell_{(1)}(f_{(2)}(y)), \ell_{(2)}(f_{(2)}(y)), f_{(1)}(y), k(g, f_{(3)}(y)), s \mid h(x)) : (H, R, C, D, G).$$
$$(5.8)$$

Now the domain D of the generator n *is* active, so we can directly apply it with another generator rule, obtaining (after permutation)

$$x : A, y : B \vdash (f_{(1)}(y), \ell_{(2)}(f_{(2)}(y)), s, \ell_{(1)}(f_{(2)}(y)) \mid n(k(g, f_{(3)}(y))), h(x)) : (C, R, G, H).$$
$$(5.9)$$

as the result of composing (5.4) and (5.6).

Finally, we must compose this with m using Proposition 5.2. Neither of the domain types C and R is active in (5.9) (in fact, *no* types are active in (5.9), since the last rule applied was a generator rule whose only generator has nullary codomain), so we have to inductively peel back to (5.8) in which R is active (though not C). Thus, we can then apply m in the same generator rule as n, obtaining

$$x : A, y : B \vdash (m(f_{(1)}(y), \ell_{(2)}(f_{(2)}(y))), s, \ell_{(1)}(f_{(2)}(y)) \mid n(k(g, f_{(3)}(y))), h(x)) : (F, G, H)$$
$$(5.10)$$

as our end result.

Note that the terms in (5.10) are indeed the result of substituting $f_{(1)}(y)$ for u, $k(g, f_{(3)}(y))$ for v, and $f_{(2)}(y)$ for w (the terms appearing in (5.4)) in the terms of (5.5), and appending the scalar term $h(x)$ of (5.4) to the scalar terms of (5.5):

$$m(u, \ell_{(2)}(w))[f_{(1)}(y)/u, k(g(f_{(3)}(y)))/v, f_{(2)}(y)/w] = m(f_{(1)}(y), \ell_{(2)}(f_{(2)}(y)))$$
$$s[f_{(1)}(y)/u, k(g(f_{(3)}(y)))/v, f_{(2)}(y)/w] = s$$
$$\ell_{(1)}(w)[f_{(1)}(y)/u, k(g(f_{(3)}(y)))/v, f_{(2)}(y)/w] = \ell_{(1)}(f_{(2)}(y))$$
$$n(v)[f_{(1)}(y)/u, k(g(f_{(3)}(y)))/v, f_{(2)}(y)/w] = n(k(g(f_{(3)}(y)))).$$

The only choice involved in the proof of Proposition 5.2 is in how to order the scalar terms in the result. We adopted the convention that those associated to the terms being substituted into come first, followed by those associated to the terms being substituted. The opposite convention would do as well for the following theorem:

5.11. PROPOSITION. *Composition is associative and unital.*

PROOF. Since derivations are determined uniquely by their terms by Theorem 4.6, this follows from the evident associativity and unitality of substitution into terms, and the associativity and unitality of concatenation of lists of scalar terms. ∎

Thus we have a category whose objects are contexts and whose morphisms are derivable judgments $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$. However, this is not quite the underlying category of our prop: we must quotient it by the equality rule from Figure 8:

$$\frac{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) : \Delta \qquad \rho \in S_n \qquad \sigma : \mathfrak{A} \cong \mathfrak{A}}{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) = (\vec{M}^\sigma \mid Z_{\rho 1}^\sigma, \ldots, Z_{\rho n}^\sigma) : \Delta} \tag{5.12}$$

(which also ensures that the choice of ordering in Proposition 5.3 is irrelevant). For this we need the evident observation:

5.13. PROPOSITION. *The equality rule (5.12) is a congruence on the category of contexts and derivable typing judgments. That is, it is an equivalence relation on the morphisms that is preserved by composition on both sides.* ∎

Therefore, the quotient by this equality judgment is again a category whose objects are the contexts (i.e. finite lists of types).

5.14. THEOREM. *The contexts and derivable term judgments in the type theory for the free prop generated by $\mathcal{G}$, modulo the equality rule (5.12), form a symmetric strict monoidal category.*

PROOF. The monoidal structure on contexts is concatenation, with the empty context as unit. To tensor morphisms, it is easiest to first tensor with identities: given $\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta$, we construct $\Gamma, \vec{x} : \vec{A} \vdash (\vec{M}, \vec{x} \mid \vec{Z}) : \Delta, \vec{A}$ by inducting until we get down to the identity rule and then adding the variables $\vec{x} : \vec{A}$ to the context. Now we obtain the tensor product of $\Gamma \vdash (\vec{M} \mid \vec{Y}) : \Delta$ and $\Phi \vdash (\vec{N} \mid \vec{Z}) : \Psi$ by first tensoring with identities to

get $\Gamma, \Phi \vdash (\vec{M}, \Gamma \mid \vec{Y}) : \Delta, \Phi$ and $\Delta, \Phi \vdash (\Delta, \vec{N} \mid \vec{Z}) : \Delta, \Psi$ and then composing to get $\Gamma, \Phi \vdash (\vec{M}, \vec{N} \mid \vec{Z}, \vec{Y}) : \Delta, \Psi$. If we did this in the other order, we would get $(\vec{M}, \vec{N} \mid \vec{Y}, \vec{Z})$ instead, which is equal by (5.12). In particular, this implies functoriality of the tensor product; associativity and unitality follow similarly. Finally, the symmetry isomorphism is $\vec{x} : \vec{A}, \vec{y} : \vec{B} \vdash (\vec{y}, \vec{x}) : \vec{B}, \vec{A}$; it is easy to verify the axioms. ∎

Thus we have a prop, which we denote $\mathfrak{F}\mathcal{G}$.

5.15. THEOREM. $\mathfrak{F}\mathcal{G}$ *is the free prop generated by* $\mathcal{G}$.

PROOF. Let $\mathcal{P}$ be a prop and $\omega : \mathcal{G} \to \mathcal{P}$ a morphism of signatures. We extend it to a morphism of props $\mathfrak{F}\mathcal{G} \to \mathcal{P}$ by induction on derivations. By the coherence theorem for symmetric monoidal categories (e.g. [ML98, Chapter XI]), there is a unique choice at each step if we are to have a (symmetric strict monoidal) functor, and likewise both equality rules corresponds to actual equalities that must hold in $\mathcal{P}$. Then we prove that this actually is a symmetric strict monoidal functor, using the definition of composition and the tensor product in $\mathfrak{F}\mathcal{G}$. ∎

5.16. REMARK. Since the free prop generated by a signature is unique up to isomorphism, it follows that our $\mathfrak{F}\mathcal{G}$ is isomorphic to the free prop on $\mathcal{G}$ presented in any other way, such as by using string diagrams whose edges and vertices are labeled by objects and morphisms of $\mathcal{G}$ respectively. Unsurprisingly, this correspondence can be made more explicit: from any derivable $\Gamma \vdash \vec{M} : \Delta$ we can construct a labeled string diagrams from Γ to Δ, whose vertices are the generator applications and whose edges are the disjoint union of the variables in Γ and the terms $f_{(i)}(\vec{N})$ appearing as subterms of $\vec{M}$.

6. Presentations of props

Since the category of props is monadic over the category of signatures by Theorem 2.5, every prop $\mathcal{P}$ has a *presentation* in terms of signatures, i.e. a coequalizer diagram

$$\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G} \to \mathcal{P}.$$

Moreover, since $\mathfrak{F}$ and its right adjoint are the identity on the set of objects, we may assume that $\mathcal{R}$ and $\mathcal{G}$ both have the same set of objects as $\mathcal{P}$ and all the morphisms are the identity on objects. Now by the universal property of $\mathfrak{F}\mathcal{R}$, the two morphisms of props $\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G}$ are equivalently morphisms of signatures $\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G}$. Thus, once $\mathcal{G}$ is given, the additional data of $\mathcal{R}$ consists of a set of pairs of parallel morphisms in $\mathfrak{F}\mathcal{G}$, which is to say a set of **equality axioms** of the form

$$\Gamma \vdash (\vec{M} \mid \vec{Y}) = (\vec{N} \mid \vec{Z}) : \Delta$$

where both $\Gamma \vdash (\vec{M} \mid \vec{Y}) : \Delta$ and $\Gamma \vdash (\vec{N} \mid \vec{Z}) : \Delta$ are derivable in the type theory for the free prop generated by $\mathcal{G}$. We obtain the **type theory for the prop presented by** $(\mathcal{G}, \mathcal{R})$ by augmenting the type theory for the free prop on $\mathcal{G}$ by these axioms for the

$$\frac{\Gamma \vdash (\vec{M} \mid \vec{Z}) : \Delta}{\Gamma \vdash (\vec{M} \mid \vec{Z}) = (\vec{M} \mid \vec{Z}) : \Delta} \qquad \frac{\Gamma \vdash (\vec{M} \mid \vec{Y}) = (\vec{N} \mid \vec{Z}) : \Delta}{\Gamma \vdash (\vec{N} \mid \vec{Z}) = (\vec{M} \mid \vec{Y}) : \Delta}$$

$$\frac{\Gamma \vdash (\vec{M} \mid \vec{X}) = (\vec{N} \mid \vec{Y}) : \Delta \qquad \Gamma \vdash (\vec{N} \mid \vec{Y}) = (\vec{P} \mid \vec{Z}) : \Delta}{\Gamma \vdash (\vec{M} \mid \vec{X}) = (\vec{P} \mid \vec{Z}) : \Delta}$$

$$\frac{\Gamma \vdash (\vec{M} \mid \vec{X}) = (\vec{N} \mid \vec{Y}) : \Delta \qquad \Delta \vdash (\vec{P} \mid \vec{Z}) : \Phi}{\Gamma \vdash (\vec{P}[\vec{M}/\Delta] \mid \vec{Z}[\vec{M}/\Delta], \vec{X}) = (\vec{P}[\vec{N}/\Delta] \mid \vec{Z}[\vec{N}/\Delta], \vec{Y}) : \Phi}$$

$$\frac{\Gamma \vdash (\vec{M} \mid \vec{X}) : \Delta \qquad \Delta \vdash (\vec{N} \mid \vec{Y}) = (\vec{P} \mid \vec{Z}) : \Phi}{\Gamma \vdash (\vec{N}[\vec{M}/\Delta] \mid \vec{Y}[\vec{M}/\Delta], \vec{X}) = (\vec{P}[\vec{M}/\Delta] \mid \vec{Z}[\vec{M}/\Delta], \vec{X}) : \Phi}$$

$$\frac{\Gamma \vdash (\vec{M} \mid \vec{X}) = (\vec{N} \mid \vec{Y}) : \Delta \qquad \Phi \vdash (\vec{P} \mid \vec{Z}) = (\vec{Q} \mid \vec{W}) : \Psi}{\Gamma, \Phi \vdash (\vec{M}, \vec{P} \mid \vec{X}, \vec{Z}) = (\vec{N}, \vec{Q} \mid \vec{Y}, \vec{W}) : \Delta, \Psi}$$

$$\frac{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) : \Delta \qquad \rho \in S_n \qquad \sigma : \mathfrak{A} \cong \mathfrak{A}}{\Gamma \vdash (\vec{M} \mid Z_1, \ldots, Z_n) = (\vec{M}^\sigma \mid Z_{\rho 1}^\sigma, \ldots, Z_{\rho n}^\sigma) : \Delta}$$

Figure 9: Rules of the equality judgment in the presence of axioms

equality judgment, together with the additional rules shown in Figure 9 (which are no longer automatic in the presence of such axioms).

The first three rules in Figure 9 are the usual reflexivity,[5] symmetry, and transitivity. The next three are congruence rules for precomposition, postcomposition, and the concatenation product. The final one is the label-renaming and scalar-permutation rule from Figure 8. Note that congruence for pre- and post-composition includes congruence under permutations of the domain and codomain.

6.1. PROPOSITION. *For any prop presentation $(\mathcal{G}, \mathcal{R})$, the equality judgment generated by the axioms of $\mathcal{R}$ together with the rules of Figure 9 is a congruence on the prop $\mathfrak{F}\mathcal{G}$. That is, it is an equivalence relation on morphisms preserved by composition and tensor product.*

PROOF. The rules of Figure 9 essentially force this to be true.   ∎

Thus we obtain a prop $\mathfrak{F}\langle\mathcal{G}|\mathcal{R}\rangle$ as the quotient of $\mathfrak{F}\mathcal{G}$ by this congruence.

6.2. THEOREM. $\mathfrak{F}\langle\mathcal{G}|\mathcal{R}\rangle$ *is the prop presented by $(\mathcal{G}, \mathcal{R})$. That is, we have a coequalizer diagram in the category of props:*

$$\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G} \to \mathfrak{F}\langle\mathcal{G}|\mathcal{R}\rangle.$$

PROOF. Since the quotient map $\mathfrak{F}\mathcal{G} \to \mathfrak{F}\langle\mathcal{G}|\mathcal{R}\rangle$ is surjective, a prop morphism $\omega : \mathfrak{F}\mathcal{G} \to \mathcal{P}$ factors through $\mathfrak{F}\langle\mathcal{G}|\mathcal{R}\rangle$ in at most one way. Moreover, it does so precisely when it identifies all pairs of tuples of terms that are identified by the equality judgment. However, the equality rules in Figure 9 are satisfied in any prop, so this happens precisely when ω respects the axioms of $\mathcal{R}$, i.e. when it coequalizes the two maps $\mathfrak{F}\mathcal{R} \rightrightarrows \mathfrak{F}\mathcal{G}$.   ∎

Thus, we can use our type theory to reason about structures in arbitrary props defined by generators and axioms, and hence also in symmetric monoidal categories (by Theorem 2.3).

7. Examples

7.1. DUALS AND TRACES. We begin by repeating the first example from the introduction more carefully. The **free prop generated by a dual pair** has a generating signature $\mathcal{G}$ with two objects A and A^* and two morphisms $\eta : () \to (A, A^*)$ and $\varepsilon : (A^*, A) \to ()$, and a signature of relations $\mathcal{R}$ that imposes two axioms

$$x : A \vdash (\eta_{(1)} \mid \varepsilon(\eta_{(2)}, x)) = x : A \qquad\qquad y : A^* \vdash (\eta_{(2)} \mid \varepsilon(y, \eta_{(1)})) = y : A^*.$$

A map from this prop to a symmetric monoidal category then reduces to the usual notion of dual.

---

[5]Actually, it is not necessary to assert reflexivity explicitly, since it is a special case of the permutation rule.

As suggested in the introduction, we write $M \triangleleft N$ for $\varepsilon(M, N)$, and $(u, \lambda^A u) : (A, A^*)$ for $(\eta_{(1)}, \eta_{(2)})$, where u is a label (i.e. an element of $\mathfrak{A}$) rather than a variable (appearing in the context). In this notation, the axioms are

$$x : A \vdash (u \mid \lambda^A u \triangleleft x) = x : A \qquad\qquad y : A^* \vdash (\lambda^A u \mid y \triangleleft u) = y : A^*.$$

Recall that $=$ is a congruence for substitution on both sides. Thus the first axiom means that if $\lambda^A u \triangleleft M$ appears in the scalars, for *any term* $M : A$ not involving u, then it can be removed by replacing u (wherever it appears, even as a subterm of some other term) with M. This justifies regarding it as a sort of "β-reduction for duality" with u playing the role of a "bound variable", although as we noted in the introduction the "binder" $\lambda^A u$ does not delimit the "scope" of u at all. Running this rule in reverse, we see that any term $M : A$ (appearing even as a sub-term of some other term) can be replaced by u, for a fresh label u, if we simultaneously add $\lambda^A u \triangleleft M$ to the scalars. The other axiom is similar; we may regard it as an "η-reduction for duality".

If A has a dual A^*, and $f : A \to A$, the **trace** of f is the composite

$$() \xrightarrow{\eta} (A, A^*) \xrightarrow{(f, \mathrm{id})} (A, A^*) \overset{\cong}{\Rightarrow} (A^*, A) \xrightarrow{\varepsilon} ()$$

In our type theory this is

$$() \vdash (\mid \lambda^A u \triangleleft f(u)) : ().$$

As advertised in the introduction, we can now prove cyclicity of the trace with a β-expansion followed by a β-reduction: for morphisms $f : A \to B$ and $g : B \to A$, with A and B dualizable, we have

$$
\begin{aligned}
\mathrm{tr}(fg) &\overset{\mathrm{def}}{=} (\mid \lambda^B y \triangleleft f(g(y))) \\
&= (\mid \lambda^B y \triangleleft f(x), \lambda^A x \triangleleft g(y)) \\
&= (\mid \lambda^A x \triangleleft g(f(x))) \\
&\overset{\mathrm{def}}{=} \mathrm{tr}(gf).
\end{aligned}
$$

More generally, any $f : (Y, A) \to (Z, A)$ has a "partial" or "twisted" trace

$$y : Y \vdash (f_{(1)}(y, u) \mid \lambda^A u \triangleleft f_{(2)}(y, u)) : Z.$$

This satisfies a version of cyclicity [PS14, Lemma 4.4]: for any morphisms $f : (Y, A) \to (Z, B)$ and $g : (W, B) \to (X, A)$, with A and B dualizable, we have

$$
\begin{aligned}
y : Y, w : W \vdash\ &(g_{(1)}(w, v), f_{(1)}(y, g_{(2)}(w, v)) \mid \lambda^B v \triangleleft f_{(2)}(y, g_{(2)}(w, v))) \\
&= (g_{(1)}(w, v), f_{(1)}(y, u) \mid \lambda^B v \triangleleft f_{(2)}(y, u), \lambda^A u \triangleleft g_{(2)}(w, v)) \\
&= (g_{(1)}(w, f_{(2)}(y, u)), f_{(1)}(y, u) \mid \lambda^A u \triangleleft g_{(2)}(w, f_{(2)}(y, u))) \\
&\quad : (X, Z).
\end{aligned}
$$

This general cyclicity includes in particular the *sliding* axiom for traces from [JSV96]. Their other axioms become simply syntactic identities in our type theory. For instance, *tightening* is the statement that if we compose $f : (Y, A) \to (Z, A)$ with $u : X \to Y$ and $v : Z \to W$ and then take its trace:

$$\frac{\dfrac{x : X \vdash u(x) : Y \qquad y : Y, a : A \vdash (f_{(1)}(y, a), f_{(2)}(y, a)) : (Z, A) \qquad z : Z \vdash v(z) : W}{x : X, a : A \vdash (v(f_{(1)}(u(x), a)), f_{(2)}(u(x), a)) : (W, A)}}{x : A \vdash (v(f_{(1)}(u(x), a)) \mid \lambda^A a \triangleleft f_{(2)}(u(x), a)) : W}$$

we get the same result as if we first take the trace of f and then compose with u and v:

$$\frac{x : X \vdash u(x) : Y \qquad \dfrac{y : Y, a : A \vdash (f_{(1)}(y, a), f_{(2)}(y, a)) : (Z, A)}{y : Y \vdash (f_{(1)}(y, a) \mid \lambda^A a \triangleleft f_{(2)}(y, a)) : Z} \qquad z : Z \vdash v(z) : W}{x : A \vdash (v(f_{(1)}(u(x), a)) \mid \lambda^A a \triangleleft f_{(2)}(u(x), a)) : W.}$$

Since the terms concluding both derivations are the same, they represent the same morphism. Of course, these are not actually derivations in our type theory, since they use the admissible rule of substitution. The uniqueness of typing derivations means that if the substitutions are eliminated according to Proposition 5.3 we obtain the same result in both cases, which in this case is:

$$\frac{\dfrac{x : A \vdash (u(x), a, \lambda^A a) : (Y, A, A^*)}{x : A \vdash (f_{(1)}(u(x), a), \lambda^A a, f_{(2)}(u(x), a)) : (Z, A^*, A)}}{x : A \vdash (v(f_{(1)}(u(x), a)) \mid \lambda^A a \triangleleft f_{(2)}(u(x), a)) : W.}$$

7.2. REMARK. Given any signature $\mathcal{G}$, we can augment it by adding a specified dual A^* for each object $A \in \mathcal{G}$ along with duality data as above. This yields a new signature $\mathcal{G}^*$ such that $\mathfrak{F}\mathcal{G}^*$ is freely generated by $\mathcal{G}$ together with a specified dual for each of its object. Thus $\mathfrak{F}\mathcal{G}^*$ is the free **compact closed prop** (i.e. prop in which every object has a dual) generated by $\mathcal{G}$. Any signature $\mathcal{R}$ of relations for $\mathcal{G}$ carries over to $\mathcal{G}^*$ as well, so we can construct presented compact closed props as well, and thereby presented compact closed monoidal categories.

In general, a prop presentation may not have any decision procedure for equality or normal forms for morphisms. However, the view of the zigzag equalities as "β and η reductions" suggests that this should be the case for some classes of presented compact closed props (whenever the equalities other than the zigzag identities can be controlled). If true, this should include in particular the explicit description of the free compact closed monoidal category on an ordinary category from [KL80].

Note also that by the "Int-construction" [JSV96], any *traced* symmetric monoidal category embeds fully-faithfully and trace-preservingly in a compact closed one. Thus, we can also use presented compact closed props to reason about traced symmetric monoidal categories.

7.3. WELL-IDEMPOTENT DUALIZABLE OBJECTS ARE SELF-DUAL. Tim Campion asked in [Cam17] whether an idempotent dualizable object in a symmetric monoidal category must be self-dual. A proof using string diagrams that this holds assuming "well-idempotence" was given by the user "MTyson"; here we recast this proof in our type theory.

We assume given one type X with a dual X^*, expressed as before, and a morphism $i : () \to X$ such that $1_X \otimes i : X \to X \otimes X$ is an isomorphism (such an i is what makes X *well-idempotent*). In our type theory, the latter can be expressed by a term $\vdash i : X$ and a morphism $x : X, y : X \vdash f(x, y) : X$ (the inverse to $1_X \otimes i$) such that $f(x, i) = x$ and $(x, y) = (f(x, y), i)$. More precisely, the latter two equalities are

$$x : X \vdash f(x, i) = x : X$$
$$x : X, y : X \vdash (x, y) = (f(x, y), i) : (X, X)$$

but we tend to omit the contexts and types in equality axioms and calculations when they are obvious.

Note that the equation $f(x, i) = x$ means that i is a "right unit" for the "binary operation" f. We now observe that it is also a left unit:

$$f(i, y) = (f(u, y) \mid \lambda^X u \triangleleft i)$$
$$= (u \mid \lambda^X u \triangleleft y)$$
$$= y.$$

Here the first line is a β-expansion and the third is a β-reduction. The second line uses the equality $(x, y) = (f(x, y), i)$ in the following way: first we introduce an extra variable to get

$$x : X, w : X^*, y : X \vdash (x, w, y) = (f(x, y), w, i) : (X, X^*, X)$$

then we precompose with

$$y : X \vdash (u, \lambda^X u, y) : (X, X^*, X)$$

to get

$$y : X \vdash (u, \lambda^X u, y) = (f(u, y), \lambda^X u, i) : (X, X^*, X)$$

and then we post-compose with

$$x : X, w : X^*, y : X \vdash (x \mid w \triangleleft y) : X$$

to get

$$y : X \vdash (u \mid \lambda^X u \triangleleft y) = (f(u, y) \mid \lambda^X u \triangleleft i) : X.$$

Note that although the type-theoretic justification is a bit complicated, at the level of terms the operation is intuitive: we simply simultaneously substitute u for $f(u, y)$ and y for i, wherever the latter appear as subterms. From now on we will perform such substitutions at term-level without further comment.

Now we define $x : X \vdash \phi(x) : X^*$ and $w : X^* \vdash \psi(w) : X$ by

$$\phi(x) \stackrel{\text{def}}{=} (\lambda^X v \mid \lambda^X u \triangleleft f(v, f(x, u)))$$
$$\psi(w) \stackrel{\text{def}}{=} (i \mid w \triangleleft i).$$

Finally, we can show that ϕ and ψ are inverse isomorphisms (so that X is isomorphic to its dual) with the following computations:

$$
\begin{aligned}
\psi(\phi(x)) &= (i \mid \lambda^X v \triangleleft i, \lambda^X u \triangleleft f(v, f(x, u))) \\
&= (i \mid \lambda^X u \triangleleft f(i, f(x, u))) && (\beta\text{-reduction for } v) \\
&= (i \mid \lambda^X u \triangleleft f(x, u)) && (i \text{ is a left unit for } f) \\
&= (u \mid \lambda^X u \triangleleft x) && ((x, u) = (f(x, u), i)) \\
&= x && (\beta\text{-reduction for } u) \\
\phi(\psi(w)) &= (\lambda^X v \mid \lambda^X u \triangleleft f(v, f(i, u)), w \triangleleft i) \\
&= (\lambda^X v \mid \lambda^X u \triangleleft f(v, u), w \triangleleft i) && (i \text{ is a left unit for } f) \\
&= (\lambda^X v \mid \lambda^X u \triangleleft v, w \triangleleft u) && ((v, u) = (f(v, u), i)) \\
&= (\lambda^X v \mid w \triangleleft v) && (\beta\text{-reduction for } u) \\
&= w && (\eta\text{-reduction for } v).
\end{aligned}
$$

We do not reproduce MTyson's string diagram proof here, but the reader is encouraged to compare it to our type-theoretic version. Note that this situation is partly "topological" in the sense of subsection 1.10 (the zigzag axioms for duality, and arguably the unit properties $f(x, i) = x$ and $f(i, x) = x$) and partly non-topological (the axiom $(x, y) = (f(x, y), i)$).

7.4. COMONOIDS AND SWEEDLER NOTATION. As in ordinary cartesian (or even linear) type theory, it is easy to use our type theory to define **monoid objects** in a symmetric monoidal category. The signature has one object M and two morphisms $m : (M, M) \to M$ and $e : () \to M$; we usually write $m(x, y)$ infix as $x \cdot y$ or just xy. The axioms are the expected

$$(xy)z = x(yz) \qquad\qquad xe = x \qquad\qquad ex = x.$$

However, with our type theory we can also define **comonoid objects**, which have instead two morphisms $\triangle : M \to (M, M)$ and $\varepsilon : M \to ()$, and axioms

$$(\triangle_{(1)}(\triangle_{(1)}(x)), \triangle_{(2)}(\triangle_{(1)}(x)), \triangle_{(2)}(x)) = (\triangle_{(1)}(x), \triangle_{(1)}(\triangle_{(2)}(x)), \triangle_{(2)}(\triangle_{(2)}(x)))$$

$$(\triangle_{(1)}(x) \mid \varepsilon(\triangle_{(2)}(x))) \qquad\qquad (\triangle_{(2)}(x) \mid \varepsilon(\triangle_{(1)}(x))).$$

As suggested in the introduction, this becomes much more manageable if we adopt the convention of traditional *Sweedler notation* for comonoids and comodules:

$$(x_{(1)}, x_{(2)}) \stackrel{\text{def}}{=} (\triangle_{(1)}(x), \triangle_{(2)}(x)).$$

Since there is no other meaning of $Z_{(i)}$ when Z is itself already a term, it is unambiguous to regard it as meaning $\triangle_{(i)}(Z)$ as long as no type has more than one relevant comultiplication. (We could formalize this with a more complicated type-checking algorithm, but we will be content to regard it as an informal abuse of notation.) We may regard this as a sort of "dual" to the shorthand notation "xy" for multiplication in a monoid, which omits the name or symbol for the product $(M, M) \to M$. With this notation, the axioms of a comonoid become

$$\left(x_{(1)(1)}, x_{(1)(2)}, x_{(2)}\right) = \left(x_{(1)}, x_{(2)(1)}, x_{(2)(2)}\right)$$

$$\left(x_{(1)} \mid \varepsilon(x_{(2)})\right) = x \qquad\qquad \left(x_{(2)} \mid \varepsilon(x_{(1)})\right) = x.$$

Traditional Sweedler notation also goes one step further: in view of the coassociativity axiom, it is unambiguous to write $\left(x_{(1)}, x_{(2)}, x_{(3)}\right)$ for either $\left(x_{(1)(1)}, x_{(1)(2)}, x_{(2)}\right)$ or $\left(x_{(1)}, x_{(2)(1)}, x_{(2)(2)}\right)$. In general, if subscripts are applied to a variable or a term that is already subscripted, with the maximum such subscript being n, then the subscript $_{(k)}$ means $\overbrace{_{(2)(2)} \cdots _{(2)}}^{k-1} {}_{(1)}$ if $k < n$ and $\overbrace{_{(2)(2)} \cdots _{(2)}}^{n-1}$ if $k = n$.

Intuitively, in cartesian type theory (i.e. in a cartesian monoidal category), everything can be duplicated and discarded with impunity; whereas a comonoid in a non-cartesian monoidal category is equipped with *specified ways* in which to duplicate and discard elements. (Indeed, a cartesian monoidal category is precisely a symmetric monoidal category in which every object is equipped with a commutative comonoid structure in a natural way.) We thus view $x_{(1)}$ and $x_{(2)}$ as "duplicated copies" of x, the subscripts tracking the order of duplication. Similarly, we can regard $\varepsilon(x)$ as "discarding" the element x, which inspires us to introduce a sort of "nullary Sweedler notation"

$$\cancel{x} \stackrel{\text{def}}{=} \varepsilon(x).$$

Thus, for instance, the counit axioms of a comonoid become $\left(x_{(1)} \mid \cancel{x_{(2)}}\right) = x$ and $\left(x_{(2)} \mid \cancel{x_{(1)}}\right) = x$. Note that by coassociativity, we also have equalities such as $\left(x_{(1)}, x_{(2)} \mid \cancel{x_{(3)}}\right) = \left(x_{(1)}, x_{(2)}\right)$ and $\left(x_{(1)}, x_{(3)} \mid \cancel{x_{(2)}}\right) = \left(x_{(1)}, x_{(2)}\right)$ and so on.

Such shorthands need not be restricted to comonoids either. For instance, traditional Sweedler notation is also used for *comodules*, which have a coaction $D \to (C, D)$ by a coalgebra C. As an example with even greater generality, suppose M is dualizable and has a coaction $\triangle : M \to (A, M)$, satisfying no axioms at all, and that $f : M \to M$ is an endomorphism that respects $\triangle$ in that $\left(f(x_{(1)}), f(x_{(2)})\right) = \left(f(x)_{(1)}, f(x)_{(2)}\right)$. Then we can use this notation to verify the *fixed-point property* of traces from [PS14, Corollary 5.3]:

$$\left(f(f(u)_{(1)}) \mid \lambda^M u \triangleleft f(u)_{(2)}\right) = \left(f(v_{(1)}) \mid \lambda^M u \triangleleft v_{(2)}, \lambda^M v \triangleleft f(u)\right)$$
$$= \left(f(v_{(1)}) \mid \lambda^M v \triangleleft f(v_{(2)})\right)$$
$$= \left(f(v)_{(1)} \mid \lambda^M v \triangleleft f(v)_{(2)}\right) : A.$$

7.5. FROBENIUS MONOIDS. A **Frobenius monoid** is an object that is both a monoid and a comonoid and satisfies the additional axiom

$$x : M, y : M \vdash \left(x_{(1)}, x_{(2)} y\right) = \left(x y_{(1)}, y_{(2)}\right). \tag{7.6}$$

Usually this is stated as two axioms saying that both sides of the above equation equal $((xy)_{(1)}, (xy)_{(2)})$. But this follows from the above axiom by the following argument, which I learned from [PS09]:

$$
\begin{aligned}
(x_{(1)}, x_{(2)}y) &= (x_{(1)}, x_{(2)}y_{(1)} \mid y_{(2)}) && \text{(by counitality)} \\
&= (x_{(1)}, x_{(2)(1)} \mid x_{(2)(2)}y) && \text{(by (7.6))} \\
&= (x_{(1)(1)}, x_{(1)(2)} \mid x_{(2)}y) && \text{(by coassociativity)} \\
&= ((xy_{(1)})_{(1)}, (xy_{(1)})_{(2)} \mid y_{(2)}) && \text{(by (7.6))} \\
&= ((xy)_{(1)}, (xy)_{(2)}) && \text{(by counitality)}.
\end{aligned}
$$

The Frobenius axiom(s) are "topological", so their string diagrams get a good deal of leverage from topological intuition. Thus, Frobenius monoids are not a very good example for the relative usefulness of type theory. However, for purposes of comparison, we include a proof of one of the basic facts about Frobenius monoids; namely that they are self-dual, with unit and counit:

$$
\begin{aligned}
(w \triangleleft x) &\overset{\text{def}}{=} (\mid wx) \\
(u, \lambda^M u) &\overset{\text{def}}{=} (e_{(1)}, e_{(2)}).
\end{aligned}
$$

The axioms of a dual pair follow quite easily:

$$
(u \mid \lambda^M u \triangleleft x) \overset{\text{def}}{=} (e_{(1)} \mid e_{(2)}x) = (ex_{(1)} \mid x_{(2)}) = ex = x
$$

$$
(\lambda^M u \mid w \triangleleft u) \overset{\text{def}}{=} (e_{(2)} \mid we_{(1)}) = (w_{(2)}e \mid w_{(1)}) = we = w.
$$

7.7. REMARK. A **hypergraph category** [Kis14] is a symmetric monoidal category in which every object is equipped with a Frobenius monoid structure that is commutative $(xy = yx)$, cocommutative $((x_{(1)}, x_{(2)}) = (x_{(2)}, x_{(1)}))$, and special a.k.a. separable $(x_{(1)} x_{(2)} = x)$, and such that the Frobenius monoid structure on any tensor product $X \otimes Y$ is induced from those on X and Y in the standard way. The definition of **hypergraph prop** is a bit simpler: it is just a prop in which every object is equipped with a commutative, cocommutative, special Frobenius monoid structure. Since tensor products in a prop are only formal, the final condition is essentially automatic. More specifically, the final condition on a hypergraph category $\mathcal{C}$ is not needed to show that it has an underlying hypergraph prop $U\mathcal{C}$, but it is precisely what is needed to show that $\mathcal{C}$ is equivalent, as a hypergraph category, to the free symmetric monoidal category generated by $U\mathcal{C}$. (One might even argue that for this reason, hypergraph props are a more natural structure than hypergraph categories.)

Now given any signature $\mathcal{G}$, we can augment it by adding a commutative, cocommutative, special Frobenius monoid structure on every object. This yields a new signature $\mathcal{G}^{\text{hy}}$ such that $\mathfrak{F}\mathcal{G}^{\text{hy}}$ is the free hypergraph prop generated by $\mathcal{G}$. Any signature $\mathcal{R}$ of relations for $\mathcal{G}$ carries over to $\mathcal{G}^{\text{hy}}$ as well, so we can construct presented hypergraph props as well.

7.8. HOPF MONOIDS AND ANTIPODES. A monoid object in a cartesian monoidal category is also (like every object) a comonoid, but the monoid and comonoid structures do not satisfy the Frobenius axiom. Instead they satisfy the **bimonoid axioms**:

$$x : M, y : M \vdash (x_{(1)}y_{(1)}, x_{(2)}y_{(2)}) = ((xy)_{(1)}, (xy)_{(2)}) : (M, M)$$
$$\vdash (e_{(1)}, e_{(2)}) = (e, e) : (M, M)$$
$$x : M, y : M \vdash (\mid \cancel{xy}) = (\mid \cancel{x}, \cancel{y}) : ()$$
$$\vdash (\mid \cancel{e}) = () : ().$$

Thus, a bimonoid object in an arbitrary monoidal category can be regarded as a "non-cartesian" version of a monoid object in a cartesian monoidal category. Indeed, if a bimonoid is cocommutative $(x_{(1)}, x_{(2)}) = (x_{(2)}, x_{(1)})$ then it is a monoid object in the cartesian monoidal category of cocommutative comonoids, and dually if it is commutative $(xy = yx)$ then it is a monoid object in the opposite of the cocartesian monoidal category of commutative monoids. But in the non-commutative, non-cocommutative case we obtain something truly new.

The analogue for a bimonoid of the inversion operation, making a monoid into a group, is called an **antipode**: an operation $x : M \vdash \overline{x} : M$ such that

$$x : M \vdash x_{(1)}\, \overline{x_{(2)}} = (e \mid \cancel{x}) : M$$
$$x : M \vdash \overline{x_{(1)}}\, x_{(2)} = (e \mid \cancel{x}) : M.$$

A bimonoid equipped with an antipode (a non-cartesian analogue of a group object) is called a **Hopf monoid**. Note that the comonoid structure is necessary in order to even formulate the antipode axioms: we need to duplicate x in order to invert one copy of it, and on the other side of the equation we need to discard x in order to write simply "e". In a cartesian monoidal category, Hopf monoids are precisely group objects in the usual sense. However, note also that bimonoids and Hopf monoids in a symmetric monoidal category are self-dual: such a structure on $M \in \mathcal{C}$ is equivalent to such a structure on $M \in \mathcal{C}^{\mathrm{op}}$.

As mentioned in the introduction, cartesian type theory can internalize the basic fact of group theory that inverses in any monoid are unique: if $\overline{x}$ and $\widehat{x}$ are both inverses of x the

$$\overline{x} = \overline{x}e = \overline{x}(x\widehat{x}) = (\overline{x}x)\widehat{x} = e\widehat{x} = \widehat{x}.$$

Therefore, a monoid object in any cartesian monoidal category admits at most one inverse, and hence both *cocommutative* Hopf monoids and *commutative* ones have unique antipodes. Cartesian type theory has nothing to say about Hopf monoids that are neither commutative nor cocommutative, but in our type theory we can reproduce essentially the same argument: if $x : M \vdash \overline{x} : M$ and $x : M \vdash \widehat{x} : M$ are both antipodes, we compute

$$\overline{x} = \overline{x}\, e = (\overline{x_{(1)}}\, e \mid \cancel{x_{(2)}}) = \overline{x_{(1)}}\, x_{(21)}\, \widehat{x_{(22)}} = (e\, \widehat{x_{(2)}} \mid \cancel{x_{(1)}}) = e\, \widehat{x} = \widehat{x}.$$

Thus, even in a non-cartesian situation we can use a very similar set-like argument, as long as we keep track of where elements get "duplicated and discarded". I encourage the

reader to write out a proof of this fact using traditional arrow notation or string diagrams for comparison.

As another example, if H and K are bimonoids, a **bimonoid homomorphism** is a morphism $x : H \vdash f(x) : K$ such that

$$f(xy) = f(x)\,f(y) \qquad\qquad (f(x)_{(1)}, f(x)_{(2)}) = (f(x_{(1)}), f(x_{(2)}))$$

$$f(e) = e \qquad\qquad (\mid f(x)) = (\mid x).$$

Now we can show that when H and K are Hopf monoids, any such bimonoid homomorphism preserves antipodes.

$$
\begin{aligned}
f(\overline{x}) &= (f(\overline{x_{(1)}}) \mid x_{(2)}) \\
&= (f(\overline{x_{(1)}}) \mid f(x_{(2)})) \\
&= (f(\overline{x_{(1)}})\,e \mid f(x_{(2)})) \\
&= f(\overline{x_{(1)}})\,f(x_{(2)})_{(1)}\,\overline{f(x_{(2)})_{(2)}} \\
&= f(\overline{x_{(1)}})\,f(x_{(2)(1)})\,\overline{f(x_{(2)(2)})} \\
&= f(\overline{x_{(1)(1)}})\,f(x_{(1)(2)})\,\overline{f(x_{(2)})} \\
&= f(\overline{x_{(1)(1)}}\,x_{(1)(2)})\,\overline{f(x_{(2)})} \\
&= (f(e)\,\overline{f(x_{(2)})} \mid x_{(1)}) \\
&= (e\,\overline{f(x_{(2)})} \mid x_{(1)}) \\
&= (\overline{f(x_{(2)})} \mid x_{(1)}) \\
&= \overline{f(x)}.
\end{aligned}
$$

There is a more general approach to results of this sort, which we will return to in the next section.

7.9. WEAK BIMONOIDS. A **weak bimonoid** [PS09][6] is a monoid and comonoid that satisfies, instead of the bimonoid axioms, the following weakened ones:

$$
\begin{aligned}
((xy)_{(1)}, (xy)_{(2)}) &= (x_{(1)}y_{(1)}, x_{(2)}y_{(2)}) \\
(\mid xyz) &= (\mid xy_{(1)}, y_{(2)}z) \\
&= (\mid xy_{(2)}, y_{(1)}z) \\
(e_{(1)}, e_{(2)}, e_{(3)}) &= (e_{(1)}, e_{(2)}e'_{(1)}, e'_{(2)}) \\
&= (e_{(1)}, e'_{(1)}e_{(2)}, e'_{(2)}).
\end{aligned}
$$

---

[6]In [PS09] these definitions are given in the additional generality of a *braided* monoidal category, but our type theory only applies to symmetric monoidal categories. This is sufficient to include a number of examples, however, such as the category algebra of a category with finitely many objects.

For a weak bimonoid we define

$$s(x) \stackrel{\text{def}}{=} (e_{(1)} \mid \cancel{e_{(2)}x})$$
$$t(x) \stackrel{\text{def}}{=} (e_{(1)} \mid \cancel{xe_{(2)}})$$
$$r(x) \stackrel{\text{def}}{=} (e_{(2)} \mid \cancel{e_{(1)}x}).$$

Many equations relating the weak bimonoid structure and the operations s, t, r are proven in [PS09] using string diagrams. All of them can also be proven in our type theory. For instance, here is a version of [PS09, Appendix B, eq. (1)]:

$$
\begin{aligned}
(s(x)_{(1)}, s(x)_{(2)}) &= (e_{(1)(1)}, e_{(1)(2)} \mid \cancel{e_{(2)}x}) \\
&= (e_{(1)}, e_{(2)} \mid \cancel{e_{(3)}x}) \\
&= (e_{(1)}, e_{(2)}e'_{(1)} \mid \cancel{e'_{(2)}x}) \\
&= (e_{(1)}, e_{(2)}s(x)).
\end{aligned}
$$

Here is a version of [PS09, Appendix B, eq. (4)]:

$$
\begin{aligned}
((x\,s(y))_{(1)}, (x\,s(y))_{(2)}) &= (x_{(1)}s(y)_{(1)}, x_{(2)}s(y)_{(2)}) \\
&= (x_{(1)}e_{(1)}, x_{(2)}e_{(2)}s(y)) \qquad \text{(using (1))} \\
&= ((xe)_{(1)}, (xe)_{(2)}s(y)) \\
&= (x_{(1)}, x_{(2)}s(y)).
\end{aligned}
$$

And here is a version of [PS09, Appendix B, eq. (13)].

$$
\begin{aligned}
t(x)\,r(y) &= (e_{(1)}e'_{(2)} \mid \cancel{xe_{(2)}}, \cancel{e'_{(1)}y}) \\
&= (e_{(2)} \mid \cancel{xe_{(3)}}, \cancel{e_{(1)}y}) \\
&= (e_{(2)}e'_{(1)} \mid \cancel{xe'_{(2)}}, \cancel{e_{(1)}y}) \\
&= r(y)\,t(x).
\end{aligned}
$$

As an example of a somewhat longer proof, here is a version of [PS09, Appendix B, eq. (11)]. Unlike the previous examples, this is not a verbatim translation of their proof; theirs builds on previous lemmas, whereas the below proof is direct.

$$
\begin{aligned}
(t(x_{(1)}), x_{(2)}y) &= (e_{(1)}, x_{(2)}y \mid \cancel{x_{(1)}e_{(2)}}) \\
&= (e_{(1)}, (xe')_{(2)}y \mid \cancel{(xe')_{(1)}e_{(2)}}) \\
&= (e_{(1)}, x_{(2)}e'_{(2)}y \mid \cancel{x_{(1)}e'_{(1)}e_{(2)}}) \\
&= (e_{(1)}, x_{(2)}e_{(3)}y \mid \cancel{x_{(1)}e_{(2)}}) \\
&= (e_{(1)}, x_{(2)}e_{(2)(2)}y \mid \cancel{x_{(1)}e_{(2)(1)}}) \\
&= (e_{(1)}, (xe_{(2)})_{(2)}y \mid \cancel{(xe_{(2)})_{(1)}}) \\
&= (e_{(1)}, xe_{(2)}y)
\end{aligned}
$$

$$= \left(e_{(1)}, x(e_{(2)}y)_{(2)} \mid \overline{(e_{(2)}y)_{(1)}}\right)$$
$$= \left(e_{(1)}, xe_{(2)(2)}y_{(2)} \mid \overline{e_{(2)(1)}y_{(1)}}\right)$$
$$= \left(e_{(1)}, xe_{(3)}y_{(2)} \mid \overline{e_{(2)}y_{(1)}}\right)$$
$$= \left(e_{(1)}, xe'_{(2)}y_{(2)} \mid \overline{e_{(2)}e'_{(1)}y_{(1)}}\right)$$
$$= \left(e_{(1)}, x(e'y)_{(2)} \mid \overline{e_{(2)}(e'y)_{(1)}}\right)$$
$$= \left(e_{(1)}, xy_{(2)} \mid \overline{e_{(2)}y_{(1)}}\right)$$
$$= \left(s(y_{(1)}), xy_{(2)}\right).$$

If H and K are weak bimonoids, a **weak bimonoid homomorphism** $f : H \to K$ must commute with both monoid and comonoid structures as in subsection 7.8. It follows that it also commutes with s, t, r; this is shown in [PS09, Lemma 1.2], and rendered below for s in type theory:

$$f(s(x)) = \left(f(e_{(1)}) \mid \overline{e_{(2)}x}\right)$$
$$= \left(f(e_{(1)}) \mid f(\overline{e_{(2)}x})\right)$$
$$= \left(f(e_{(1)}) \mid \overline{f(e_{(2)})f(x)}\right)$$
$$= \left(f(e)_{(1)} \mid \overline{f(e)_{(2)}f(x)}\right)$$
$$= \left(e_{(1)} \mid \overline{e_{(2)}f(x)}\right)$$
$$= s(f(x)).$$

A weak bimonoid is a **weak Hopf monoid** if it has an **antipode**: a morphism $x : H \vdash \overline{x} : H$ satisfying

$$\overline{x_{(1)}}\, x_{(2)} = t(x)$$
$$x_{(1)}\, \overline{x_{(2)}} = r(x)$$
$$\overline{x_{(1)}}\, x_{(2)}\, \overline{x_{(3)}} = \overline{x}.$$

This implies immediately that

$$\overline{x} = \overline{x_{(1)}}\, x_{(2)}\, \overline{x_{(3)}} = \overline{x_{(1)}}\, r(x_{(2)}) \qquad\qquad \overline{x} = \overline{x_{(1)}}\, x_{(2)}\, \overline{x_{(3)}} = t(x_{(1)})\, \overline{x_{(2)}}.$$

As for ordinary bimonoids in subsection 7.8, antipodes for weak bimonoids are unique: if $x : H \vdash \overline{x} : H$ and $x : H \vdash \widehat{x} : H$ are both antipodes, we have:

$$\overline{x} = \overline{x_{(1)}}\, r(x_{(2)})$$
$$= \overline{x_{(1)}}\, x_{(2)(1)}\, \widehat{x_{(2)(2)}}$$
$$= \overline{x_{(1)(1)}}\, x_{(1)(2)}\, \widehat{x_{(2)}}$$
$$= t(x_{(1)})\, \widehat{x_{(2)}}$$
$$= \widehat{x}.$$

Similarly, we can translate the argument of [PS09, Proposition 2.2] that homomorphisms $f : H \to K$ of weak bimonoids preserve antipodes:

$$
\begin{aligned}
f(\overline{x}) &= f(\overline{x_{(1)}}\, r(x_{(2)})) \\
&= f(\overline{x_{(1)}})\, f(r(x_{(2)})) \\
&= f(\overline{x_{(1)}})\, r(f(x_{(2)})) \\
&= f(\overline{x_{(1)}})\, f(x_{(2)})_{(1)}\, \overline{f(x_{(2)})_{(2)}} \\
&= f(\overline{x_{(1)}})\, f(x_{(2)(2)})\, \overline{f(x_{(2)(2)})} \\
&= f(\overline{x_{(1)(1)}})\, f(x_{(1)(2)})\, \overline{f(x_{(2)})} \\
&= f(\overline{x_{(1)(1)}}\, x_{(1)(2)})\, \overline{f(x_{(2)})} \\
&= f(t(x_{(1)}))\, \overline{f(x_{(2)})} \\
&= t(f(x_{(1)}))\, \overline{f(x_{(2)})} \\
&= t(f(x)_{(1)})\, \overline{f(x)_{(2)}} \\
&= \overline{f(x)}.
\end{aligned}
$$

We end by sketching another approach to these sorts of uniqueness results that is inspired by the "types are like sets" perspective of our type theory. For monoids in the category of sets, the uniqueness of inverses is a *pointwise* property: for any element x, if y and z are two elements that are both inverses of x, then $y = z$. However, the uniqueness of antipodes as we have proven it above, for both bimonoids and weak bimonoids, is instead a statement about *inversion operators* that apply to all elements at once.

The pointwise statement is stronger, and this makes for simpler proofs. For instance, we can simply show that a monoid homomorphism $f : H \to K$ preserves inverses *of elements* in the sense that if y is an inverse of x then $f(y)$ is an inverse of $f(x)$, and conclude directly from pointwise uniqueness of inverses in K that f of "the" inverse of x coincides with "the" inverse of $f(x)$.

We can also formulate a "pointwise" sort of uniqueness of inverses in the general case. Suppose H is a weak Hopf monoid, with given antipode $x : H \vdash \overline{x} : H$. Suppose also we have a comonoid X and a comonoid morphism $x : X \vdash g(x) : H$, and also a morphism $x : X \vdash i(x) : H$ such that

$$
\begin{aligned}
i(x_{(1)})\, g(x_{(2)}) &= t(g(x)) \\
g(x_{(1)})\, i(x_{(2)}) &= r(g(x)) \\
i(x_{(1)})\, g(x_{(2)})\, i(x_{(3)}) &= i(x).
\end{aligned}
\tag{7.10}
$$

We think of g as an X-indexed family of elements of H, and i as a similarly indexed family of "inverses". Then we can calculate

$$
t(g(x_{(1)}))\, i(x_{(2)}) = i(x_{(1)})\, g(x_{(2)})\, i(x_{(3)}) = i(x)
$$

$$
i(x_{(1)})\, r(g(x_{(2)})) = i(x_{(1)})\, g(x_{(2)})\, i(x_{(3)}) = i(x)
$$

and hence

$$
\begin{aligned}
i(x) &= i(x_{(1)}) \, r(g(x_{(2)})) \\
&= i(x_{(1)}) \, g(x_{(2)}) \, \overline{g(x_{(3)})} \\
&= t(g(x_{(1)})) \, \overline{g(x_{(2)})} \\
&= t(g(x)_{(1)}) \, \overline{g(x)_{(2)}} \\
&= \overline{g(x)}.
\end{aligned}
$$

Note that in fact it suffices for X to be a co-semigroup and g to be a co-semigroup morphism. Moreover, instead of a single co-semigroup, X could be a context of co-semigroups.

Applying this with $X \overset{\mathrm{def}}{=} H$ and $g(x) \overset{\mathrm{def}}{=} x$, we obtain uniqueness of the antipode itself. But we can also derive preservation of the antipode by a weak bimonoid homomorphism $f : H \to K$, by taking $X \overset{\mathrm{def}}{=} H$ and $g \overset{\mathrm{def}}{=} f$ with $i(x) \overset{\mathrm{def}}{=} f(\overline{x})$. We simply check that this satisfies the three properties (7.10):

$$
\begin{aligned}
f(\overline{x_{(1)}}) \, f(x_{(2)}) &= f(\overline{x_{(1)}} \, x_{(2)}) \\
&= f(t(x)) \\
&= t(f(x)). \\
f(x_{(1)}) \, f(\overline{x_{(2)}}) &= f(x_{(1)} \, \overline{x_{(2)}}) \\
&= f(r(x)) \\
&= r(f(x)). \\
f(\overline{x_{(1)}}) \, f(x_{(2)}) \, f(\overline{x_{(3)}}) &= f(\overline{x_{(1)}} \, x_{(2)} \, \overline{x_{(3)}}) \\
&= f(\overline{x}).
\end{aligned}
$$

Then the above argument shows immediately that $f(\overline{x}) = \overline{f(x)}$, as desired.

As a final example, we show that for a weak Hopf monoid H, the antipode is a weak monoid anti-homomorphism. (Since Hopf monoids are self-dual, this implies that the antipode is also a comonoid anti-homomorphism, which was proven by a long string diagram calculation in [PS09, Proposition 2.3]). Consider how we prove the analogous statement in the category of sets, that inversion in a group is a monoid anti-homomorphism. Arguably the most natural way is to simply check that $y^{-1}x^{-1}$ is an inverse of xy:

$$
\begin{aligned}
(y^{-1}x^{-1})(xy) &= y^{-1}(x^{-1}x)y = y^{-1}ey = y^{-1}y = e \\
(xy)(y^{-1}x^{-1}) &= x(yy^{-1})x^{-1} = xex^{-1} = xx^{-1} = e
\end{aligned}
$$

and conclude by uniqueness of inverses that $y^{-1}x^{-1} = (xy)^{-1}$.

Using our notion of pointwise uniqueness, we can reproduce this inside our type theory to apply to weak Hopf monoids. We take $X \overset{\mathrm{def}}{=} (H, H)$, with its induced comonoid structure $x : H, y : H \vdash (x_{(1)}, y_{(1)}, x_{(2)}, y_{(2)}) : (H, H, H, H)$. We define $g(x, y) \overset{\mathrm{def}}{=} xy$, which is a co-semigroup morphism (though not a comonoid morphism) by the first axiom of a weak

bimonoid, and $i(x, y) \stackrel{\text{def}}{=} \overline{y}\,\overline{x}$, and check the three properties (7.10):

$$
\begin{aligned}
i((x, y)_{(1)})\, g((x, y)_{(2)}) &= \overline{y_{(1)}}\, \overline{x_{(1)}}\, x_{(2)}\, y_{(2)} \\
&= \overline{y_{(1)}}\, t(x)\, y_{(2)} \\
&= \overline{(t(x)\, y)_{(1)}}\, (t(x)\, y)_{(2)} \quad (4) \\
&= t(t(x)\, y) \\
&= t(xy) \quad\quad\quad\quad (3) \\
&= t(g(x, y)).
\end{aligned}
$$

Here the line labeled (4) uses the so-numbered equation $(y_{(1)}, t(x)\, y_{(2)}) = ((t(x)\, y)_{(1)}, (t(x)\, y)_{(2)})$ from [PS09], and similarly the line (3) is an instance of their equation (3). The next calculation is dual.

$$
\begin{aligned}
g((x, y)_{(1)})i((x, y)_{(2)}) &= x_{(1)}\, y_{(1)}\, \overline{y_{(1)}}\, \overline{x_{(2)}} \\
&= x_{(1)}\, r(y)\, \overline{x_{(2)}} \\
&= (x\, r(y))_{(1)}\, \overline{(x\, r(y))_{(2)}} \\
&= r(x\, r(y)) \\
&= r(xy) \\
&= r(g(x, y)).
\end{aligned}
$$

The final calculation uses the commutativity of t with r, equation (13) from [PS09] which we also proved above.

$$
\begin{aligned}
i((x, y)_{(1)})\, g((x, y)_{(2)})\, i((x, y)_{(3)}) &= \overline{y_{(1)}}\, \overline{x_{(1)}}\, x_{(2)}\, y_{(2)}\, \overline{y_{(3)}}\, \overline{x_{(3)}} \\
&= \overline{y_{(1)}}\, t(x_{(1)})\, r(y_{(2)})\, \overline{x_{(2)}} \\
&= \overline{y_{(1)}}\, r(y_{(2)})\, t(x_{(1)})\, \overline{x_{(2)}} \quad (13) \\
&= \overline{y}\,\overline{x} \\
&= i(x, y).
\end{aligned}
$$

We leave further applications to the interested reader.

References

[Bar79] Michael Barr. *∗-autonomous categories*, volume 752 of *Lecture Notes in Mathematics*. Springer, 1979.

[Bar91] Michael Barr. *-autonomous categories and linear logic. *Mathematical Structures in Computer Science*, 1(2):159–178, 1991.

[BCR18] John C. Baez, Brandon Coya, and Franciscus Rebro. Props in network theory. *Theory and Applications of Categories*, 33(25):727–783, 2018. arXiv:1707.08321.

[BCST96] R. Blute, R. Cockett, R. Seely, and T. Trimble. Natural deduction and coherence for weakly distributive categories. *J. Pure and Appl. Algebra*, 113:229–296, 1996.

[Cam17] Tim Campion. If a ⊗-idempotent object has a dual, must it be self-dual? MathOverflow, 2017. https://mathoverflow.net/q/277991 (version: 2017-08-09).

[CPT16] Luís Caires, Frank Pfenning, and Bernardo Toninho. Linear logic propositions as session types. *Mathematical Structures in Computer Science*, 26(3):367–423, 2016.

[CS97] Robin Cockett and Robert Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997. Corrected version available at https://www.math.mcgill.ca/rags/linear/wdc-fix.pdf.

[Dun06] Ross Duncan. *Types for quantum computing*. PhD thesis, Oxford University Computing Laboratory, 2006.

[EL89] P. Eades and Xuemin Lin. How to draw a directed graph. *[Proceedings] 1989 IEEE Workshop on Visual Languages*, pages 13–17, 1989.

[Has97] Masahito Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. PhD thesis, University of Edinburgh, 1997. http://www.kurims.kyoto-u.ac.jp/~hassei/papers/ECS-LFCS-97-360.pdf.

[Joh02] Peter T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volumes 1 and 2*. Number 43 in Oxford Logic Guides. Oxford Science Publications, 2002.

[JS91] André Joyal and Ross Street. The geometry of tensor calculus. I. *Adv. Math.*, 88(1):55–112, 1991.

[JSV96] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Math. Proc. Cambridge Philos. Soc.*, 119(3):447–468, 1996.

[Kis14] Aleks Kissinger. Finite matrices are complete for (dagger-)hypergraph categories. arXiv:1406.5942, 2014.

[KL80] G. M. Kelly and M. L. Laplaza. Coherence for compact closed categories. *J. Pure Appl. Algebra*, 19:193–213, 1980.

[Mac65] Saunders MacLane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71:40–106, 1965.

[ML98] Saunders Mac Lane. *Categories For the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, second edition, 1998.

[Ong96] C.-H. L. Ong. A semantic view of classical proofs – type-theoretic, categorical, and denotational characterizations (extended abstract). In *Proceedings of LICS '96*, pages 230–241. IEEE Press, 1996.

[PLC17] Jovana Obradović Pierre-Louis Curien. A formal language for cyclic operads. *Higher Structures*, 1(1), 2017. arXiv:1602.07502.

[PR84] Roger Penrose and Wolfgang Rindler. *Spinors and space-time. Vol. 1*. Cambridge Monographs on Mathematical Physics. Cambridge University Press, Cambridge, 1984. Two-spinor calculus and relativistic fields.

[PS09] Craig Pastro and Ross Street. Weak Hopf monoids in braided monoidal categories. *Algebra Number Theory*, 3(2):149–207, 2009.

[PS14] Kate Ponto and Michael Shulman. Traces in symmetric monoidal categories. *Expositiones Mathematicae*, 32(2):248–273, 2014. arXiv:1107.6032.

[Red93] Uday S. Reddy. A typed foundation for directional logic programming. In E. Lamma and P. Mello, editors, *Extensions of Logic Programming*, pages 282–318, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[Sel11] Peter Selinger. A survey of graphical languages for monoidal categories. In Bob Coeke, editor, *New Structures for Physics*, chapter 4. Springer, 2011. arXiv:0908.3347.

[Shi99] Masaru Shirahata. A sequent calculus for compact closed categories, 1999. `http://www.fbc.keio.ac.jp/~sirahata/Research/cmll.ps`.

[Sza75] M.E. Szabo. Polycategories. *Communications in Algebra*, 3(8):663–689, 1975.

Department of Mathematics
University of San Diego
5998 Alcala Park
San Diego, CA, 92110
USA
Email: `shulman@sandiego.edu`

THEORY AND APPLICATIONS OF CATEGORIES will disseminate articles that significantly advance the study of categorical algebra or methods, or that make significant new contributions to mathematical science using categorical methods. The scope of the journal includes: all areas of pure category theory, including higher dimensional categories; applications of category theory to algebra, geometry and topology and other areas of mathematics; applications of category theory to computer science, physics and other mathematical sciences; contributions to scientific knowledge that make use of categorical methods.

Articles appearing in the journal have been carefully and critically refereed under the responsibility of members of the Editorial Board. Only papers judged to be both significant and excellent are accepted for publication.