String Diagrams for Non-Strict Monoidal Categories

Paul Wilson
University of Southampton, UK

Dan Ghica
University of Birmingham, UK

Fabio Zanasi
University College London, UK

Abstract

Whereas string diagrams for strict monoidal categories are well understood, and have found application in several fields of Computer Science, graphical formalisms for non-strict monoidal categories are far less studied. In this paper, we provide a presentation by generators and relations of string diagrams for non-strict monoidal categories, and show how this construction can handle applications in domains such as digital circuits and programming languages. We prove the correctness of our construction, which yields a novel proof of Mac Lane's strictness theorem. This in turn leads to an elementary graphical proof of Mac Lane's *coherence* theorem, and in particular allows for the inductive construction of the canonical isomorphisms in a monoidal category.

2012 ACM Subject Classification Theory of computation

Keywords and phrases String Diagrams, Strictness, Coherence

Digital Object Identifier 10.4230/LIPIcs.CSL.2023.37

Related Version *Extended Version*: https://arxiv.org/abs/2201.11738

1 Introduction

String diagrams are a rigorous graphical notation for category theory that is proving useful in a broad variety of application domains, such as quantum systems [9], computational linguistics [8], digital circuits [12], or signal flow analysis [5]. What the majority of string diagrammatic notations have in common is that they are devised for monoidal categories in which the tensor is *strict*, i.e. the associator and unitor morphisms are identities. As Joyal and Street explain in their seminal *Geometry of Tensor Calculus* [16], the choice of using a strict monoidal category was motivated by convenience ("simplicity of exposition") and by a wish to focus on "aspects other than the associativity of tensor product". Furthermore, they believed that "most results obtained with the hypothesis that a tensor category is strict can be reformulated and proved without this condition."

Indeed, in terms of mathematical power, this statement is true. However, string diagrams have been used increasingly as a convenient *syntax* for languages with models in (strict) monoidal categories. And, when used as syntax, the distinction between strict and non-strict tensor becomes relevant, if not in terms of mathematical expressiveness then at least as a mechanism of abstraction. This is why modern programming languages, and even some modern hardware design languages such SystemVerilog [22], use non-strict features such as *tuples* and *structs* which can nest in non-trivial ways. These non-strict structures could be manually "strictified" by the programmer by flattening them into arrays. Using such programmer conventions instead of native syntactic support does not entail a loss of expressiveness, but a loss of code readability, convenience, and general programmer effectiveness.

In this paper, we address the problem of expanding the graphical language of string diagrams with the required features that allow the expression of non-strict tensors. What makes the language of strict tensors convenient for the graphical representation is that objects are naturally represented as *lists of wires*. This suggests that string diagrams make use of strictness in an essential way and, indeed, naive attempts to define string diagram languages for non-strict monoidal categories can render the notation so heavy-going as to lose the intuitiveness that makes it so attractive in the first place. A more sophisticated solution, which we propose here, is to deliberately use the strictification of a possibly non-strict monoidal category in order to make string diagrams function in this setting with a minimum of additional overhead. These points will be illustrated with examples in Section 2.

Concretely, the basic idea is to use new operations to "pack" pairs of wires into single wires with internal tensorial structure and to "unpack" structured wires into pairs of wires labelled with the tensor component objects. The repeated application of unpacking can flatten any wire with an arbitrarily complex tensor structure into a list of wires labelled with elementary objects. Other new operations are used to "hide" or "reveal" wires labelled with the tensor unit. These four families of new operations are used to define the associators and the unitors of the strictified category.

1.1 Contributions

We propose a strictification construction yielding a graphical language for non-strict monoidal categories. With respect to traditional string diagrams, it provides a more fine-grained representation of tensoring, whose usefulness we demonstrate in motivating examples drawn from circuit theory and programming language semantics. The bulk of the paper is then dedicated to showing that the construction is correct, i.e. the strictified category in which string diagrams live is monoidally equivalent to the original non-strict category. Our proof of monoidal equivalence is new: in contrast to Mac Lane's we do not rely on the coherence theorem, and instead construct the functors of the equivalence explicitly. Consequently, we are able to give a new elementary proof of the coherence theorem: we show *graphically* that the free monoidal category on a single generator forms a preorder. The remainder of the coherence result is largely a reformulation Mac Lane's original corollary, but in a way that we believe has pedagogical value. For brevity, we reserve discussion of the corollary for an extended version of the paper [23].

1.2 Related Work

The use of adapter morphisms which can "pack" and "unpack" wires has been explored in various forms. Our adapters can be recovered as instances of more sophisticated constructs used in the study of coherence of weakly distributive categories [3]. These categories use two distinct tensors and an additional kind of wire ("thinning links") in their string diagram language. If this additional structure were to be somehow omitted the remaining constructs would be rather similar to ours. However, this is not explored in [3], and it is not obvious how this streamlined setting would lead to coherence results for "mere" monoidal categories as a corresponding simplification of the coherence of weakly distributive categories. A non-diagrammatic approach giving a type theory for symmetric monoidal categories can also be found in [21]. The idea also appears in the study of quantum [6] and reversible [7] circuits, although these do not study the connection to the strictness and coherence theorems. The distinctive feature of our paper is to show explicitly how introducing adapters allows non-strict categories to take advantage of graph based datastructures for *strict* monoidal

categories such as those of [24]. More concretely, by *explicitly* defining the functors mapping between a non-strict category and its "strictified" counterpart, we obtain datastructure-independent algorithms for translating between strict and non-strict settings. In addition, we formulate our approach using presentations by generators and relations, which brings the strictness question much closer to current graphical calculi approaches to circuits such as [17, 25, 13, 4, 2]. Finally, our approach allows us to give a self-contained proof of Mac Lane's strictness theorem that does not rely on the coherence theorem, allowing us to avoid its associated pitfalls.

1.3 Synopsis

In Section 2 we present our graphical calculus for (non-strict) monoidal categories, in the form of a strictification procedure. Subsections 2.1, 2.2, and 2.3 illustrate a series of motivating examples. Section 3 justifies our construction by proving that it yields an equivalence of categories. Section 4 revisits MacLane's Coherence theorem and some of its consequences in light of the approach we presented. Section 5 is dedicated to conclusions and future work.

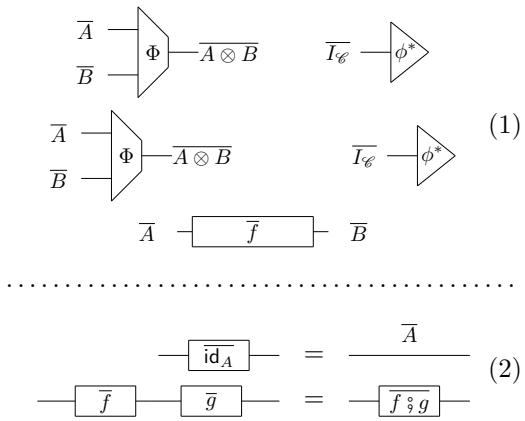
2 A graphical language for (non-strict) monoidal categories

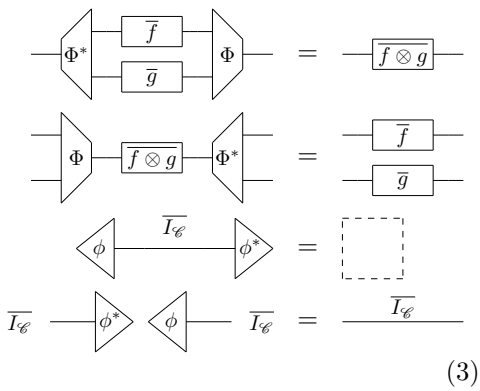
We assume familiarity with string diagrams for strict monoidal categories, see e.g. [20]. Let us fix an arbitrary (non-strict) monoidal category $\mathscr{C}$. We construct its *strictification* as the strict monoidal category $\overline{\mathscr{C}}$ defined as follows.

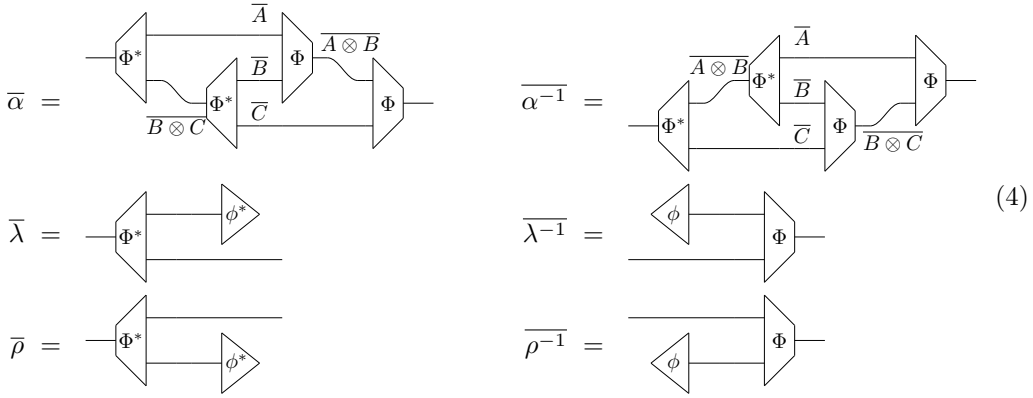
▶ **Definition 1.** *$(\overline{\mathscr{C}}, \bullet)$ is the strict monoidal category freely generated by:*

1. *Objects $\overline{A}$ for each $A \in \mathscr{C}$*
2. *Generators (1), with $\overline{f} : \overline{A} \to \overline{B}$ for each $f : A \to B \in \mathscr{C}$*
3. *functoriality equations (2)*
4. *adapter equations (3), and*
5. *associator/unitor equations (4)*

Note that because $\overline{\mathscr{C}}$ is strict by definition, we are entitled to use string diagrammatic notation. Thus, a morphism with m inputs and n outputs will have domain and codomain of the form $\overline{A_1} \bullet \cdots \bullet \overline{A_m}$ and $\overline{B_1} \bullet \cdots \bullet \overline{B_n}$, respectively.







(4)

This is a functorial construction, yielding a monoidal equivalence between $\mathscr{C}$ and $\overline{\mathscr{C}}$, as we will prove in Section 3. Note that although the category $\overline{\mathscr{C}}$ is essentially the same as that given by Mac Lane [18, p. 257], its construction differs in one key respect. Namely, to define his equivalent strict category, Mac Lane relies on the coherence theorem to define both composition of arrows and to ensure the functors in the equivalence are monoidal. In contrast, the adapter generators and equations of $\overline{\mathscr{C}}$ mean that Definition 1 does not require use of the coherence theorem, and can therefore be used to prove it.

The functoriality equations are so-called as they ensure functoriality of the construction. The "adapter" equations and "associator/unitor" equations further ensure this functor is *monoidal* and it forms one half of a *monoidal equivalence*. Sec. 3 will make it clear that these equations are essentially obtained by freely adding the morphisms required by the definition of a monoidal functor (2).

Besides its mathematical significance, the interest of this construction lies in providing a means of manipulating morphisms of non-strict monoidal categories graphically. In particular, the ϕ and ϕ^* generators can be used to explicitly summon and dispell the monoidal unit, while the Φ and Φ^* generators can be thought of as systematic ways of packing and unpacking wires into more complex wires with internal structure. The next subsections will showcase how this additional layer of structure can be useful in categorical models of computation.

2.1 Circuit Description Languages with Tuples

Categorical models of circuit description languages are a prime source of examples of monoidal categories, for instance combinational [17] or sequential [12] circuits. The graphical representation of circuits also fits naturally and intuitively the box-and-wire model used by string diagrams. More precisely, the circuit description languages in *loc. cit.* (and variations thereof) are instances of *strict* monoidal categories.

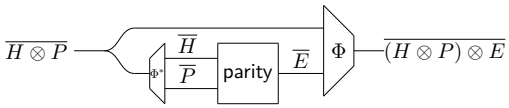
From the point of view of *expressiveness*, i.e. realising circuits with certain desired behaviours, the strict setting does not introduce any limitations. Consistent with this observation, standard hardware description languages (HDL) such as Verilog can also be modelled using a strict monoidal tensor. However, larger and more complex designs stand to benefit from the additional level of structure which a non-strict tensor can offer and, indeed, more modern HDLs, intended for more complex designs, such as SystemVerilog have syntactic facilities which require a non-strict tensor: *struct*s.

Consider the following simple example. Suppose that some circuitry is needed to process network packets, which consist of a header (of size $h = 96$ bits), a payload (of size $p = 896$ bits) and an error-correcting trailer (of size $e = 32$ bits). In the older Verilog language, the

header and the payload can be combined in a single, wider, data bus of $h + p = 992$ bits, but the two components can only be extracted using numerical indexing. This is a primitive form of "flattening" a data structure into an array, and in the more modern SystemVerilog it can be avoided by using a *struct*. This means that a data type of "message" (say m) can access its components as *fields* (projections), namely $m.h$ and $m.p$. Since structs can have other structs as fields the way in which the components are associated is relevant, which means that the tensor must no longer be strict.

On the other hand, "flattening" the structure of a data bus to an array of bits can be useful. In the current example, in computing the error-correcting code e, the way the message is partitioned into header and payload is no longer relevant, so it is convenient to unpack the tensor $h \otimes p$ into a flat array of $h + p$ wires from which an error-correcting code e is computed by a generic circuit of the appropriate width. Structures that can be flattened like this are called in SystemVerilog *packed structs*, and to model them properly both strict and non-strict tensorial facilities are required in the categorical model.

Finally, the error-correcting code can be packed with the original message into an error-correcting message with three components. It is obviously important to be able to retrieve the header, payload, and error-correcting code separately from the message, and it should be equally obvious that once the internal structure of the message is non-trivial a calculus of indices would be a complicated, awkward, and error-prone way to access the components.



Graphically, this circuit is represented above. In order to make this diagram completely formal, what we are using here is the $\overline{\mathscr{C}}$ construction described in Sec. 2 applied to one of the categories $\mathscr{C}$ of digital circuits (combinational or sequential) mentioned earlier. This gives us the best of both worlds: the "non-strictness" of circuits-with-tuples, and the graphical syntax of string diagrams.

2.1.0.1 Strictifying Strict Categories

The "strictification" procedure is not just useful for providing a graphical syntax for non-strict monoidal categories, but can also provide a more ergonomic syntax for monoidal categories that are *already* strict. Suppose we wish to work in Lafont's strict monoidal category of circuits [17], and suppose we would like to define the "parity" function used earlier. Using our construction, we can define it recursively as follows:

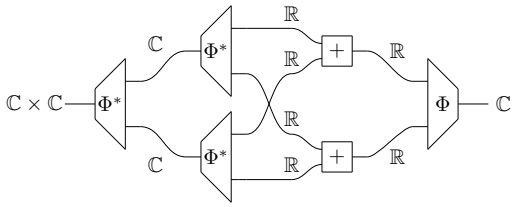
Notice that in the "base" language of Lafont's PROP of circuits we cannot truly depict this diagram, since there is no way to treat a bundle of n wires as a pair of 1 and $n - 1$ wires. To do this formally we require the adapter morphisms as defined in Section 2.

2.2 Programming Languages

Programming languages, largely based on the lambda calculus, commonly include product formation as a syntactic feature. Therefore, a graphical syntax based on its categorical model, as used for example in [1], needs to have a non-strict tensor. However, having only

the non-strict tensor leads to an awkward graphical syntax in which all generators have a single wire going in and a single wire going out. Diagrams in which the interfaces can be intermediated using lists of wires require mechanisms for strictification. This can be realised by applying the strictification construction to a Cartesian closed category, which will allow the expression of examples such as the one below.

Consider the simple task of summing two complex numbers, whose real and imaginary parts are encoded as floating-point numbers. That is, while we have a primitive type of reals, we model complex numbers as pairs $\mathbb{C} = \mathbb{R} \times \mathbb{R}$. A natural way to write such a program in a diagrammatic form is pictured below.



Even in categorical models of the simply-typed λ-calculus (STLC) without product, strictification has a role to play. As usual, this role is cloaked in informality which in some contexts can lead to ambiguity. STLC is interpreting by giving meaning to type judgements $\Gamma \vdash t : T$ with Γ a context, t a term, and T a type. The context $\Gamma = x_1 : T_1, \ldots, x_n : T_n$ is a list of typed variables which is interpreted as the tensor $T_1 \otimes \cdots \otimes T_n$, virtually always treated as if it were strict. This informal strictification can be problematic though when product types are used, as the objects T_i in the interpretation of the context also contain tensors. So the strictification must be fine-grained enough to allow only the flattening of those tensors representing the comma of the context, and not those of the product formation. Our approach offers this level of granularity.

2.3 Strict vs. Non-Strict String Diagrams

Our final example concerns the usability problems of non-strict diagrams *without* strictification and illustrate how our approach to strictification with packing and unpacking wires makes rigorous the intuition that formulating certain properties in terms of strict monoidal categories does not entail a loss of generality. Our example uses braided autonomous categories. Here, each object A has a dual A^*, there exists a family of isomorphisms $c_{A,B} : A \otimes B \to B \otimes A$ called braidings, and families of adjunctions $\eta_A : I \to A^* \otimes A$, $\epsilon : A \otimes A^* \to I$ with certain properties which we may elide in the formulation of the example. Consider the property of braided monoidal categories to be autonomous if and only if they are right-autonomous [15, Prop. 7.2]. The proof is formulated in terms of string diagrams in [20, Lem. 4.17], which makes it more intuitive.

The idea of the proof is to show that isomorphisms $b_A : A^{**} \to A$, $b_A^{-1} : A \to A^{**}$ can be constructed. They are defined as follows:

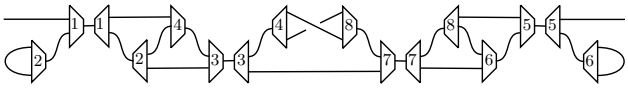
$$b_A = A^{**} \xrightarrow{\eta_A \otimes \mathsf{id}} A^* \otimes A \otimes A^{**} \xrightarrow{\mathsf{id} \otimes c_{A,A^{**}}} A^* \otimes A^{**} \otimes A \xrightarrow{\epsilon_{A^*} \otimes \mathsf{id}} A$$

$$b_A^{-1} = A \xrightarrow{\mathsf{id} \otimes \eta_{A^*}} A \otimes A^{**} \otimes A^* \xrightarrow{c^{-1}_{A^{**},A}} A^{**} \otimes A \otimes A^* \xrightarrow{\mathsf{id} \otimes \epsilon_A} A^{**}.$$

The fact that $b_A ; b_A^{-1} = \mathsf{id}$ becomes elegantly obvious when the terms are rendered as string diagrams which can be manipulated graphically:

The exposition includes the standard caveat that "Here we have written, without loss of generality, as if [the category] were strict monoidal." We shall now show, graphically, that this is indeed the case.

First we note that in the non-strict setting (without strictification) all string diagrams must be equipped with gadgets that make sure that there is a single wire on the left, and a single wire on the right. These gadgets are of course the bundlers and unbundlers introduced earlier. Therefore, in the non-strict setting, taking into account all the relevant associators, the diagram for b_A becomes much more complicated, denying the intuitiveness we expect from a graphical notation (see below).



This is why a naive approach to non-strict string diagram construction is not effective. However, the complications are only an artefact of the construction of the diagram in a purely non-strict setting. The strictification equations come to rescue and, in this case, cancel out all bundler-unbundler pairs in the order indicated by the numerical labels attached to them, resulting in exactly the same diagram of b_A that was constructed in the strict setting. So, indeed, working in the strict setting implied no loss of generality!

3 Strictness

We now show that $\mathscr{C}$ is monoidally equivalent to $\overline{\mathscr{C}}$, constituting a proof of Mac Lane's strictness theorem, since $\mathscr{C}$ is an arbitrary monoidal category. Our approach is to define monoidal functors $\mathcal{S} : \mathscr{C} \to \overline{\mathscr{C}} : \mathcal{N}$, and we begin by recalling the definition of monoidal functor.

▶ **Definition 2** (Monoidal Functor). *Let $(\mathscr{C}, \otimes, I_{\mathscr{C}})$ and $(\mathscr{D}, \bullet, I_{\mathscr{D}})$ be monoidal categories. A monoidal functor is a functor $F : \mathscr{C} \to \mathscr{D}$ equipped with natural isomorphisms $\Phi_{X,Y} : F(X) \bullet F(Y) \to F(X \otimes Y)$ and $\phi : I_{\mathscr{D}} \to F(I_{\mathscr{C}})$ such that the following diagrams commute for all objects $A, B, C \in \mathscr{C}$.*

$$\begin{array}{ccc}
(F(A) \bullet F(B)) \bullet F(C) & \xleftarrow{\alpha_{\mathscr{D}}} & F(A) \bullet (F(B) \bullet F(C)) \\
\downarrow{\scriptstyle \Phi_{A,B} \bullet \mathsf{id}_{F(C)}} & & \downarrow{\scriptstyle \mathsf{id}_{F(A)} \bullet \Phi_{B,C}} \\
F(A \otimes B) \bullet F(C) & & F(A) \bullet F(B \otimes C) \\
\downarrow{\scriptstyle \Phi_{A\otimes B,C}} & & \downarrow{\scriptstyle \Phi_{A,B\otimes C}} \\
F((A \otimes B) \otimes C) & \xleftarrow[F(\alpha_{\mathscr{C}})]{} & F(A \otimes (B \otimes C))
\end{array} \tag{5}$$

$$\begin{array}{ccc}
F(A) \bullet I_{\mathscr{D}} & \xrightarrow{\mathsf{id}_{F(A)} \bullet \phi} & F(A) \bullet F(I_{\mathscr{C}}) \\
\downarrow{\scriptstyle \rho_{\mathscr{D}}} & & \downarrow{\scriptstyle \Phi_{A,I_{\mathscr{C}}}} \\
F(A) & \xleftarrow[F(\rho_{\mathscr{C}})]{} & F(A \otimes I_{\mathscr{C}})
\end{array}
\qquad
\begin{array}{ccc}
I_{\mathscr{D}} \bullet F(B) & \xrightarrow{\phi \bullet \mathsf{id}_{F(B)}} & F(I_{\mathscr{C}}) \bullet F(B) \\
\downarrow{\scriptstyle \lambda_{\mathscr{D}}} & & \downarrow{\scriptstyle \Phi_{I_{\mathscr{C}},B}} \\
F(B) & \xleftarrow[F(\lambda_{\mathscr{C}})]{} & F(I_{\mathscr{C}} \otimes B)
\end{array} \tag{6}$$

With this definition it is straightforward to see how to define a monoidal functor from $\mathscr{C}$ to $\overline{\mathscr{C}}$.

▶ **Definition 3.** *Let $\mathcal{S} : \mathscr{C} \to \overline{\mathscr{C}}$ be the* strictification functor *defined on objects and morphisms as $\mathcal{S}(A) := \overline{A}$ and $\mathcal{S}(f) := \overline{f}$, respectively*

▶ **Proposition 4.** *$(\mathcal{S}, \Phi, \phi)$ is a monoidal functor.*

Proof. $\mathcal{S}$ preserves identities and composition (and is therefore a functor) by the functor equations (2):

$$\mathcal{S}(\mathsf{id}_A) = \overline{\mathsf{id}_A} = \mathsf{id}_{\overline{A}} \qquad \mathcal{S}(f \fatsemi g) = \overline{f \fatsemi g} = \overline{f} \fatsemi \overline{g} = \mathcal{S}(f) \fatsemi \mathcal{S}(g)$$

It is a *monoidal* functor using the adapter generators $\Phi = \Phi$ and $\phi = \phi$ from (1). For this to work, we must have that Φ is a natural isomorphism and ϕ an isomorphism, respectively. This is a straightforward consequence of the adapter equations (3): $\Phi^* \fatsemi (\overline{f} \bullet \overline{g}) = \Phi^* \fatsemi (\overline{f} \bullet \overline{g}) \fatsemi \Phi \fatsemi \Phi^* = \overline{f \otimes g} \fatsemi \Phi^*$ and $\phi \fatsemi \phi^* = \mathsf{id}$ by definition. Similarly, we require that the diagrams of (5) and (6) commute. Again, this is precisely what the the associator/unitor equations (4) state, and so $\mathcal{S}$ is a monoidal functor. ◀

▶ Remark 5. Notice that $\overline{\mathscr{C}}$ is *defined* by freely adding the requirements of Definition 2. Generators Φ and ϕ and equations (3) give the natural isomorphism Φ and isomorphism ϕ, while the commuting diagrams (5) and (6) are precisely the "associator/unitor" equations (4).

We can now define the other half of the monoidal equivalence $\mathcal{S} \dashv \mathcal{N}$. In doing so, we'll make use of the fact that morphisms of a monoidal category can be written in a "sequential normal form" (Appendix A), i.e. as a series of "slices"

$$(\mathsf{id} \otimes g_1 \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g_2 \otimes \mathsf{id}) \fatsemi \ldots \fatsemi (\mathsf{id} \otimes g_n \otimes \mathsf{id})$$

where each g_i is a generator. We take advantage of this form to define $\mathcal{N}$: our definition is defined on "slices" $\mathsf{id}_X \bullet q \bullet \mathsf{id}_Y$ for some generator q, and then freely on composition so that $\mathcal{N}(f \fatsemi g) = \mathcal{N}(f) \fatsemi \mathcal{N}(g)$.

▶ **Definition 6.** *We define the nonstrictification functor $\mathcal{N} : \overline{\mathscr{C}} \to \mathscr{C}$ inductively on objects:*

$$\mathcal{N}(I_{\overline{\mathscr{C}}}) := I_{\mathscr{C}} \qquad \mathcal{N}(\overline{A}) := A \qquad \mathcal{N}(\overline{A} \bullet R) := A \otimes \mathcal{N}(R)$$

And on morphisms we give a recursive definition, with the following base cases:

$$\begin{aligned}
\mathcal{N}(\mathsf{id}_{I_{\overline{\mathscr{C}}}}) &:= \mathsf{id}_{I_{\mathscr{C}}} \\
\mathcal{N}(\overline{f}) &:= f \\
\mathcal{N}(\Phi_{A,B}) &:= \mathsf{id}_{A \otimes B} = \mathcal{N}(\Phi^*_{A,B}) \\
\mathcal{N}(\phi) &:= \mathsf{id}_{I_{\mathscr{C}}} = \mathcal{N}(\phi^*)
\end{aligned}
\qquad
\begin{aligned}
\mathcal{N}(\overline{f} \bullet \mathsf{id}_Y) &:= f \otimes \mathsf{id}_{\mathcal{N}(Y)} \\
\mathcal{N}(\Phi_{A,B} \bullet \mathsf{id}_Y) &:= \alpha_{A,B,\mathcal{N}(Y)} \\
\mathcal{N}(\Phi^*_{A,B} \bullet \mathsf{id}_Y) &:= \alpha^{-1}_{A,B,\mathcal{N}(Y)} \\
\mathcal{N}(\phi \bullet \mathsf{id}_Y) &:= \lambda^{-1}_{\mathcal{N}(Y)} \\
\mathcal{N}(\phi^* \bullet \mathsf{id}_Y) &:= \lambda_{\mathcal{N}(Y)}
\end{aligned}
\qquad
\begin{aligned}
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \overline{f}) &:= \mathsf{id}_A \otimes f \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \Phi_{B,C}) &:= \mathsf{id}_{A \otimes (B \otimes C)} \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \Phi^*_{B,C}) &:= \mathsf{id}_{A \otimes (B \otimes C)} \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \phi) &:= \rho^{-1}_A \\
\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet \phi^*) &:= \rho_A
\end{aligned}$$

With a single recursive case, for $q \in \{\Phi, \phi, \Phi^*, \phi^*, \mathsf{id}_{\overline{Q}}\}$

$$\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet q \bullet r) := \mathsf{id}_A \otimes \mathcal{N}(q \bullet r)$$

Finally take $\mathcal{N}(f \fatsemi g) := \mathcal{N}(f) \fatsemi \mathcal{N}(g)$.

This definition is well defined with respect to the equations of Definition 1; we give a proof in Appendix B, where we also note that $N(f)$ is the same regardless of which "sequential normal form" decomposition we choose for f.

▶ Remark 7. The definition of $\mathcal{N}$ can be explained more intuitively in terms of programming. If we think of each "slice" of the sequential normal form as a list of primitive arrows of $\mathscr{C}$, then the definition of $\mathcal{N}$ is essentially a list recursion in which we have a separate case for 1, 2, and n-element lists.

Now we will show that $\mathcal{N}$ is a *monoidal* functor. To do this, we must specify the "coherence maps": a natural isomorphism $\Psi_{X,Y} : \mathcal{N}(X) \otimes \mathcal{N}(Y) \to \mathcal{N}(X \bullet Y)$ and isomorphism $\psi : I_{\mathscr{C}} \to \mathcal{N}(I_{\overline{\mathscr{C}}})$ as mandated by Definition 2.

▶ **Definition 8.** *We define Ψ, the coherence natural isomorphism for $\mathcal{N}$, in the following cases:*

$$\Psi_{I_{\overline{\mathscr{C}}},I_{\overline{\mathscr{C}}}} := \lambda_{I_{\mathscr{C}}} = \rho_{I_{\mathscr{C}}} \qquad \Psi_{X,I_{\overline{\mathscr{C}}}} := \rho_{\mathcal{N}(X)} \qquad \Psi_{I_{\overline{\mathscr{C}}},Y} := \lambda_{\mathcal{N}(Y)}$$

$$\Psi_{\overline{A},Y} := \mathsf{id}_{A \otimes \mathcal{N}(Y)} \qquad \Psi_{\overline{A} \bullet X,Y} := \alpha^{-1}_{A,\mathcal{N}(X),\mathcal{N}(Y)} \fatsemi (\mathsf{id}_A \otimes \Psi_{X,Y})$$

▶ **Definition 9.** *The coherence isomorphism ψ for $\mathcal{N}$ is defined as follows:*

$$\psi_{I_{\mathscr{C}}} := \mathsf{id}_{I_{\mathscr{C}}}$$

▶ Remark 10. Note that both $\lambda_{I_{\mathscr{C}}}$ and $\rho_{I_{\mathscr{C}}}$ have the correct type as a choice for $\Psi_{I_{\overline{\mathscr{C}}},I_{\overline{\mathscr{C}}}}$. In fact, they are equal: unitors coincide at the unit object, i.e. $\lambda_{I_{\mathscr{C}}} = \rho_{I_{\mathscr{C}}}$, as noted in [10, Corollary 2.2.5].

▶ **Proposition 11.** *$(\mathcal{N}, \Psi, \psi)$ is a monoidal functor.*

Proof. It is clear that Ψ and ψ are natural isomorphisms since they are both composites of natural isomorphisms. Thus it remains to check the diagrams of Definition 2 commute.

The squares (6) commute because $\psi = \mathsf{id}$, and $\Psi_{A,I_{\overline{\mathscr{C}}}} = \rho$ and $\Psi_{I_{\overline{\mathscr{C}}},B} = \lambda$ by definition.

Now let us check that the hexagon (5) commutes. Note that in the following we use that $\mathcal{N}(\alpha_{\overline{\mathscr{C}}}) = \mathsf{id}$, because $\overline{\mathscr{C}}$ is strict, and so the hexagon axiom becomes a pentagon.

We will approach the problem inductively, checking base cases where $A = I$ and $A = \overline{A}$, and finally the inductive step with $A = \overline{A} \bullet R$. Let us begin with $A = I$, and taking the outer path of the hexagon we calculate as follows:

$$\begin{aligned}
&(\mathsf{id}_{I_{\mathscr{C}}} \otimes \Psi_{B,C}) \fatsemi \Psi_{I_{\mathscr{C}},B \bullet C} \fatsemi \Psi^{-1}_{B,C} \fatsemi (\Psi_{I_{\mathscr{C}},B} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = (\mathsf{id}_{I_{\mathscr{C}}} \otimes \Psi_{B,C}) \fatsemi \lambda_{\mathcal{N}(B \bullet C)} \fatsemi \Psi^{-1}_{B,C} \fatsemi (\lambda_{\mathcal{N}(B)} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = \lambda_{\mathcal{N}(B) \otimes \mathcal{N}(C)} \fatsemi (\lambda_{\mathcal{N}(B)} \otimes \mathsf{id}_{\mathcal{N}(C)})^{-1} \\
&\qquad = \alpha_{I_{\mathscr{C}},\mathcal{N}(B),\mathcal{N}(C)}
\end{aligned}$$

Wherein we expanded the definition of Ψ, then used naturality of $\Psi_{B,C}$ before applying the monoidal triangle lemma of [10, (2.12)].

Now consider the second base case, where A is the "singleton list" $\overline{A}$. In this case, the hexagon diagram commutes immediately because $\Psi_{\overline{A},B} = \mathsf{id}_{\overline{A} \otimes \mathcal{N}(B)}$ and $\Psi_{\overline{A},B \bullet C} = \mathsf{id}_{\overline{A} \otimes \mathcal{N}(B \bullet C)}$. More explicitly, we calculate as follows, starting again with the outer path of the hexagon and expanding definitions:

$$\begin{aligned}
&(\mathsf{id}_A \otimes \Psi_{B,C}) \fatsemi \Psi_{A,B \bullet C} \fatsemi \Psi^{-1}_{A \bullet B,C} \fatsemi (\Psi_{\overline{A},B} \otimes \mathsf{id}_{\mathcal{N}(C)}) \\
&\qquad = (\mathsf{id}_A \otimes \Psi_{B,C}) \fatsemi (\mathsf{id}_A \otimes \Psi_{B,C})^{-1} \fatsemi \alpha_{A,\mathcal{N}(B),\mathcal{N}(C)} \\
&\qquad = \alpha_{A,\mathcal{N}(B),\mathcal{N}(C)}
\end{aligned}$$

Finally let us prove the inductive step. Assume that the hexagon commutes for objects R, B, C, giving us the equation

$$\Psi_{R,B\bullet C} \fatsemi \Psi^{-1}_{R\bullet B,C} = (\mathsf{id}_{\mathcal{N}(R)} \otimes \Psi^{-1}_{B,C}) \fatsemi \alpha_{\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)} \fatsemi (\Psi_{R,B} \otimes \mathsf{id}_{\mathcal{N}(C)})$$

We may then rewrite the following subterm of the monoidal hexagon as follows:

$$\mathsf{id}_A \otimes (\Psi_{R,B\bullet C} \fatsemi \Psi^{-1}_{R\bullet B,C}) = \mathsf{id}_A \otimes (\mathsf{id}_{\mathcal{N}(R)} \otimes \Psi^{-1}_{B,C}) \fatsemi \mathsf{id}_A \otimes \alpha_{\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)} \fatsemi \mathsf{id}_A \otimes (\Psi_{R,B} \otimes \mathsf{id}_{\mathcal{N}(C)})$$

We can then rewrite $\mathsf{id}_A \otimes \alpha_{\mathcal{N}(R),\mathcal{N}(B)\mathcal{N}(C)}$ using the monoidal category pentagon axiom, and then use naturality of α to reduce the outer path of the monoidal hexagon until we are left with $\alpha_{A\otimes\mathcal{N}(R),\mathcal{N}(B),\mathcal{N}(C)}$, as required. ◀

Finally, we must check that $\mathcal{S}$ and $\mathcal{N}$ indeed form an equivalence. First, recall the definition

▶ **Definition 12** (Equivalence of categories)**.**
An equivalence is a pair of functors $\mathscr{C} \underset{G}{\overset{F}{\rightleftarrows}} \mathscr{D}$ *and a pair of natural isomorphisms* $\eta : \mathsf{id}_{\mathscr{C}} \to G \circ F$ *and* $\epsilon : F \circ G \to \mathsf{id}_{\mathscr{D}}$*.*

We begin by showing naturality of η.

▶ **Proposition 13.** $\mathcal{N} \circ \mathcal{S} = \mathsf{id}_{\mathscr{C}}$*.*

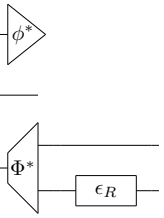
Proof. $\mathcal{N}(\mathcal{S}(f)) = \mathcal{N}(\overline{f}) = f = \mathsf{id}_{\overline{\mathscr{C}}}(f)$ ◀

▶ Remark 14. Note that Proposition 13 shows that the composite $\mathcal{N} \circ \mathcal{S}$ is actually *equal* to the identity functor, and thus $\eta_A = \mathsf{id}_A$. In fact, this will make the composite of the two functors a *split idempotent*.

Now we prove naturality of ϵ. This proof is somewhat more involved: unlike 13, the composite $\mathcal{S} \circ \mathcal{N}$ is merely isomorphic to the identity functor, not equal on the nose. Thus, we begin with an inductive definition:

▶ **Definition 15.** *We define the (monoidal) natural isomorphism* $\epsilon : \mathcal{S} \circ \mathcal{N} \to \mathsf{id}_{\overline{\mathscr{C}}}$ *for the composite* $\mathcal{S} \circ \mathcal{N}$ *inductively:*

$$\begin{aligned} \epsilon_{I_{\overline{\mathscr{C}}}} &:= \phi^* \\ \epsilon_{\overline{A}} &:= \mathsf{id}_{\overline{A}} \\ \epsilon_{\overline{A}\bullet R} &:= \Phi^* \fatsemi (\mathsf{id}_{\overline{A}} \bullet \epsilon_R) \end{aligned} \qquad = \quad \tag{7}$$



▶ **Proposition 16.** *If* ϵ *is natural for* f *and* g*, then it is natural for* $f \fatsemi g$*.*

Proof. Take morphisms $f : X \to Y$ and $g : Y \to Z$. By assumption, we have:

$$\mathcal{S}(\mathcal{N}(f)) = \epsilon_X \fatsemi f \fatsemi \epsilon_Y^{-1} \qquad \mathcal{S}(\mathcal{N}(g)) = \epsilon_Y \fatsemi g \fatsemi \epsilon_Z^{-1}$$

from which we can derive

$$\begin{aligned} \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(fg)) \fatsemi \epsilon_Z = \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(f) \fatsemi \mathcal{N}(g)) \fatsemi \epsilon_Z &= \epsilon_X^{-1} \fatsemi \mathcal{S}(\mathcal{N}(f)) \fatsemi \mathcal{S}(\mathcal{N}(g)) \fatsemi \epsilon_Z \\ &= \epsilon_X^{-1} \fatsemi \epsilon_X \fatsemi f \fatsemi \epsilon_Y^{-1} \fatsemi \epsilon_Y \fatsemi g \fatsemi \epsilon_Z^{-1} \fatsemi \epsilon_Z \\ &= f \fatsemi g \end{aligned} \tag{8}$$

as required. ◀

▶ **Proposition 17.** *$\epsilon : \mathcal{S} \circ \mathcal{N} \to \mathsf{id}_{\overline{\mathscr{C}}}$ is a monoidal natural isomorphism.*

Proof. We begin by showing naturality inductively, having already proven the inductive step for composition in Proposition 16. We again use Proposition 36–that each morphism f in $\overline{\mathscr{C}}$ can be decomposed into "slices" $f = t_1 \fatsemi \ldots \fatsemi t_n$ with each t_i of the form $\mathsf{id}_X \bullet g_i \bullet \mathsf{id}_Y$, with each $g_i : A \to B$ a generator. It thus suffices to prove that $t = \epsilon^{-1}_{X \bullet A \bullet Y} \fatsemi \mathcal{S}(\mathcal{N}(t)) \fatsemi \epsilon_{X \bullet B \bullet Y}$ for an arbitrary "slice" t. One can check this by a second induction whose base case and inductive step correspond to the definition of $\mathcal{N}$ (Definition 6). To be precise, one can check this property graphically for each base case $\mathcal{N}(\mathsf{id}_{\overline{I_{\mathscr{C}}}}) \ldots \mathcal{N}(\mathsf{id}_{\overline{A}})$, and additionally for the inductive step $\mathcal{N}(\mathsf{id}_{\overline{A}} \bullet q \bullet r)$. Finally, note that ϵ is indeed a *monoidal* natural transformation, which can be verified by another straightforward induction. ◀

▶ **Theorem 18** (Mac Lane's Strictness Theorem)**.** *For any monoidal category $\mathscr{C}$ there is a monoidally equivalent strict category.*

Proof. $\mathcal{S}$ and $\mathcal{N}$ are monoidal functors by Propositions 4 and 11, and they form a monoidal equivalence with η and ϵ by Propositions 13 and 17. Since $\mathscr{C}$ was arbitrary, the proof is complete. ◀

Note that in contrast to Mac Lane's proof of Theorem 18, we make no reference to the coherence theorem. We can therefore make use of the strictness theorem to prove coherence, which is the subject Section 4.

4 Coherence

We can now give an elementary proof of Mac Lane's *coherence theorem*. In [18], Mac Lane gives his theorem in two parts: Theorem 1 [18, p. 166] and its corollary [18, p. 169]. The "meat" of the proof is in the former part, corresponding to our Section 4.1. We do not give a proof of Mac Lane's corollary here, but the interested reader may find a graphical exposition in the extended version of this paper [23].

Mac Lane begins by defining a certain preorder $\mathscr{W}$, which he then shows enjoys the following property:

▶ **Theorem 19** (Mac Lane's Coherence Theorem [18, p. 166])**.** *Let $\mathscr{M}$ be an arbitrary monoidal category, and let M be an object of $\mathscr{M}$. Then there is a unique strict monoidal functor $\mathscr{W} \to \mathscr{M}$ such that $W \mapsto M$.*

In contrast, we will define $\mathscr{W}$ so this unique functor is easy to construct, and then use $\overline{\mathscr{W}}$ to give a *graphical proof* that $\mathscr{W}$ is a preorder. Note that the monoidal functor in question is *strict*, so its coherence maps are identities.

4.1 The free monoidal category on one generator

We begin by defining $\mathscr{W}$. Again, recall that our definition differs from Mac Lane; we will later show that this definition indeed yields a preorder in order to guarantee that we indeed prove the same theorem.

▶ **Definition 20.** *We define $\mathscr{W}$ as the monoidal category freely generated by a single object W and no morphisms except those required by the definition of a monoidal category.* [1]

[1] Mac Lane denotes the generating object as $(-)$ to suggest an "empty place". We follow Peter Hines' convention [14] and use W instead.

▶ Remark 21. The objects of $\mathscr{W}$ are $I_{\mathscr{W}}$, W, and their tensor products. The arrows are $\mathsf{id}, \rho, \lambda, \alpha$ and their composites and tensor products.

It is now clear that the statement of Mac Lane's Theorem 1 holds for our definition of $\mathscr{W}$:

▶ **Proposition 22.** *Given an arbitrary monoidal category $\mathscr{M}$ and object $M \in \mathscr{M}$, there is a unique strict monoidal functor $\mathscr{W} \to \mathscr{M}$ with $W \mapsto M$.*

Proof. Suppose $\mathsf{U} : \mathscr{W} \to \mathscr{M}$ is such a (strict) monoidal functor. Then we must have that $\mathsf{U}(W) = M$ by assumption, and

$$\mathsf{U}(I) = I \quad \mathsf{U}(A \otimes M) = \mathsf{U}(A) \otimes \mathsf{U}(M) \quad \mathsf{U}(f) = f, f \in \{\alpha, \lambda, \rho, \mathsf{id}\} \quad \mathsf{U}(f \otimes g) = \mathsf{U}(f) \otimes \mathsf{U}(g)$$

because U is strict. But this accounts for all objects and morphisms of $\mathscr{W}$, and so U must be unique. ◀

However, to constitute a proof of the coherence theorem we must now *prove* that $\mathscr{W}$ is a preorder. Our argument proceeds in three main steps. We will show the following:

1. For any monoidal category $\mathscr{C}$, If $\overline{\mathscr{C}}$ is a preorder, then so is $\mathscr{C}$
2. $\overline{\mathscr{W}}$ is generated solely by adapters $\{\Phi, \phi\}$ and their inverses.
3. $\overline{\mathscr{W}}$ is a preorder (which we prove graphically)

The first two steps are straightforward; we address them now. The third requires more work, and is contained in Section 4.2.

▶ **Proposition 23.** *If $\overline{\mathscr{C}}$ is a preorder, then so is $\mathscr{C}$.*

Proof. Let $f, g : \mathscr{C}(A, B)$. Recall that $\mathcal{N} \circ \mathcal{S} = \mathsf{id}$, and so we can derive $f = \mathcal{N}(\mathcal{S}(f)) = \mathcal{N}(\mathcal{S}(g)) = g$ where we used that $\mathcal{S}(f) = \mathcal{S}(g)$ because $\overline{\mathscr{C}}$ is a preorder. ◀

Another lemma shows we can reason about $\overline{\mathscr{W}}$ by considering only adapters:

▶ **Proposition 24.** *$\overline{\mathscr{W}}$ is generated by Φ, ϕ and their inverses.*

Proof. Arrows of $\overline{\mathscr{W}}$ are by definition either adapters Φ, ϕ, their inverses, or morphisms $\overline{f}$ for some $f \in \mathscr{W}$. But note that all such $f \in \mathscr{W}$ are either $\mathsf{id}, \rho, \lambda, \alpha$ or their composites. It is clear that each of λ, ρ, α can each be written as adapters by equations (4), so it remains to show that composites of such morphisms can also be written this way.

That is, we must show that $\mathcal{S}(f \otimes g)$ can be expressed using only adapters and their composites. This can be proved inductively: if $\mathcal{S}(f), \mathcal{S}(g)$ can be expressed using adapters, then so too can compositions $\mathcal{S}(f \fatsemi g) = \mathcal{S}(f) \fatsemi \mathcal{S}(g)$ and tensors $\mathcal{S}(f \otimes g) = \Phi \fatsemi (\mathcal{S}(f) \bullet \mathcal{S}(g)) \fatsemi \Phi^*$.

Thus every morphism of $\overline{\mathscr{W}}$ can be expressed in terms of adapters, and so the category can be said to be *generated* by (only) adapters. ◀

4.2 Graphical proof that $\overline{\mathscr{W}}$ is a preorder

We can now prove graphically that $\overline{\mathscr{W}}$ is a preorder using a normal form argument. Our approach is as follows:

1. Define for each object a `size` in $\mathbb{N}$ (Definition 25)
2. Prove all morphisms in $\overline{\mathscr{W}}$ go between objects of the same size (Proposition 26)
3. Define a canonical arrow $\mathsf{can}(A, B)$ between any two objects of the same size (Definition 31)
4. Show that any arrow is equal to the canonical one (Proposition 33)

Note that we make heavy use of Proposition 24, which lets us reason about $\overline{\mathscr{W}}$ inductively in terms of adapters and their tensors and composites.

We begin–following Mac Lane–by defining the *size* of an object (the same as Mac Lane's notion of *length* [18, p. 165]) as follows:

▶ **Definition 25.** *We define the* `size` *of an object as the number of occurrences of* W*, defined inductively:*

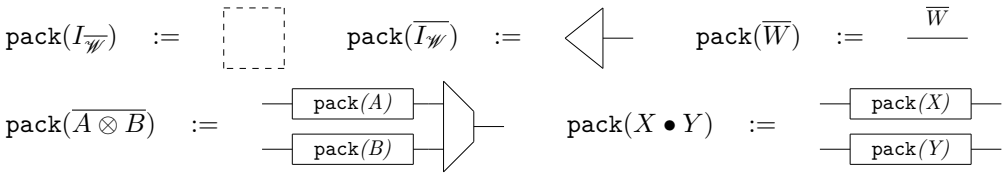
$$\texttt{size}(I_{\overline{\mathscr{W}}}) := 0 \qquad \texttt{size}(\overline{I_{\mathscr{W}}}) := 0 \qquad \texttt{size}(\overline{W}) := 1$$
$$\texttt{size}(\overline{A \otimes B}) := \texttt{size}(A) + \texttt{size}(B) \qquad \texttt{size}(X \bullet Y) := \texttt{size}(X) + \texttt{size}(Y)$$

▶ **Proposition 26.** $\overline{\mathscr{W}}$ *morphisms preserve* `size`*:*
If $f : A \to B$ *is a morphism in* $\overline{\mathscr{W}}$*, then* $\texttt{size}(A) = \texttt{size}(B)$*.*

Proof. Induction on morphisms. ◀

We will define the canonical arrow $\texttt{can}(A, B)$ in two halves, `pack` and `unpack`. To do so, we will first need some additional definitions.

▶ **Definition 27.** *We define the "packing" and "unpacking" morphisms* `pack` *and* `unpack` *in terms of objects of* $\overline{\mathscr{W}}$*. Let* $A \in \overline{\mathscr{W}}$ *be an object. Then* $\texttt{pack}(A)$ *is the morphism defined inductively as follows:*



And define $\texttt{unpack}(A)$ *as* $\texttt{pack}(A)^{-1}$*.*

▶ Remark 28. It can be more intuitive to define `unpack` first, thinking of it as the adapter which removes extraneous $I_{\mathscr{C}}$ objects and "normalises" the object into a flat array of $\overline{W}$ objects. In this view, `pack` is the adapter morphism taking a fixed number of $\overline{W}$ objects and assembling them into a certain bracketing, with unit objects inserted as appropriate.

In Definition 27 we implicitly used that $\overline{\mathscr{W}}$ is a groupoid to define `unpack`, which we now prove:

▶ **Proposition 29.** $\overline{\mathscr{W}}$ *is a groupoid.*

Proof. Generators and identities have inverses by Definition 1, which allows an inductive definition for tensor and composition, i.e. $(f \fatsemi g)^{-1} = g^{-1}f^{-1}$ and $(f \bullet g)^{-1} = f^{-1} \bullet g^{-1}$ respectively. ◀

Now, in order to define the canonical arrow as a composition of `pack` and `unpack`, we will need the following lemma which states that for objects of the same size, we can compose their `unpack` and `pack` morphisms.

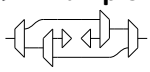
▶ **Proposition 30.** $\texttt{pack}(A) : \overline{W}^{\texttt{size}(A)} \to A$*.*
In other words, for an object A *of size* n*, the domain of* $\texttt{pack}(A)$ *is the* n*-fold* $\bullet$*-tensoring of* $\overline{W}$*.*

Proof. Simple induction on objects (the domain of each $\mathtt{pack}(A)$ is either $I_{\overline{\mathscr{W}}}$, $\overline{W}^k$ or a tensoring of terms) ◀

▶ **Definition 31.** *To each pair of objects A, B of the same size, we can define a canonical arrow as follows:*

$$\mathsf{can}(A, B) := \mathtt{unpack}(A) \fatsemi \mathtt{pack}(B)$$

Note that the composition of Definition 31 is well-typed because $\mathtt{size}(A) = \mathtt{size}(B)$ by Proposition 26: $\mathsf{cod}(\mathtt{unpack}(A)) = \overline{W}^{\mathtt{size}(A)} = \overline{W}^{\mathtt{size}(B)} = \mathsf{dom}(\mathtt{pack}(B))$

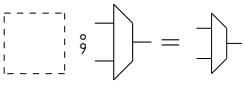
▶ **Example 32.** The canonical arrow between $W \otimes (I_{\mathscr{W}} \otimes W)$ and $(W \otimes I_{\mathscr{W}}) \otimes W$ is  . Note that this is equal to the associator $\alpha_{W,I_{\mathscr{W}},W}$.

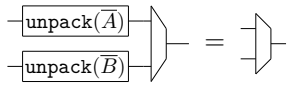
We can now show that every morphism $f : A \to B$ in $\overline{\mathscr{W}}$ is equal to $\mathsf{can}(A, B)$.

▶ **Proposition 33.** *$f = \mathtt{unpack}(A) \fatsemi \mathtt{pack}(B)$ for all $f : A \to B$ in $\overline{\mathscr{W}}$.*

Proof. By induction. On the base case–generators–the proof is straightforward; we give it for identities and generators  and  , with the proofs for inverse generators following by a symmetric argument.

$$\mathsf{can}(X, X) = \mathtt{unpack}(X) \fatsemi \mathtt{pack}(X) = \mathtt{pack}(X)^{-1} \fatsemi \mathtt{pack}(X) = \mathsf{id}_X$$

$$\mathsf{can}(I_{\overline{\mathscr{W}}}, \overline{I_{\mathscr{W}}}) = \mathtt{unpack}(I_{\overline{\mathscr{W}}}) \fatsemi \mathtt{pack}(\overline{I_{\mathscr{W}}}) =$$ 

$$\mathsf{can}(\overline{A} \bullet \overline{B}, \overline{A \otimes B}) = \mathtt{unpack}(\overline{A} \bullet \overline{B}) \fatsemi \mathtt{pack}(\overline{A \otimes B}) =$$ 

The composition of canonical morphisms is canonical:

$$\begin{aligned}
\mathsf{can}(X, Y) \fatsemi \mathsf{can}(Y, Z) &= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Y) \fatsemi \mathtt{unpack}(Y) \fatsemi \mathtt{pack}(Z) \\
&= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Y) \fatsemi \mathtt{pack}(Y)^{-1} \fatsemi \mathtt{pack}(Z) \\
&= \mathtt{unpack}(X) \fatsemi \mathtt{pack}(Z) \\
&= \mathsf{can}(X, Z)
\end{aligned}$$

And so is the tensor product:

$$\mathsf{can}(X_1, Y_1) \bullet \mathsf{can}(X_2, Y_2) =$$ 

$$= \mathtt{unpack}(X_1 \bullet X_2) \fatsemi \mathtt{pack}(Y_1 \bullet Y_2)$$
$$= \mathsf{can}(X_1 \bullet X_2, Y_1 \bullet Y_2)$$ ◀

▶ **Proposition 34.** *$\overline{\mathscr{W}}$ is a preorder.*

Proof. By Proposition 26 we know that all morphisms $f : A \to B$ have the property that $\mathtt{size}(A) = \mathtt{size}(B)$. We then define for any such objects a canonical morphism $\mathsf{can}(A, B)$ in Definition 31. This canonical isomorphism is unique by Definition 33, and so $\overline{\mathscr{W}}$ is a preorder. ◀

Since we have now proven that $\overline{\mathscr{W}}$ is a preorder, it is now straightforward to prove Theorem 19. Note that this is essentially the opposite of the approach taken by Mac Lane, who *defines* a preorder, and then shows the existence of a unique strict monoidal functor.

Proof of Theorem 19. By Proposition 22 there is a unique, strict monoidal functor from $\mathscr{W}$ to an arbitrary monoidal category $\mathscr{M}$ with $W \mapsto A$ for some $A \in \mathscr{M}$. Moreover, $\overline{\mathscr{W}}$ is a preorder, and so by Proposition 23, so is $\mathscr{W}$. ◀

A first consequence of the coherence theorem is that $\mathcal{N}$ is a *strict* inverse to $\mathcal{S}$ for morphisms $f : \overline{A} \to \overline{B}$ in $\overline{\mathscr{W}}$.

▶ **Proposition 35.** *If $f : \overline{A} \to \overline{B}$ then $\mathcal{S}(\mathcal{N}(f)) = f$.*

Proof. We know that for any $A \in \mathscr{W}$ that $\mathcal{N}(\overline{A}) = A$. Thus for $f : \overline{A} \to \overline{B}$ we have $\mathcal{N}(f) : A \to B$ and thus $\mathcal{S}(\mathcal{N}(f)) : \overline{A} \to \overline{B}$. But $\overline{\mathscr{W}}$ is a preorder, so we have $\mathcal{S}(\mathcal{N}(f)) = f$. ◀

Proposition 35 guarantees that any morphism of this type formed from adapters genuinely represents a specific morphism in $\overline{\mathscr{C}}$ built from associators and unitors; we can use this fact in to restate the coherence theorem in terms of adapter morphisms. Mac Lane's corollary then follows straightforwardly: the basic idea is to "export" commuting diagrams from $\mathscr{W}$ to an arbitrary monoidal category by interpreting the objects of $\mathscr{W}$ as functors, and arrows as natural transformations. We provide a graphical exposition of this proof in the extended version of our paper [23].

5 Conclusions

The body of work on string diagrams in general is broad and growing rapidly. It is therefore slightly surprising that the fundamental issue of non-strict tensorial composition has been neglected for so long. On the one hand, this is reasonable. The assumption of strictness does not entail a loss of generality, as indeed we have confirmed via an example in Sec. 2.3. However, non-strict tupling is a basic feature of programming languages, and even hardware description languages, and modelling it using string diagrams requires the proper mathematical framework.

This framework, the main contribution of the paper given in Def. 1, shows a way to strictify a possibly non-strict monoidal category. The body of the paper proves that the definition has all the desired properties and, in the process, we discuss two new proofs for Mac Lane's strictness and coherence theorems, respectively. We believe that, as is usually the case, the string-diagrammatic perspective has pedagogical value, lending concrete intuitions to what otherwise seems like an abstract symbolic exercise.

5.1 Further work

Lack of support for non-strict tensor limits the range of many applications of string diagrams. The first immediate question to study is whether the strictification recipe we give is compatible with hierarchical string diagrams ("functorial boxes" [19]) which can be used in the representation of monoidal-closed and cartesian-closed categories. This, in turn, makes them useful for applications to programming languages with higher-order functions, such as high-level circuit synthesis [11] or automatic differentiation [1], which currently do not offer support for product. Similar considerations motivate the study of strictification of trace monoidal categories, which can be used as models of digital circuits [12].

Further, our construction expands the use of datastructures and algorithms currently limited only to the strict case (e.g., [24]). Such datastructures are typically based on graph or hypergraph representations for performance reasons; applying our construction allows us to leverage those benefits essentially for free. In cases where such datastructures and algorithms are proven correct, it may be beneficial to reproduce the proofs in this paper in a formal theorem prover in order to provide end-to-end verification of applications.

Finally, a formal understanding of non-strict monoidal categories may open the door to more graphical approaches for theorem proving. Interactive graphical theorem provers using string diagrams for strict monoidal categories such as `homotopy.io` represent a refreshingly new approach to the design of proof assistants. Since models of, for example, intuitionistic logic are non-strict, the novel string diagrams in this paper could be used perhaps to develop similar graphical proof assistant for more conventional logics.

References

1 Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger, and Fabio Zanasi. Functorial string diagrams for reverse-mode automatic differentiation. *CoRR*, abs/2107.13433, 2021. `arXiv:2107.13433`.

2 John C. Baez and Brendan Fong. A compositional framework for passive linear networks. *Theory and Applications of Categories*, 33(38):1158–1222, 2018. `doi:10.48550/arXiv.1504.05625`.

3 R.F. Blute, J.R.B. Cockett, R.A.G. Seely, and T.H. Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996. `doi:10.1016/0022-4049(95)00159-X`.

4 Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Graphical affine algebra. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, June 2019. `doi:10.1109/lics.2019.8785877`.

5 Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. Full abstraction for signal flow graphs. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 515–526. ACM, 2015. `doi:10.1145/2676726.2676993`.

6 Titouan Carette, Yohann D'Anello, and Simon Perdrix. Quantum algorithms and oracles with the scalable ZX-calculus. *Electronic Proceedings in Theoretical Computer Science*, 343:193–209, September 2021. `doi:10.4204/eptcs.343.10`.

7 Vikraman Choudhury, Jacek Karwowski, and Amr Sabry. Symmetries in reversible programming: From symmetric rig groupoids to reversible programming languages, 2021. `doi:10.48550/arXiv.2110.05404`.

8 Bob Coecke, Edward Grefenstette, and Mehrnoosh Sadrzadeh. Lambek vs. lambek: Functorial vector space semantics and string diagrams for lambek calculus. *Ann. Pure Appl. Log.*, 164(11):1079–1100, 2013. `doi:10.1016/j.apal.2013.05.009`.

9 Bob Coecke and Aleks Kissinger. *Picturing quantum processes: A first course in quantum theory and diagrammatic reasoning*. Cambridge University Press, 2017.

10 P. I. Etingof, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik, editors. *Tensor categories*. Number volume 205 in Mathematical surveys and monographs. American Mathematical Society, 2015. URL: `https://klein.mit.edu/~etingof/egnobookfinal.pdf`.

11 Dan R. Ghica. Geometry of synthesis: a structured approach to VLSI design. In Martin Hofmann and Matthias Felleisen, editors, *Proceedings of the 34th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2007, Nice, France, January 17-19, 2007*, pages 363–375. ACM, 2007. `doi:10.1145/1190216.1190269`.

12 Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic semantics for digital circuits. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. `doi:10.4230/LIPIcs.CSL.2017.24`.

13 Dan R. Ghica, George Kaye, and David Sprunger. Full abstraction for digital circuits, 2022. `doi:10.48550/arXiv.2201.10456`.

14 Peter Hines. Coherence and strictification for self-similarity, 2015. `arXiv:1304.5954`.

15 A. Joyal and R. Street. Braided tensor categories. *Advances in Mathematics*, 102(1):20–78, 1993. `doi:10.1006/aima.1993.1055`.

16 André Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

17 Yves Lafont. Towards an algebraic theory of Boolean circuits. *Journal of Pure and Applied Algebra*, 184(2-3):257–310, November 2003. doi:10.1016/S0022-4049(03)00069-0.

18 Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1997. doi:10.1007/978-1-4757-4721-8.

19 Paul-André Melliès. Functorial boxes in string diagrams. In Zoltán Ésik, editor, *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, volume 4207 of *Lecture Notes in Computer Science*, pages 1–30. Springer, 2006. doi:10.1007/11874683_1.

20 Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. arXiv:0908.3347.

21 Michael Shulman. A practical type theory for symmetric monoidal categories, 2019. doi:10.48550/arXiv.1911.00818.

22 Stuart Sutherland, Simon Davidmann, and Peter Flake. *SystemVerilog for Design Second Edition: A Guide to Using SystemVerilog for Hardware Design and Modeling*. Springer Science & Business Media, 2006.

23 Paul Wilson, Dan Ghica, and Fabio Zanasi. String diagrams for non-strict monoidal categories, 2022. doi:10.48550/arXiv.2201.11738.

24 Paul Wilson and Fabio Zanasi. The cost of compositionality: A high-performance implementation of string diagram composition, 2021. arXiv:2105.09257.

25 Paul Wilson and Fabio Zanasi. Categories of differentiable polynomial circuits for machine learning, 2022. doi:10.48550/arXiv.2203.06430.

A Sequential Normal Form

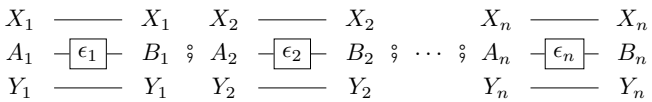
The following proposition is well-known (see for example [17]) and straightforward to prove, but we provide a proof anyway for completeness.

▶ **Proposition 36.** *let $\mathscr{C}$ be a monoidal category presented by generators Σ and some equations. Then any (finite) term t representing a morphism of $\mathscr{C}$ can be factored into "slices":*

$$(\mathsf{id} \otimes \epsilon_1 \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes \epsilon_2 \otimes \mathsf{id}) \fatsemi \ldots \fatsemi (\mathsf{id} \otimes \epsilon_n \otimes \mathsf{id})$$

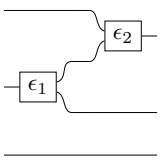
where each ϵ_i is a generator.

This factorization can be diagrammed as follows:



Note that in general $X_i \neq X_{i+1}$ and so on- i.e., the generators need not be "aligned" in this factorization. For example, we can have morphisms like the following:

▶ **Example 37.**



Proof. We proceed by induction on terms. Let S_0 denote the set of terms consisting of identities and generators, Then let

$$S_n = S_0 \cup \{t \fatsemi u | t, u \in S_{n-1}\} \cup \{t \otimes u | t, u \in S_{n-1}\}$$

It is clear that terms in S_0 are already in sequential normal form, so it remains to prove the inductive case, beginning with composition. Let v be a term in S_{n+1}. Now by inductive hypothesis, any term in $w \in S_n$ has an equivalent term in sequential normal form, which we'll denote $\hat{w}$. Now there are three cases:

1. If $v \in S_n$, then we have $\hat{v}$ by inductive hypothesis.
2. If $v = t \fatsemi u$, then $\hat{t}$ and $\hat{u}$ exist by inductive hypothesis, and we can form $\hat{v} = \hat{t} \fatsemi \hat{u}$.
3. If $v = t \otimes u$, then $\hat{v} = (\hat{t} \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes \hat{u})$

and the proof is complete. ◀

B $\mathcal{N}$ is well-defined

In this appendix we verify that $\mathcal{N}$ is well-defined. This amounts to two things: first that $\mathcal{N}$ is well-defined with respect to the interchange law, and second that it respects the equations of Definition 1.

In the former case, sequential normal forms are only unique up to interchange, so it must be verified that $\mathcal{N}$ preserves this property. This is a straightforward if tedious exercise, which can be done by verifying each of the cases in Definition 6. Essentially, the only axioms required are naturality and equations [10, 2.12, 2.13].

Finally, we need to verify the equations of 1. Specifically, for each of the monoidal, adapter, and associator/unitor equations $\texttt{lhs} = \texttt{rhs}$, we show that $\mathcal{N}(\texttt{lhs}) = \mathcal{N}(\texttt{rhs})$. We give derivations for these below. Using these derivations one can also tediously check that the equations hold for cases $\mathsf{id} \bullet \texttt{lhs} \bullet \mathsf{id} = \mathsf{id} \bullet \texttt{rhs} \bullet \mathsf{id}$, and so $\mathcal{N}$ is equal under any rewrite involving those equations; the only cases of interest are for associator and unitor equations, which require the use of the pentagon and triangle axioms, respectively.

We begin with the functor equations (2)

$$\mathcal{N}(\overline{\mathsf{id}_A}) = \mathsf{id}_A = \mathcal{N}(\mathsf{id}_{\overline{A}}) \qquad \mathcal{N}(\overline{f} \fatsemi \overline{g}) = \mathcal{N}(\overline{f}) \fatsemi \mathcal{N}(\overline{g}) = f \fatsemi g = \mathcal{N}(\overline{f \fatsemi g})$$

Now the adapter equations (3):

$$\begin{aligned}
\mathcal{N}(\Phi \fatsemi (\overline{f} \bullet \overline{g}) \fatsemi \Phi^*) &= \mathcal{N}(\Phi) \fatsemi \mathcal{N}(\overline{f} \bullet \overline{g}) \fatsemi \mathcal{N}(\Phi^*) \\
&= \mathcal{N}(\overline{f} \bullet \overline{g}) \\
&= \mathcal{N}(\overline{f} \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \overline{g}) \\
&= (f \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g) \\
&= f \otimes g \\
&= \mathcal{N}(\overline{f \otimes g})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\phi \fatsemi \phi^*) &= \mathcal{N}(\phi) \fatsemi \mathcal{N}(\phi^*) \\
&= \mathsf{id}_{I_{\mathscr{C}}} \fatsemi \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathcal{N}(\mathsf{id}_{I_{\overline{\mathscr{C}}}})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\Phi^* \fatsemi \overline{f \otimes g} \fatsemi \Phi^*) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\overline{f \otimes g}) \fatsemi \mathcal{N}(\Phi) \\
&= \mathcal{N}(\overline{f \otimes g}) \\
&= f \otimes g \\
&= (f \otimes \mathsf{id}) \fatsemi (\mathsf{id} \otimes g) \\
&= \mathcal{N}(\overline{f} \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \overline{g}) \\
&= \mathcal{N}((\overline{f} \bullet \mathsf{id}) \fatsemi (\mathsf{id} \bullet \overline{g})) \\
&= \mathcal{N}(\overline{f} \bullet \overline{g})
\end{aligned}$$

$$\begin{aligned}
\mathcal{N}(\phi^* \fatsemi \phi) &= \mathcal{N}(\phi^*) \fatsemi \mathcal{N}(\phi) \\
&= \mathsf{id}_{I_{\mathscr{C}}} \fatsemi \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathsf{id}_{I_{\mathscr{C}}} \\
&= \mathcal{N}(\overline{\mathsf{id}_{I_{\mathscr{C}}}}) \\
&= \mathcal{N}(\mathsf{id}_{\overline{I_{\mathscr{C}}}})
\end{aligned}$$

Finally the associator/unitor equations (4):

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\mathsf{id} \bullet \Phi^*) \fatsemi (\Phi \bullet \mathsf{id}) \fatsemi \Phi) \\ &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\mathsf{id} \bullet \Phi^*) \fatsemi \mathcal{N}(\Phi \bullet \mathsf{id}) \fatsemi \mathcal{N}(\Phi) \\ &= \mathsf{id} \fatsemi \mathsf{id} \fatsemi \alpha \fatsemi \mathsf{id} \\ &= \alpha \\ &= \mathcal{N}(\overline{\alpha})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\Phi^* \bullet \mathsf{id}) \fatsemi (\mathsf{id} \bullet \Phi) \fatsemi \Phi) \\ &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\Phi^* \bullet \mathsf{id}) \fatsemi \mathcal{N}(\mathsf{id} \bullet \Phi) \fatsemi \mathcal{N}(\Phi) \\ &= \mathsf{id} \fatsemi \alpha^{-1} \fatsemi \mathsf{id} \fatsemi \mathsf{id} \\ &= \alpha^{-1} \\ &= \mathcal{N}(\overline{\alpha^{-1}})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\phi^* \bullet \mathsf{id})) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\phi^* \bullet \mathsf{id}) \\ &= \mathsf{id} \fatsemi \lambda \\ &= \lambda \\ &= \mathcal{N}(\overline{\lambda})\end{aligned}$$

$$\begin{aligned}\mathcal{N}((\phi \bullet \mathsf{id}) \fatsemi \Phi) &= \mathcal{N}(\phi \bullet \mathsf{id}) \fatsemi \mathcal{N}(\Phi) \\ &= \lambda^{-1} \fatsemi \mathsf{id} \\ &= \lambda^{-1} \\ &= \mathcal{N}(\overline{\lambda^{-1}})\end{aligned}$$

$$\begin{aligned}\mathcal{N}(\Phi^* \fatsemi (\mathsf{id} \bullet \phi^*)) &= \mathcal{N}(\Phi^*) \fatsemi \mathcal{N}(\mathsf{id} \bullet \phi^*) \\ &= \mathsf{id} \fatsemi \rho \\ &= \rho \\ &= \mathcal{N}(\overline{\rho})\end{aligned}$$

$$\begin{aligned}\mathcal{N}((\mathsf{id} \fatsemi \phi) \fatsemi \Phi) &= \mathcal{N}(\mathsf{id} \fatsemi \phi) \fatsemi \mathcal{N}(\Phi) \\ &= \rho^{-1} \fatsemi \mathsf{id} \\ &= \rho^{-1} \\ &= \mathcal{N}(\overline{\rho^{-1}})\end{aligned}$$

Thus $\mathcal{N}$ is well-defined with respect to the monoidal equations.