Logical Methods in Computer Science
Volume 21, Issue 2, 2025, pp. 9:1–9:20
https://lmcs.episciences.org/

Submitted Apr. 10, 2024
Published Apr. 29, 2025

STRING DIAGRAMS FOR PREMONOIDAL CATEGORIES

MARIO ROMÁN [a] AND PAWEŁ SOBOCIŃSKI [b]

[a] University of Oxford and Tallinn University of Technology
e-mail address: mario.roman.garcia@cs.ox.ac.uk

[b] Tallinn University of Technology
e-mail address: pawel.sobocinski@taltech.ee

Abstract. Premonoidal categories are monoidal categories without the interchange law while effectful categories are premonoidal categories with a chosen monoidal subcategory of interchanging morphisms. In the same sense that string diagrams—pioneered by Joyal and Street—are an internal language for monoidal categories, we show that string diagrams with an added "runtime object"—pioneered by Alan Jeffrey—are an internal language for effectful categories and can be used as string diagrams for effectful, premonoidal and Freyd categories.

1. Introduction

Graphical programming languages have been of perennial interest to programmers: they are intuitive even for end-users without software development training, and they are closer to the digrammatic descriptions found across multiple branches of engineering [KFHB21]. While classical programming semantics employs categorical structures such as *monads* and *premonoidal categories* [Mog91, PR97], graphical programming semantics is traditionally a semi-formal field. This manuscript defends that graphical programming languages can be given semantics with the same formal techniques as their textual counterparts.

Indeed, category theory has fostered two successful applications that are rarely combined: monoidal string diagrams [JS91] and functional programming semantics [Mog91]. String diagrams are used to describe about quantum transformations [AC09], relational queries [BSS18], and even computability [Pav13]; at the same time, proof nets and the geometry of interaction [Gir89, BCST96] have been widely applied in computer science [AHS02, HMH14]. On the other hand, monads and comonads, Kleisli categories and premonoidal categories

Mario Román and Paweł Sobociński were supported by the Estonian Research Council grant PRG1210 and by the Advanced Research + Invention Agency (ARIA) Safeguarded AI Programme. Paweł Sobociński was additionally supported by European Union and Estonian Research Council via project TEM-TA5, by the Estonian Research Council funded Estonian Centre of Excellence in Artificial Intelligence project TK213U9 and by the European Union under Grant Agreement No. 101087529. Mario Román was additionally supported by the Air Force Office of Scientific Research (AFOSR) award number FA9550-21-1-0038. This manuscript is an extended version of "Promonads and String Diagrams for Effectful Categories" by the first-named author, presented at Applied Category Theory 2022.

DOI:10.46298/LMCS-21(2:9)2025

are used to explain effectful functional programming [Hug00, JHH09, Mog91, PT99, UV08]. Although Freyd categories with a cartesian base [Pow02] are commonly used, it is also possible to consider non-cartesian Freyd categories [SL13], which are called *effectful categories*.

These applications are well-known. However, some foundational results in the intersection between string diagrams, premonoidal categories and effectful categories are missing in the literature. This manuscript contributes one such result: *we introduce string diagrams for premonoidal and effectful categories*. Jeffrey [Jef97b] was the first to employ string diagrams of premonoidal categories, and we are indebted to his and Schweimeier's expositions [Sch01]. Jeffrey's technique consisted in introducing an extra wire—which we call the *runtime*—that prevents some morphisms from interchanging.

We promote this technique into a result about the construction of free premonoidal and free effectful categories: the free premonoidal category can be constructed in terms of the free monoidal category with an additional wire. For this construction, we rely on Joyal and Street's analogue result on string diagrams for monoidal categories [JS91]. Our slogan, which constitutes the statement of Theorem 3.14, says that

Premonoidal categories are monoidal categories with a runtime.

1.1. **Synopsis.** Sections 2.1 and 2.2 contain mostly preliminary material on premonoidal, Freyd and effectful categories. Section 3 recalls string diagrams for monoidal categories and constructs string diagrams for effectful categories. Section 5 describes string diagrams for premonoidal categories.

1.2. **Contributions.** Our main original contribution is in Section 3; we prove that effectful categories are monoidal categories with runtime via an adjunction (Theorem 3.14). We immediately obtain string diagrams for effectful categories (Corollary 3.15). Our second contribution is an adjunction constructing string diagrams for premonoidal categories (Theorem 5.5).

2. Premonoidal and Effectful Categories

2.1. **Premonoidal categories.** Premonoidal categories are monoidal categories without the *interchange law*, $(f \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g) \neq (\mathrm{id} \otimes g) \fatsemi (f \otimes \mathrm{id})$. This means that we cannot tensor any two arbitrary morphisms, $(f \otimes g)$, without explicitly stating which one is to be composed first, $(f \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g)$ or $(\mathrm{id} \otimes g) \fatsemi (f \otimes \mathrm{id})$, and the two compositions are not equivalent (Figure 1).

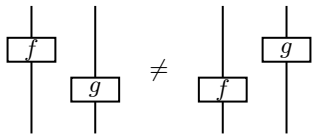


Figure 1. The interchange law does not hold in a premonoidal category.

In technical terms, the tensor of a premonoidal category $(\otimes)\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is not a functor, but only what is called a *sesquifunctor*: independently functorial on each variable. Indeed, tensoring with any identity, which is sometimes referred to as *whiskering*, is a functor $(\bullet \otimes \mathrm{id})\colon \mathbb{C} \to \mathbb{C}$, but there is no functor $(\bullet \otimes \bullet)\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$.

A good motivation for dropping the interchange law can be found when describing transformations that affect some global state. These effectful processes should not interchange in general, because the order in which we modify the global state is meaningful. For instance, in the Kleisli category of the *writer monad*, $(\Sigma^* \times \bullet)\colon \mathbf{Set} \to \mathbf{Set}$ for some alphabet $\Sigma \in \mathbf{Set}$, we can consider the function $\mathsf{print}\colon \Sigma^* \to \Sigma^* \times 1$, which is an effectful map from Σ^* to 1. The order in which we "print" does matter (Figure 2).

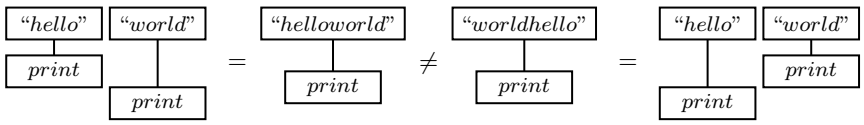


FIGURE 2. Writing does not interchange.

Not surprisingly, the paradigmatic examples of premonoidal categories are the Kleisli categories of Set-based monads $T\colon \mathbf{Set} \to \mathbf{Set}$ (more generally, of strong monads), which fail to be monoidal unless the monad itself is commutative [Gui80, PR97, PT99]. Intuitively, the morphisms are "effectful", and these effects do not always commute.

However, we may still want to allow some morphisms to interchange. For instance, apart from asking the same associators and unitors of monoidal categories to exist, we ask them to be *central*: that means that they interchange with any other morphism. This notion of centrality forces us to write the definition of premonoidal category in two different steps: first, we introduce the minimal setting in which centrality can be considered (*binoidal* categories [PT99]) and then we use that setting to bootstrap the full definition of premonoidal category with central coherence morphisms.

Definition 2.1 (Binoidal category)**.** A *binoidal category* is a category $\mathbb{C}$ endowed with an object $I \in \mathbb{C}$ and an object $A \otimes B$ for each $A \in \mathbb{C}$ and $B \in \mathbb{C}$. There are whiskering functors $(A \otimes \bullet)\colon \mathbb{C} \to \mathbb{C}$, and $(\bullet \otimes B)\colon \mathbb{C} \to \mathbb{C}$ that coincide on $(A \otimes B)$, even if $(\bullet \otimes \bullet)$ is not necessarily itself a functor.

Again, this means that we can tensor with identities (whiskering), functorially; but we cannot tensor two arbitrary morphisms: the interchange law stops being true in general.

Definition 2.2 (Centre, central morphism)**.** The *centre* of a binoidal category, $\mathcal{Z}(\mathbb{C})$, is the wide subcategory of morphisms that do satisfy the interchange law with any other morphism. That is, $f\colon A \to B$ is *central* if, for each $g\colon A' \to B'$,

$$(f \otimes \mathrm{id}_{A'}) \fatsemi (\mathrm{id}_B \otimes g) = (\mathrm{id}_A \otimes g) \fatsemi (f \otimes \mathrm{id}_{B'}), \text{ and}$$
$$(\mathrm{id}_{A'} \otimes f) \fatsemi (g \otimes \mathrm{id}_B) = (g \otimes \mathrm{id}_A) \fatsemi (\mathrm{id}_{B'} \otimes f).$$

Definition 2.3 (Premonoidal category)**.** A *premonoidal category* is a binoidal category $(\mathbb{C}, \otimes, I)$ together with the following coherence isomorphisms $\alpha_{A,B,C}\colon A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, $\rho_A\colon A \otimes I \to A$ and $\lambda_A\colon I \otimes A \to A$ which are central, natural *separately at each given component*, and satisfy the pentagon and triangle equations.

A premonoidal category is *strict* when these coherence morphisms are identities. A premonoidal category is moreover *symmetric* when it is endowed with a coherence isomorphism $\sigma_{A,B}\colon A \otimes B \to B \otimes A$ that is central and natural at each given component, and satisfies the symmetry condition and hexagon equations.

Remark 2.4. The coherence theorem of monoidal categories still holds for premonoidal categories: every premonoidal is equivalent to a strict one. We will construct the free

strict premonoidal category using string diagrams. However, the usual string diagrams for monoidal categories need to be restricted: in premonoidal categories, we cannot consider two morphisms in parallel unless one of the two is *central*.

2.2. **Effectful and Freyd categories.** Premonoidal categories immediately present a problem: what are the strong premonoidal functors? If we want them to compose, they should preserve centrality of the coherence morphisms (so that the central coherence morphisms of $F \fatsemi G$ are these of F after applying G), but naively asking them to preserve all central morphisms would rule out otherwise valid functors[1] [SL13]. The solution is to explicitly choose some central morphisms that represent "pure" computations. These do not need to form the whole centre: it could be that some morphisms considered *effectful* just "happen" to fall in the centre of the category, while we do not ask our functors to preserve them. This is the well-studied notion of a *non-cartesian Freyd category*, which we shorten to *effectful monoidal category* or *effectful category*.

Thus, effectful categories are premonoidal categories endowed with a chosen family of central morphisms. These central morphisms are called pure morphisms, constrasting with the general, non-central, morphisms that fall outside this family, which we call effectful.

Definition 2.5. An *effectful category* is an identity-on-objects functor $\mathbb{V} \to \mathbb{C}$ from a monoidal category $\mathbb{V}$ (the pure morphisms, or "values") to a premonoidal category $\mathbb{C}$ (the effectful morphisms, or "computations"), that strictly preserves all of the premonoidal structure and whose image is central. It is *strict* when both are. A *Freyd category* [Lev22] is an effectful category where the pure morphisms form a cartesian monoidal category.

Remark 2.6 (Terminology)**.** The name "Freyd category" sometimes assumes cartesianity of the pure morphisms, but it is sometimes also used for the general case. To avoid this clash of nomenclature we reserve "effectful categories" for the general case and "Freyd categories" for the cartesian case. There exists also the more fine-grained notion of "Cartesian effect category" [DDR11], which generalizes Freyd categories.

Effectful categories solve the problem of defining premonoidal functors: a functor between effectful categories only needs to preserve the pure morphisms. We are not losing expressivity: premonoidal categories are effectful with their own centre, $\mathcal{Z}(\mathbb{C}) \to \mathbb{C}$. From now on, we focus on effectful categories.

Definition 2.7 (Effectful functor)**.** Let $\mathbb{V} \to \mathbb{C}$ and $\mathbb{W} \to \mathbb{D}$ be effectful categories. An *effectful functor* is a quadruple $(F, F_0, \varepsilon, \mu)$ consisting of a functor $F\colon \mathbb{C} \to \mathbb{D}$ and a functor $F_0\colon \mathbb{V} \to \mathbb{W}$ making the square commute, and two natural and pure isomorphisms $\varepsilon\colon J \cong F(I)$ and $\mu\colon F(A \otimes B) \cong F(A) \otimes F(B)$ such that they make F_0 a monoidal functor. It is *strict* if these are identities. Strict effectful categories and functors form a category, $\mathbf{EffCat}_{\mathsf{Str}}$.

When drawing string diagrams in an effectful category, we could use two different colours to declare if we are depicting either a value or a computation (Figure 3).

[1] For instance, the inclusion of a commutative monoid into a non-commutative monoid does not necessarily preserve the monoid centre; in turn, this induces an inclusion between the Kleisli categories of their writer monads that does not necessarily preserve the premonoidal centre.

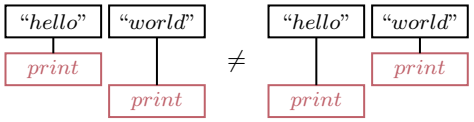


FIGURE 3. "Hello world" is not "world hello".

Here, the values "hello" and "world" satisfy the interchange law as in an ordinary monoidal category. However, the effectful computation "print" does not need to satisfy the interchange law. String diagrams like these can be found in the work of Alan Jeffrey [Jef97b]. Jeffrey presents a clever mechanism to graphically depict the failure of interchange: all effectful morphisms need to have a control wire as an input and output. This control wire needs to be passed around to all the computations in order, and it prevents them from interchanging.

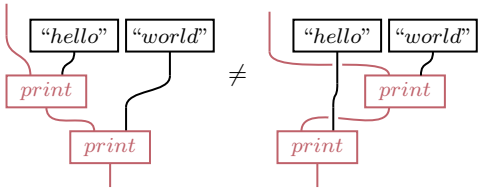


FIGURE 4. An additional wire prevents interchange.

A common interpretation of monoidal categories is as theories of resources. We can interpret premonoidal categories as monoidal categories with an extra resource—the "runtime"—that needs to be passed to all computations. The next section promotes Jeffrey's observation into a theorem.

Remark 2.8 (Non-interchanging string diagrams)**.** Could we avoid the extra runtime wire and simply keep track of which morphisms must not be interchanged? This is an easy solution and we employ it in Figure 3; unfortunately, it raises an important concern: we lose a "locality of substitutions" property that, intuitively, we expect string diagrams to satisfy.

Explicitly, imagine that we know that a pure morphism arises as the composition of two effectful morphisms, $f = g \fatsemi h$. We expect to be able to substitute f by the pair of morphisms $g \fatsemi h$ at whichever point in a string diagram without regarding the rest of the diagram, as we do with monoidal string diagrams. However, say that f is tensored with an effectful morphism k; in this case, the relative order of the morphisms, g, h and k, is meaningful and forces us to be careful about the position of the rest of the nodes on the diagram (see Figure 5).

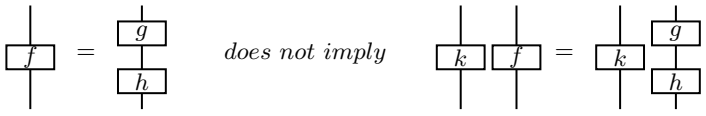


FIGURE 5. Substitution with non-interchanging string diagrams.

Considering runtime as an extra wire addresses this problem in two ways: *(i)* it becomes obvious that pure and effectful morphisms are topologically different, and *(ii)* it shows how the second substitution is not local and should not follow from the assumption (see Figure 6).

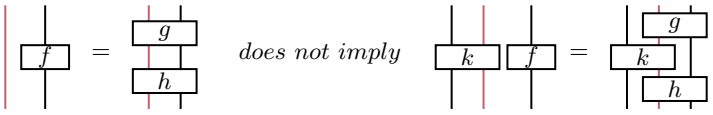


Figure 6. Substitution with runtime string diagrams.

3. String Diagrams for Effectful Categories

3.1. **String Diagrams for Monoidal Categories.** Soundness and completeness of string diagrams can be proven by showing that the morphisms of the monoidal category freely generated over a polygraph of generators are string diagrams on these generators, quotiented by topological deformations [JS91]. We will justify string diagrams for premonoidal categories by proving that the freely generated effectful category over a pair of polygraphs (for pure and effectful generators, respectively) can be constructed as the freely generated monoidal category over a particular polygraph that includes an extra wire. Let us start by recalling string diagrams for monoidal categories, following the work of Joyal and Street.

Definition 3.1. A *polygraph* $\mathcal{G}$ (analogue of a *multigraph* [Shu16]) is given by a set of objects, $\mathcal{G}_{\text{obj}}$, and a set of generators $\mathcal{G}(A_0, \ldots, A_n; B_0, \ldots, B_m)$ for any two sequences of objects $A_0, \ldots, A_n$ and $B_0, \ldots, B_m$. A morphism of polygraphs $f \colon \mathcal{G} \to \mathcal{H}$ is a function between their object sets, $f_{\text{obj}} \colon \mathcal{G}_{\text{obj}} \to \mathcal{H}_{\text{obj}}$, and a function between their corresponding morphism sets,

$$f_{A_0,\ldots,A_n;B_0,\ldots,B_n} \colon \mathcal{G}(A_0, \ldots, A_n; B_0, \ldots, B_m) \to \mathcal{H}(f_{\text{obj}}(A_0), \ldots, f_{\text{obj}}(A_n); f_{\text{obj}}(B_0), \ldots, f_{\text{obj}}(B_m)).$$

Polygraphs and polygraph morphisms form a category, **PolyGr**.

There exists an adjunction between polygraphs and strict monoidal categories. Any monoidal category $\mathbb{C}$ can be seen as a polygraph, $\mathsf{Forget}(\mathbb{C})$; the objects are those of the monoidal category; the edges, $\mathsf{Forget}(\mathbb{C})(A_0, \ldots, A_n; B_0, \ldots, B_m)$, are the morphisms $\mathbb{C}(A_0 \otimes \ldots \otimes A_n, B_0 \otimes \ldots \otimes B_m)$; we forget about composition and tensoring. Given a polygraph $\mathcal{G}$, the free strict monoidal category $\mathsf{String}(\mathcal{G})$ is the strict monoidal category that has as morphisms the string diagrams over the generators in the polygraph.

Definition 3.2. A *string diagram* over a *polygraph* $\mathcal{G}$ (or *progressive plane graph* in the work of Joyal and Street [JS91, Definition 1.1]) is a graph Γ embedded in the squared interval, up to planar isotopy, and such that

(1) the boundary of the graph touches only the top and the bottom of the square, $\delta\Gamma \subseteq \{0, 1\} \times [0, 1]$;
(2) and the second projection is injective on each component of the graph without its vertices, $\Gamma - \Gamma_0$; this makes it acyclic and progressive.

We call the components of $\Gamma - \Gamma_0$ *wires*, W; we call the vertices of the graph *nodes*, Γ_0. Wires must be labeled by the objects of the polygraph, $o \colon W \to \mathcal{G}_{obj}$, nodes must be labeled by the generators of the polygraph, $m \colon \Gamma_0 \to \mathcal{G}$; and each node must be connected to wires exactly typed by the objects of its generator—a string diagram must be well-typed.

Remark 3.3 (The symmetric case)**.** Compare these with the string diagrams for symmetric monoidal categories [Sel10]. String diagrams for symmetric monoidal categories have a practical combinatorial characterization in terms of acyclic hypergraphs [BGK+22]. This

characterization would simplify our results, but we prefer to work for the full generality of (non-symmetric) monoidal categories.

Lemma 3.4. *String diagrams over a polygraph $\mathcal{G}$ form a monoidal category, which we call* $\mathsf{String}(\mathcal{G})$*. This determines a functor,*

$$\mathsf{String}\colon \mathbf{PolyGr} \to \mathbf{MonCat}_{\mathsf{Str}}.$$

Proof sketch. The objects of the category are lists of objects of the polygraph, which we write as $[X_0, \dots, X_n]$, for $X_i \in \mathcal{G}_{obj}$. These form a (free) monoid with concatenation and the empty list.

Morphisms $[X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ are string diagrams over the polygraph $\mathcal{G}$ such that *(i)* the ordered list of wires that touches the upper boundary is typed by $[X_0, \dots, X_n]$, and *(ii)* the ordered list of wires that touches the lower boundary is typed by $[Y_0, \dots, Y_m]$.

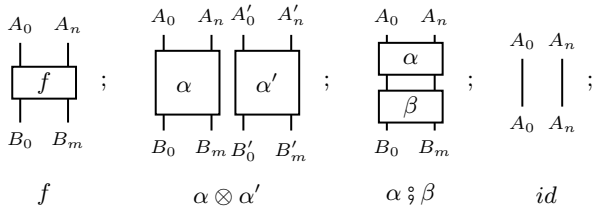


FIGURE 7. Strict monoidal category of string diagrams.

Figure 7 describes the operations of the category. The parallel composition of two diagrams $\alpha\colon [X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ and $\alpha'\colon [X'_0, \dots, X'_{n'}] \to [Y'_0, \dots, Y'_{m'}]$ is their horizontal juxtaposition. The sequential composition of two diagrams $\alpha\colon [X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ and $\beta\colon [Y_0, \dots, Y_m] \to [Z_0, \dots, Z_k]$ is the diagram obtained by vertical juxtaposition linking the outputs of the first to the inputs of the second. The identity on the object $[X_0, \dots, X_n]$ is given by a diagram containing n identity wires labeled by these objects. □

Lemma 3.5. *Forgetting about the sequential and parallel composition defines a functor from monoidal categories to polygraphs,*

$$\mathsf{Forget}\colon \mathbf{MonCat}_{\mathsf{Str}} \to \mathbf{PolyGr}.$$

Proof. Any monoidal category $\mathbb{C}$ can be seen as a polygraph, $\mathsf{Forget}(\mathbb{C})$, where the edges are determined by the morphisms,

$$\mathsf{Forget}(\mathbb{C})(A_0, \dots, A_n; B_0, \dots, B_m) = \mathbb{C}(A_0 \otimes \dots \otimes A_n, B_0 \otimes \dots \otimes B_m),$$

and we forget about composition and tensoring. It can be checked, by its definition, that any strict monoidal functor induces a homomorphism on the underlying polygraphs. □

Theorem 3.6 (Joyal and Street, [JS91, Theorem 2.3])**.** *There exists an adjunction between polygraphs and strict monoidal categories,* $\mathsf{String} \dashv \mathsf{Forget}$*. Given a polygraph $\mathcal{G}$, the free strict monoidal category,* $\mathsf{String}(\mathcal{G})$*, is the strict monoidal category that has as morphisms the string diagrams over the generators of the polygraph; the underlying polygraph determines the right adjoint.*

3.2. **String Diagrams for Effectful Categories.** String diagrams *for monoidal categories* arise from the adjunction between polygraphs and monoidal categories. Let us translate this story to the effectful case: we will construct a similar adjunction between effectful polygraphs and effectful categories. We start by formally adding the runtime to a free monoidal category.

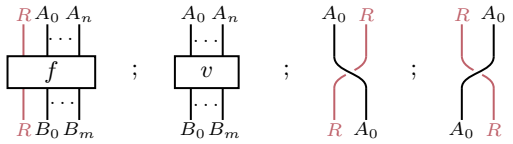
Definition 3.7 (Effectful polygraph)**.** An *effectful polygraph* is a pair of polygraphs $(\mathcal{V}, \mathcal{G})$ sharing the same objects, $\mathcal{V}_{\text{obj}} = \mathcal{G}_{\text{obj}}$. A morphism of effectful polygraphs $(u, f) \colon (\mathcal{V}, \mathcal{G}) \to (\mathcal{W}, \mathcal{H})$ is a pair of morphisms of polygraphs, $u \colon \mathcal{V} \to \mathcal{W}$ and $f \colon \mathcal{G} \to \mathcal{H}$, such that they coincide on objects, $f_{\text{obj}} = u_{\text{obj}}$. Effectful polygraphs form a category, **EffPolyGr**.

Definition 3.8 (Runtime monoidal category)**.** Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph. Its *runtime monoidal category*, $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$, is the monoidal category freely generated from adding an extra object—the runtime, R—to the input and output of every effectful generator in $\mathcal{G}$ (but not to those in $\mathcal{V}$), and letting that extra object be braided with respect to every other object of the category.

In other words, it is the monoidal category freely generated by the following polygraph, $\text{Run}(\mathcal{V}, \mathcal{G})$, (Figure 8), assuming $A_0, \dots, A_n$ and $B_0, \dots, B_m$ are distinct from R,

- $\text{Run}(\mathcal{V}, \mathcal{G})_{\text{obj}} = \mathcal{G}_{\text{obj}} + \{R\} = \mathcal{V}_{\text{obj}} + \{R\}$,
- $\text{Run}(\mathcal{V}, \mathcal{G})(R, A_0, \dots, A_n; R, B_0, \dots, B_n) = \mathcal{G}(A_0, \dots, A_n; B_0, \dots, B_n)$,
- $\text{Run}(\mathcal{V}, \mathcal{G})(A_0, \dots, A_n; B_0, \dots, B_n) = \mathcal{V}(A_0, \dots, A_n; B_0, \dots, B_n)$,
- $\text{Run}(\mathcal{V}, \mathcal{G})(R, A_0; A_0, R) = \text{Run}(\mathcal{V}, \mathcal{G})(A_0, R; R, A_0) = \{\sigma\}$,

with $\text{Run}(\mathcal{V}, \mathcal{G})$ empty in any other case, and quotiented by the braiding axioms for R (Figure 9).



For each $f \in \mathcal{G}(A_0, \dots, A_n; B_0, \dots, B_m)$ and each $v \in \mathcal{V}(A_0, \dots, A_n; B_0, \dots, B_m)$.

FIGURE 8. Generators for the runtime monoidal category.

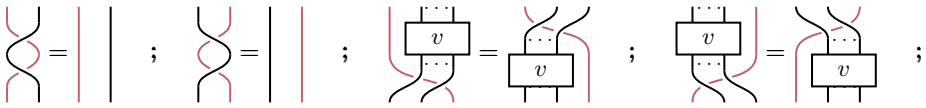


FIGURE 9. Axioms for the runtime monoidal category.

Some readers may prefer to understand this as asking the runtime R to be in the Drinfeld centre [DGNO10] of the monoidal category.[2] The extra wire that R provides is only used to prevent interchange, and so it does not really matter where it is placed in the input and the output. We can choose to always place it on the left, for instance—and indeed we will be able to do so—but a better solution is to just consider objects "up to some runtime braidings". This is formalized by the notion of *braid clique*.

[2]The Drinfeld centre of a monoidal category $(\mathbb{C}, \otimes, I)$ is braided monoidal category that has as objects the pairs (R, σ), where $\sigma \colon R \otimes \bullet \to \bullet \otimes R$ is a natural isomorphism satisfying the braiding axioms; morphisms in the Drinfeld centre are those that preserve these braidings [DGNO10].

Definition 3.9 (Braid clique)**.** Given any list of objects $A_0, \ldots, A_n$ in $\mathcal{V}_{\text{obj}} = \mathcal{G}_{\text{obj}}$, we construct a *clique* [Tod10, Shu18] in the category $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$: we consider the objects, $A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n$, created by inserting the runtime R in all of the possible $0 \leqslant i \leqslant n+1$ positions; and we consider the family of commuting isomorphisms constructed by braiding the runtime,

$$\sigma_{i,j} \colon A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n \to A_0 \otimes \ldots \otimes R_{(j)} \otimes \ldots \otimes A_n.$$

We call this family of isomorphisms the *braid clique*, $\mathrm{Braid}_R(A_0, \ldots, A_n)$, on that list.

Definition 3.10. A *braid clique morphism*,

$$f \colon \mathrm{Braid}_R(A_0, \ldots, A_n) \to \mathrm{Braid}_R(B_0, \ldots, B_m),$$

is a family of morphisms in the runtime monoidal category, $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$, from each of the objects of first clique to each of the objects of the second clique,

$$f_{ik} \colon A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R_{(k)} \otimes \ldots \otimes B_m,$$

that moreover commutes with all braiding isomorphisms, $f_{ij} \fatsemi \sigma_{jk} = \sigma_{il} \fatsemi f_{lk}$.

A braid clique morphism $f \colon \mathrm{Braid}_R(A_0, \ldots, A_n) \to \mathrm{Braid}_R(B_0, \ldots, B_m)$ is fully determined by *any* of its components: any component can be obtained uniquely from any other, simply by pre/post-composing it with braidings. In particular, a braid clique morphism is always fully determined by its leftmost component,

$$f_{00} \colon R \otimes A_0 \otimes \ldots \otimes A_n \to R \otimes B_0 \otimes \ldots \otimes B_m.$$

For the following two lemmas, we choose to always deal with the leftmost component of the braid clique morphism. Given any clique $\mathrm{Braid}_R(A_0, \ldots, A_n)$ we let $A = A_0 \otimes \ldots \otimes A_n$; so clique morphisms $\mathrm{Braid}_R(A_0, \ldots, A_n) \to \mathrm{Braid}_R(B_0, \ldots, B_m)$ are represented by morphisms $R \otimes A \to R \otimes B$.

We now use braid morphisms to construct a premonoidal category. Braid cliques are the objects of a category; however, it is interesting to note they it is *not* a monoidal category. Indeed, the tensor of morphisms cannot be defined in the naive way: a morphism $R \otimes A \to R \otimes B$ cannot be tensored with a morphism $R \otimes A' \to R \otimes B'$ to obtain a morphism $R \otimes A \otimes A' \to R \otimes B \otimes B'$ (as the runtime would appear twice); they only form a premonoidal category.

Lemma 3.11. *Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph. There exists a premonoidal category, $\mathsf{EffString}(\mathcal{V}, \mathcal{G})$, that has as objects the braid cliques, $\mathrm{Braid}_R(A_0, \ldots, A_n)$, in $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$, and as morphisms the braid clique morphisms between them.*

Proof. Let us first give $\mathsf{EffString}(\mathcal{V}, \mathcal{G})$ category structure. The identity on $\mathrm{Braid}_R(A_0, \ldots, A_n)$ is the identity on $R \otimes A$. The composition of a morphism $R \otimes A \to R \otimes B$ with a morphism $R \otimes B \to R \otimes C$ is their plain composition as string diagrams, in $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$.

Let us now check that it is moreover a premonoidal category. Tensoring of braid cliques is given by concatenation of lists, which coincides with the tensor of objects in $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})$. Whiskering of a morphism $f \colon R \otimes A \to R \otimes B$ is defined with braidings in the left case, $R \otimes C \otimes A \to R \otimes C \otimes B$, and by plain whiskering in the right case, $R \otimes A \otimes C \to R \otimes B \otimes C$, as depicted in Figure 10.

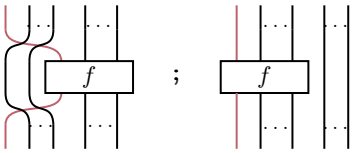


Figure 10. Whiskering in the runtime premonoidal category.

Finally, the associators and unitors are identities, which are natural and central. □

Lemma 3.12. *Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph. There exists an identity-on-objects functor $\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G})$ that strictly preserves the premonoidal structure and whose image is central. This determines an effectful category.*

Proof. A morphism $v \in \mathsf{String}(\mathcal{V})(A, B)$ induces a morphism $(\mathrm{id}_R \otimes v) \in \mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(R \otimes A, R \otimes B)$, which can be read as a morphism of cliques $(\mathrm{id}_R \otimes v) \in \mathsf{EffString}(\mathcal{V}, \mathcal{G})(A, B)$. This is tensoring with an identity, which is indeed functorial.

Let us now show that this functor strictly preserves the premonoidal structure. The fact that it preserves right whiskerings is immediate. The fact that it preserves left whiskerings follows from the axioms of symmetry (Figure 11, left). Associators and unitors are identities, which are preserved by tensoring with an identity.

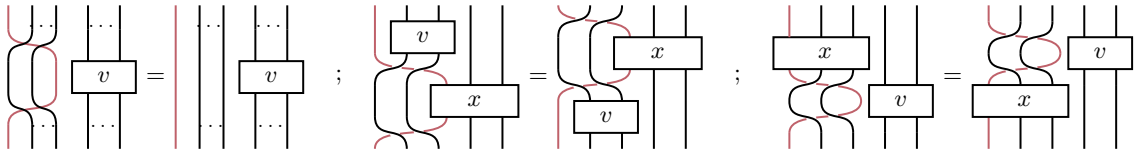


Figure 11. Preservation of whiskerings, and centrality.

Finally, we can check by string diagrams that the image of this functor is central, interchanging with any given $x \colon R \otimes C \to R \otimes D$ (Figure 11, center and right). □

Lemma 3.13. *Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph and consider the effectful category determined by $\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G})$. Let $\mathbb{V} \to \mathbb{C}$ be a strict effectful category endowed with an effectful polygraph morphism $F \colon (\mathcal{V}, \mathcal{G}) \to \mathcal{U}(\mathbb{V}, \mathbb{C})$. There exists a unique strict effectful functor from $(\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G}))$ to $(\mathbb{V} \to \mathbb{C})$ commuting with F as an effectful polygraph morphism.*

Proof. By the same freeness result for monoidal categories, there already exists a unique strict monoidal functor $H_0 \colon \mathsf{String}(\mathcal{V}) \to \mathbb{V}$ that sends any object $A \in \mathcal{V}_{\mathrm{obj}}$ to $F_{obj}(A)$.

We will show there is a unique way to extend this functor together with the hypergraph assignment $\mathcal{G} \to \mathbb{C}$ into a functor $H \colon \mathsf{EffString}(\mathcal{V}, \mathcal{G}) \to \mathbb{C}$. Even when we know that the runtime can appear always on the left, we choose here to define the functor independently of the position of the runtime and later check that it preserves braidings. That is, giving such a functor amounts to give some mapping of morphisms containing the runtime R in some position in their input and output,

$$f \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$$

to morphisms $H(f) \colon FA_0 \otimes \ldots \otimes FA_n \to FB_0 \otimes \ldots \otimes FB_n$ in $\mathbb{C}$, in a way that preserves composition, whiskerings, inclusions from $\mathsf{String}(\mathcal{V})$, and that is invariant to composition with braidings. In order to define this mapping, we will perform structural induction over the monoidal terms of the runtime monoidal category of the form $\mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(A_0 \otimes \ldots \otimes$

$R^{(i)} \otimes \ldots \otimes A_n, R \otimes B_0 \otimes \ldots \otimes R^{(j)} \otimes \ldots \otimes B_m$) and show that it is the only mapping with these properties (Figure 12).

Monoidal terms in a strict, freely presented, monoidal category are formed by identities (id), composition ($\fatsemi$), tensoring ($\otimes$), and some generators (in this case, in Figure 8). Here is where we apply the main result of Joyal and Street [JS91]: reasoning about monoidal terms is the same as reasoning about string diagrams; thus, we can prove a result about string diagrams reasoning inductively on monoidal terms. Monoidal terms are subject to *(i)* functoriality of the tensor, $\mathrm{id} \otimes \mathrm{id} = \mathrm{id}$ and $(f \fatsemi g) \otimes (h \fatsemi k) = (f \otimes h) \fatsemi (g \otimes k)$; *(ii)* associativity and unitality of the tensor, $f \otimes \mathrm{id}_I = f$ and $f \otimes (g \otimes h) = (f \otimes g) \otimes h$; *(iii)* the usual unitality, $f \fatsemi \mathrm{id} = f$ and $\mathrm{id} \fatsemi f = f$ and associativity $f \fatsemi (g \fatsemi h) = (f \fatsemi g) \fatsemi h$; *(iv)* the axioms of our presentation (in this case, in Figure 9).

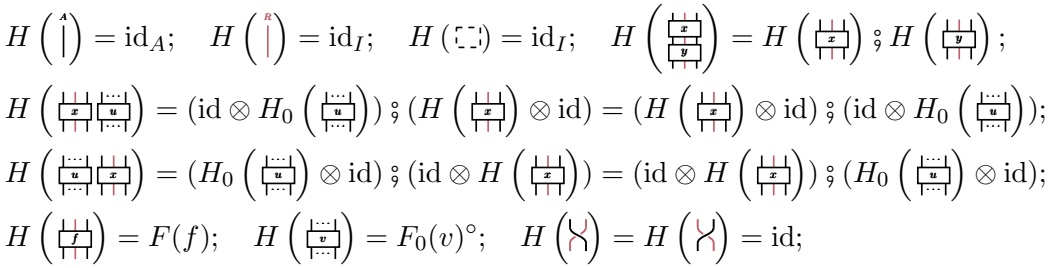


FIGURE 12. Assignment on morphisms, defined by structural induction on terms.

- If the term is an identity, it can be *(i)* an identity on an object of the effectful polygraph, $A \in (\mathcal{V}, \mathcal{G})_{\mathrm{obj}}$, in which case it must be mapped to the same identity by functoriality, $H(\mathrm{id}_A) = \mathrm{id}_A$; *(ii)* an identity on the runtime, in which case it must be mapped to the identity on the unit object, $H(\mathrm{id}_R) = \mathrm{id}_I$; or *(iii)* an identity on the unit object, in which case it must be mapped to the identity on the unit, $H(\mathrm{id}_I) = \mathrm{id}_I$.
- If the term is a composition, $(f \fatsemi g) \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to C_0 \otimes \ldots \otimes R \otimes \ldots \otimes C_k$, it must be along a boundary of the form $B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$: this is because every generator leaves the number of runtimes, R, invariant. Thus, each one of the components determines itself a braid clique morphism. We must preserve composition of braid clique morphisms, so we must map $H(f \fatsemi g) = H(f) \fatsemi H(g)$.
- If the term is a tensor of two terms, $(x \otimes u) \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$, then only one of them was a term taking R as input and output (without loss of generality, assume it to be the first one) and the other was not: again, by construction, there are no morphisms taking one R as input and producing none, or viceversa. We split this morphism into $x \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_{i-1} \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_{j-1}$ and $u \colon A_i \otimes \ldots \otimes A_n \to B_j \otimes \ldots \otimes B_m$.

  Again by structural induction, this time over terms $u \colon A_i \otimes \ldots \otimes A_n \to B_j \otimes \ldots \otimes B_m$, we know that the morphism must be either a generator in $\mathcal{V}(A_i, \ldots, A_n; B_j, \ldots, B_n)$ or a composition and tensoring of them. That is, u is a morphism in the image of $\mathsf{String}(\mathcal{V})$, and it must be mapped according to the functor $H_0 \colon \mathsf{String}(\mathcal{V}) \to \mathbb{V}$.

  By induction hypothesis, we know how to map the morphism $x \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_{i-1} \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_{j-1}$. This means that, given any tensoring $x \otimes u$, we must map it to $H(x \otimes u) = (H(x) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes H_0(u)) = (\mathrm{id} \otimes H_0(u)) \fatsemi (H(x) \otimes \mathrm{id})$, where $H_0(u)$ is central.

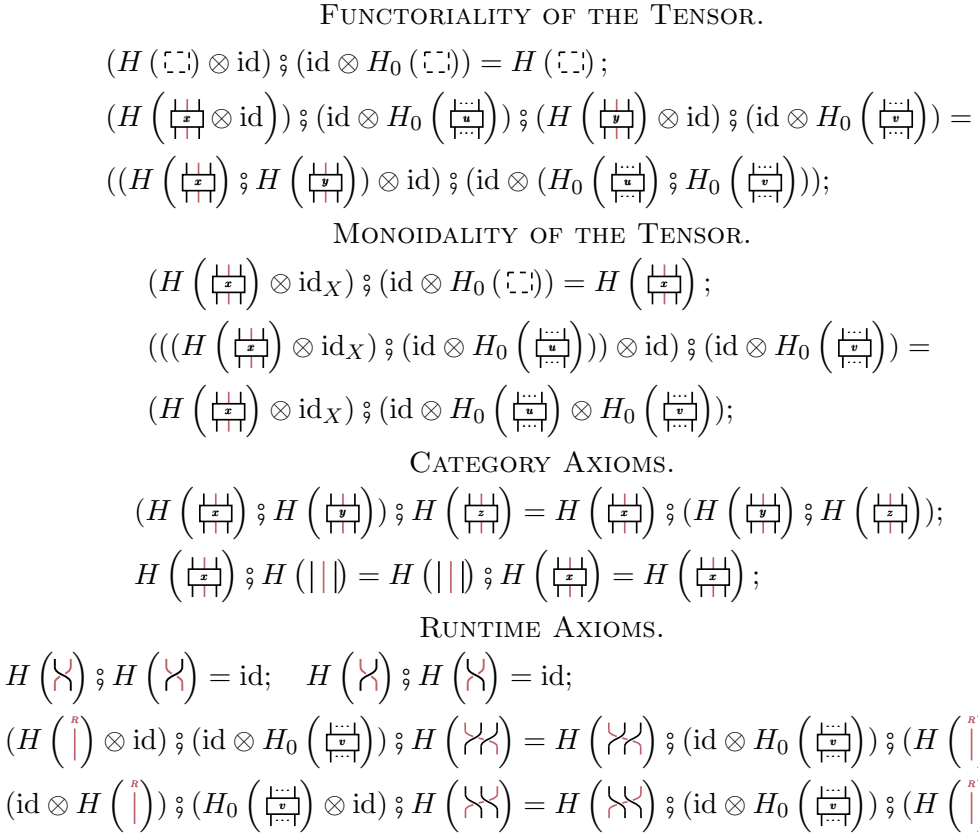


Figure 13. The assignment is well defined.

- If the string diagram consists of a single generator, $f\colon R \otimes A \to R \otimes B$, it can only come from a generator

  $$f \in \mathrm{Run}(\mathcal{V}, \mathcal{G})(R, A_0, \ldots, A_n; R, B_0, \ldots, B_m) = \mathcal{G}(A_0, \ldots, A_n; B_0, \ldots, B_m),$$

  which must be mapped to $H(f) = F(f) \in \mathbb{C}(A_0 \otimes \ldots \otimes A_n, B_0 \otimes \ldots \otimes B_m)$. If the string diagram consists of a single braiding, it must be mapped to the identity, because the want the assignment to be invariant to braidings.

Now, we need to prove that this assignment is well-defined with respect to the axioms of these monoidal terms. Our reasoning follows Figure 13.

- The tensor is functorial. We know that $H(\mathrm{id} \otimes \mathrm{id}) = H(\mathrm{id})$, both are identities and that can be formally proven by induction on the number of wires. Now, for the interchange law, consider a quartet of morphisms that can be composed or tensored first and such that, without loss of generality, we assume the runtime to be on the left side. Then, we can use centrality to argue that

  $$\begin{aligned} H((x \otimes u) \fatsemi (y \otimes v)) &= (H(x) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes H_0(u)) \fatsemi (H(y) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes H_0(v)) \\ &= ((H(x) \fatsemi H(y)) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes (H_0(u) \fatsemi H_0(v))) \\ &= H((x \fatsemi y) \otimes (u \fatsemi v)). \end{aligned}$$

- The tensor is monoidal. We know that $H(x \otimes \mathrm{id}_I) = (H(x) \otimes \mathrm{id}_I) \fatsemi (\mathrm{id} \otimes \mathrm{id}_I) = H(x)$. Now, for associativity, consider a triple of morphisms that can be tensored in two ways

and such that, without loss of generality, we assume the runtime to be on the left side. Then, we can use centrality to argue that

$$\begin{aligned} H((x \otimes u) \otimes v) &= (((H(x) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes H_0(u))) \otimes \mathrm{id}) \fatsemi \mathrm{id} \otimes H_0(v) \\ &= (H(x) \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes H_0(u) \otimes H_0(v)) \\ &= H(x \otimes (u \otimes v)). \end{aligned}$$

- The terms form a category. And indeed, it is true by construction that $H(x \fatsemi (y \fatsemi z)) = H((x \fatsemi y) \fatsemi z)$ and also that $H(x \fatsemi \mathrm{id}) = H(x)$, because H preserves composition.
- The runtime category enforces some axioms. The composition of two braidings is mapped to the identity by the fact that H preserves composition and sends both to the identity. Both sides of the braid naturality over a morphism v are mapped to $H_0(v)$; with the multiple braidings being mapped again to the identity.

Thus, H is well-defined and it defines the only possible assignment and the only possible strict premonoidal functor. □

Theorem 3.14 (Runtime as a resource)**.** *The free strict effectful category over an effectful polygraph* $(\mathcal{V}, \mathcal{G})$ *is* $\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G})$*. Its morphisms* $A \to B$ *are in bijection with the morphisms* $R \otimes A \to R \otimes B$ *of the runtime monoidal category,*

$$\mathsf{EffString}(\mathcal{V}, \mathcal{G})(A, B) \cong \mathsf{String}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(R \otimes A, R \otimes B).$$

Proof. We must first show that $\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G})$ is an effectful category. The first step is to see that $\mathsf{EffString}(\mathcal{V}, \mathcal{G})$ forms a premonoidal category; we apply Lemma 3.11. We also know that $\mathsf{String}(\mathcal{V})$ is a monoidal category: in fact, a strict, freely generated one. There exists an identity on objects functor, $\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G})$, that strictly preserves the premonoidal structure and centrality, by Lemma 3.12.

Let us now show that it is the free one over the effectful polygraph $(\mathcal{V}, \mathcal{G})$. Let $\mathbb{V} \to \mathbb{C}$ be an effectful category, with an effectful polygraph map $F \colon (\mathcal{V}, \mathcal{G}) \to \mathcal{U}(\mathbb{V}, \mathbb{C})$. We can construct a unique effectful functor from $(\mathsf{String}(\mathcal{V}) \to \mathsf{EffString}(\mathcal{V}, \mathcal{G}))$ to $(\mathbb{V} \to \mathbb{C})$ giving its universal property; this is Lemma 3.13, which concludes the proof. □

Corollary 3.15 (String diagrams for effectful categories)**.** *We can use string diagrams for effectful categories, quotiented under the same isotopy as for monoidal categories, provided that we do represent the runtime as an extra wire that needs to be the input and output of every effectful morphism.*

4. Example: Global State

Let us provide an example of reasoning that uses string diagrams for effectful categories. Imperative programs are characterized by the presence of a global state that can be mutated. Reading or writing to this global state constitutes an effectful computation: the order of operations that affect some global state cannot be changed. Let us propose a simple theory of global state (Figure 14) and let us show that it is enough to capture the phenomenon of *race conditions*. These are an adaptation of the *lens laws* [FGM+07].

Definition 4.1. The theory of *global state* is generated by the effectful polygraph with a single object X; two pure generators, $(\mathsf{copy}) \colon X \to X \otimes X$ and $(\mathsf{discard}) \colon X \to I$, representing copy and discard, quotiented by the comonoid axioms; and two effectful generators, $(\mathsf{get}) \colon I \to X$

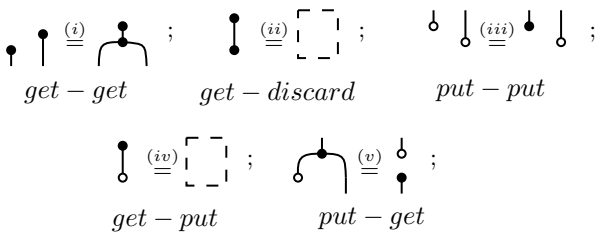


FIGURE 14. Non-interchanging diagrams for the axioms of global state.

and $(\text{ϕ}) \colon X \to I$, representing get and put (in Figure 15), quotiented by the equations in Figure 16.

FIGURE 15. Generators for global state.

These two put and get generators, without extra axioms, represent what happens when a process can *send* or *receive* resources from an unrestricted environment. Right now, we are concerned with the theory of a global state accessed by a single process: we will impose the following axioms, which assert that the global state was not modified by anyone but this single process (Figure 16).

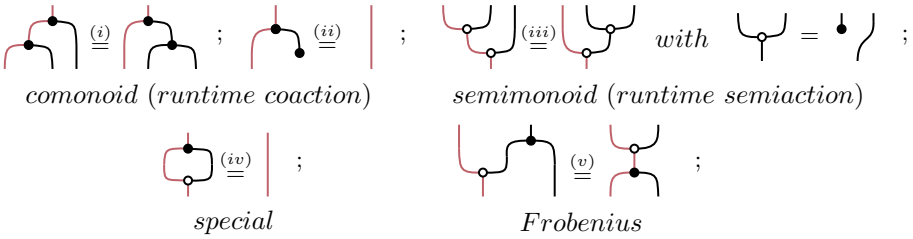


FIGURE 16. Effectful string diagrams for the axioms of global state.

The equations in Figure 16 (and Figure 14) say that: (i) reading the global state twice gets us the same result twice, (ii) reading the global state and discarding the result is the same as doing nothing, (iii) writing twice to the global state keeps only the last thing that was written, (iv) reading something and immediately writing it to the global state is the same as doing nothing, (v) keeping a copy of something that we write to global state is the same as writing it to the global state and then reading it.

Remark 4.2 (Global state is semi-Frobenius)**.** The theory in Figure 16 can be summarized as a *semi-Frobenius action* on the runtime R, where the semimonoid is the *left-absorbing semigroup*[3]. The *get-get* and *get-discard* laws correspond to a comonoid action on the runtime; the *put-put* law correspond to a semimonoid action of a left-absorbing semigroup; the *get-put* law correspond to the special axiom; and the *put-get* law corresponds to the Frobenius axiom.

[3]Left-absorbing semigroups are those satisfying the equation $x \cdot y = x$ [KKM11]; we use the name *semimonoid* and interpret them more generally in a symmetric monoidal category.

While this algebra exists also in Figure 14, it only becomes apparent when we use the string diagrams of effectful categories, in Figure 16.

Proposition 4.3 (Race conditions)**.** *Concurrently mixing two processes that share a global state, f and g, can produce four possible results: (i) only the result of the first one is preserved, (ii) only the result of the second one is preserved, or (iii,iv) the composition of both is preserved, in any order.*

Proof. Formally, we work in the theory of global state adding two processes, $f\colon X \to X$ and $g\colon X \to X$, that can moreover be discarded, meaning $f \fatsemi \bullet = g \fatsemi \bullet = \bullet$. We employ string diagrams for effectful categories. The following three diagrams in Figure 17 correspond to the first three cases of race conditions; the last one is analogous to the third. □

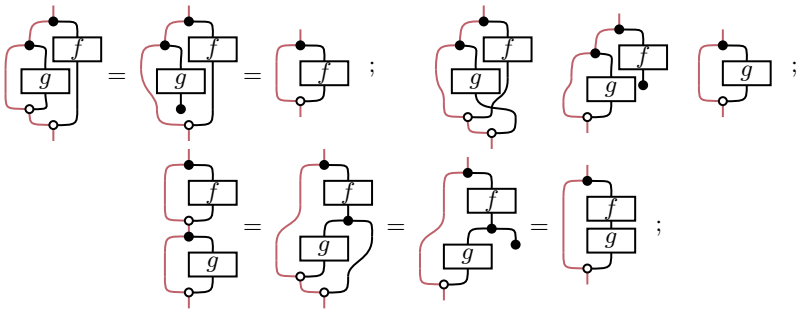


FIGURE 17. Possible results of a race condition.

5. String Diagrams for Premonoidal Categories

We have constructed string diagrams for effectful categories via an adjunction (Theorem 3.14); and as a consequence, we will also have constructed string diagrams for premonoidal categories. At the beginning of this text, we left open the question of what were the correct functors between premonoidal categories: we now choose to assemble them into a category where functors preserve the central morphisms. This is not as satisfactory as working directly with effectful categories, but we do so for completeness; we refer the reader to the work of Staton and Levy for a detailed discussion of why effectful categories may be a better notion than premonoidal categories [SL13].

Definition 5.1. Strict premonoidal categories and central functors between them form a category, $\mathbf{Premon}_{\mathsf{Str}}$, endowed with a fully faithful functor $\mathsf{J}\colon \mathbf{Premon}_{\mathsf{Str}} \to \mathbf{EffCat}_{\mathsf{Str}}$ that acts by sending a premonoidal category $\mathbb{C}$ to the effectful category given by the inclusion of its center, $\mathcal{Z}(\mathbb{C}) \to \mathbb{C}$.

On the syntax side, we will use the fact that every polygraph—every signature of a premonoidal category—can be read as an effectful polygraph with no pure generators.

Proposition 5.2. *There exists an adjunction between polygraphs and effectful polygraphs. The left adjoint is given by the functor interpreting every edge as an effectful one,* $\mathsf{I}\colon \mathbf{PolyGr} \to \mathbf{EffPolyGr}$*. The right adjoint functor forgets the pure edges,* $\mathsf{V}\colon \mathbf{EffPolyGr} \to \mathbf{PolyGr}$*.*

On the semantics side, we may observe that the free effectful category over a polygraph constructed by the previous runtime construction, $\mathsf{R}\colon \mathbf{EffPolyGr} \to \mathbf{EffCat}_{\mathsf{Str}}$, is actually a premonoidal category that has a trivial centre consisting only of identities.

Lemma 5.3. *The functor* $\mathsf{I} \fatsemi \mathsf{R} \colon \mathbf{PolyGr} \to \mathbf{EffCat}_{\mathsf{Str}}$ *factors through the inclusion of premonoidal categories as* $\mathsf{I} \fatsemi \mathsf{R}^* \fatsemi \mathsf{J}$ *for a codomain restriction* $\mathsf{R}^* \colon \mathbf{PolyGr} \to \mathbf{Premon}_{\mathsf{Str}}$.

Corollary 5.4. *Finally, from the previous section, we extract that there is an adjunction between effectful polygraphs and effectful categories with effectful functors given by the runtime construction of the free effectful category* $\mathsf{R} \colon \mathbf{EffPolyGr} \to \mathbf{EffCat}_{\mathsf{Str}}$ *and the forgetful functor* $\mathsf{U} \colon \mathbf{EffCat}_{\mathsf{Str}} \to \mathbf{EffPolyGr}$.

Theorem 5.5. *There is an adjunction between premonoidal categories and polygraphs given by the string diagrams of effectful categories with every node needing the runtime.*

Proof. We show the adjunction by a natural bijection of morphism sets. Let $\mathcal{G}$ be any polygraph and let $\mathbb{P}$ be any premonoidal category. The right adjoint will be $\mathsf{U}_\bullet = \mathsf{V} \fatsemi \mathsf{U} \fatsemi \mathsf{J}$; the left adjoint will be $\mathsf{I} \fatsemi \mathsf{R}^*$, as obtained in Lemma 5.3.

$$\mathbf{PolyGr}(\mathcal{G}; \mathsf{U}_\bullet \mathbb{P}) \cong \mathbf{PolyGr}(\mathcal{G}; \mathsf{VUJ}\mathbb{P}) \overset{(i)}{\cong} \mathbf{EffPolyGr}(\mathsf{I}\mathcal{G}; \mathsf{UJ}\mathbb{P}) \overset{(ii)}{\cong} \mathbf{EffCat}_{\mathsf{Str}}(\mathsf{RI}\mathcal{G}; \mathsf{J}\mathbb{P})$$
$$\overset{(iii)}{\cong} \mathbf{EffCat}_{\mathsf{Str}}(\mathsf{JR}^*\mathsf{I}\mathcal{G}; \mathsf{J}\mathbb{P}) \overset{(iv)}{\cong} \mathbf{Premon}_{\mathsf{Str}}(\mathsf{R}^*\mathsf{I}\mathcal{G}; \mathbb{P}).$$

Here, we have used that *(i)* the adjunction between polygraphs and effectful polygraphs by Lemma 5.2; *(ii)* the adjunction between effectful polygraphs and effectful categories in Corollary 5.4; *(iii)* the factorization in Lemma 5.3; and *(iv)* that premonoidal categories embed fully-faithfully into effectful categories. □

6. Conclusions

Premonoidal categories are monoidal categories with runtime, and we can still use monoidal string diagrams and unrestricted topological deformations to reason about them. Instead of dealing directly with premonoidal categories, we employ the better-behaved notion of non-cartesian Freyd categories, effectful categories. There exists a more fine-grained notion of "Cartesian effect category" [DDR11], which generalizes Freyd categories and justifies using "effectful category" for the general case.

Ultimately, this is a first step towards our more ambitious project of presenting the categorical structure of programming languages in a purely diagrammatic way, revisiting Alan Jeffrey's work [Jef97b, Jef97a, Rom22a]. The internal language of premonoidal categories and effectful categories is given by the *arrow do-notation* [Pat01]; at the same time, we have shown that it is given by suitable string diagrams. This correspondence allows us to translate between programs and string diagrams (Figure 18).

Example 6.1. This example illustrates a "Hello world", in diagrams and do-notation.

```
proc () -> do
  question <- "What's your name?"
  () <- print -< question
  name <- get -< ()
  greeting <- "Hello"
  output <- concatenate(greeting, name)
  () <- print -< output
return ()
```

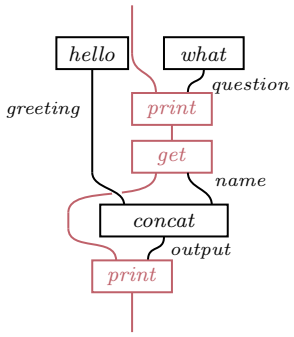


FIGURE 18. Premonoidal program in arrow do-notation and string diagrams.

7. Related and Further work.

This manuscript is an extended version of "Promonads and String Diagrams for Effectful Categories" by the first-named author, presented at Applied Category Theory 2022 [Rom22b]. It also extends and takes examples from the PhD thesis of the first-named author, which was supervised by the second-named author [Rom23]. The present version restructures the presentation for clarity, includes fully detailed proofs of all results, and adds a new section on premonoidal categories (Section 5).

Staton and Møgelberg [MS14] propose a formalization of Jeffrey's graphical calculus for effectful categories that arise as the Kleisli category of a strong monad. They prove that *'every strong monad is a linear-use state monad'*, that is, a state monad of the form $R \multimap !(\bullet) \otimes R$, where the state R is an object that cannot be copied nor discarded. Levy's Call-By-Push-Value [Lev22] is the closest translation of Freyd and effectful categories into a programming language that synthesizes the functional and imperative paradigms. Our work can be regarded as providing a string diagrammatic counterpart preserving effects but without higher-order functions.

Relatedly, the "Functional Machine Calculus" is a model of higher-order effectful computation developed in the work of Heijltjes, Barrett and McCusker [Hei22, BHM23, Bar23]. The functional machine calculus uses a string diagrammatic language reminiscent of the string diagrams for effectful categories that we introduce here; further work formalizing this connection is warranted. One of the aspects that separates other graphical languages like this one from the string diagrams for effectful categories is the occurrence of multiple independent runtimes, corresponding to multiple independent effects. Earnshaw and Nester [ENR23], in joint work with this author, explore how multiple runtimes may permit a more fine-grained presentation of effectful categories; this opens a future direction for generalizing this work.

Finally, Earnshaw and this second author [ES23] have developed a string diagrammatic trace theory for monoidal formal languages and monoidal automata that uses string diagrams of effectful categories.

Acknowledgements. The authors want to thank the anonymous reviewers at LMCS and the reviewers for "Promonads and String Diagrams for Effectful Categories" at Applied Category Theory 2022 [Rom22b]; especially their suggestion of separating and highlighting the first part of that work, which considerably improved the narrative. The authors want to thank Matt Earnshaw, Sam Staton, Tarmo Uustalu, and Niels Voorneveld for many helpful comments and discussions.

References

[AC09] Samson Abramsky and Bob Coecke. Categorical quantum mechanics. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of Quantum Logic and Quantum Structures*, pages 261–323. Elsevier, Amsterdam, 2009. URL: https://www.sciencedirect.com/science/article/pii/B9780444528698500104, doi:10.1016/B978-0-444-52869-8.50010-4.

[AHS02] Samson Abramsky, Esfandiar Haghverdi, and Philip J. Scott. Geometry of interaction and linear combinatory algebras. *Math. Struct. Comput. Sci.*, 12(5):625–665, 2002. doi:10.1017/S0960129502003730.

[Bar23] Chris Barrett. On the simply-typed functional machine calculus: Categorical semantics and strong normalisation. *CoRR*, abs/2305.16073, 2023. URL: https://doi.org/10.48550/arXiv.2305.16073, arXiv:2305.16073, doi:10.48550/ARXIV.2305.16073.

[BCST96] Richard F. Blute, J. Robin B. Cockett, Robert A.G. Seely, and Todd H. Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996. URL: https://www.sciencedirect.com/science/article/pii/002240499500159X, doi:10.1016/0022-4049(95)00159-X.

[BGK+22] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory II: rewriting with symmetric monoidal structure. *Math. Struct. Comput. Sci.*, 32(4):511–541, 2022. doi:10.1017/S0960129522000317.

[BHM23] Chris Barrett, Willem Heijltjes, and Guy McCusker. The functional machine calculus II: semantics. In Bartek Klin and Elaine Pimentel, editors, *31st EACSL Annual Conference on Computer Science Logic, CSL 2023, February 13-16, 2023, Warsaw, Poland*, volume 252 of *LIPIcs*, pages 10:1–10:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.CSL.2023.10, doi:10.4230/LIPICS.CSL.2023.10.

[BSS18] Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical conjunctive queries. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.13.

[DDR11] Jean-Guillaume Dumas, Dominique Duval, and Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. *Journal of Symbolic Computation*, 46(3):272–293, 2011. doi:10.1016/j.jsc.2010.09.008.

[DGNO10] Vladimir Drinfeld, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik. On braided fusion categories I. *Selecta Mathematica*, 16(1):1–119, 2010. doi:10.1007/s00029-010-0017-z.

[ENR23] Matt Earnshaw, Chad Nester, and Mario Román. Presentations of Premonoidal Categories by Devices, Extended Abstract. *Nordic Workshop on Programming Languages (NWPT'23). Online, https://mroman42.github.io/notes/papers/presentations-of-premonoidals-by-devices.pdf*, 2023.

[ES23] Matthew Earnshaw and Pawel Sobocinski. String diagrammatic trace theory. In Jérôme Leroux, Sylvain Lombardy, and David Peleg, editors, *48th International Symposium on Mathematical Foundations of Computer Science, MFCS 2023, August 28 to September 1, 2023, Bordeaux, France*, volume 272 of *LIPIcs*, pages 43:1–43:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.MFCS.2023.43, doi:10.4230/LIPICS.MFCS.2023.43.

[FGM+07] J Nathan Foster, Michael B Greenwald, Jonathan T Moore, Benjamin C Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17–es, 2007.

[Gir89] Jean-Yves Girard. Geometry of interaction 1: Interpretation of system f. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. Elsevier, 1989. URL: https://www.sciencedirect.com/science/article/pii/S0049237X08702714, doi:10.1016/S0049-237X(08)70271-4.

[Gui80] René Guitart. Tenseurs et machines. *Cahiers de topologie et géométrie différentielle catégoriques*, 21(1):5–62, 1980. URL: http://www.numdam.org/item/CTGDC_1980__21_1_5_0/.

[Hei22] Willem Heijltjes. The functional machine calculus. *CoRR*, abs/2212.08177, 2022. URL: https://doi.org/10.48550/arXiv.2212.08177, arXiv:2212.08177, doi:10.48550/ARXIV.2212.08177.

[HMH14] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014. doi:10.1145/2603088.2603124.

[Hug00] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. doi:10.1016/S0167-6423(99)00023-4.

[Jef97a] Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint at ResearchGate*, 1997. URL: https://www.researchgate.net/profile/Alan-Jeffrey/publication/228639836_Premonoidal_categories_and_a_graphical_view_of_programs/links/00b495182cd648a874000000/Premonoidal-categories-and-a-graphical-view-of-programs.pdf.

[Jef97b] Alan Jeffrey. Premonoidal categories and flow graphs. *Electron. Notes Theor. Comput. Sci.*, 10:51, 1997. doi:10.1016/S1571-0661(05)80688-7.

[JHH09] Bart Jacobs, Chris Heunen, and Ichiro Hasuo. Categorical semantics for arrows. *J. Funct. Program.*, 19(3-4):403–438, 2009. doi:10.1017/S0956796809007308.

[JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. URL: https://www.sciencedirect.com/science/article/pii/000187089190003P, doi:10.1016/0001-8708(91)90003-P.

[KFHB21] Mohammad Amin Kuhail, Shahbano Farooq, Rawad Hammad, and Mohammed Bahja. Characterizing visual programming approaches for end-user developers: A systematic review. *IEEE Access*, 9:14181–14202, 2021.

[KKM11] Mati Kilp, Ulrich Knauer, and Alexander V Mikhalev. *Monoids, Acts and Categories: With Applications to Wreath Products and Graphs. A Handbook for Students and Researchers*, volume 29. Walter de Gruyter, 2011.

[Lev22] Paul Blain Levy. Call-by-push-value. *ACM SIGLOG News*, 9(2):7–29, may 2022. doi:10.1145/3537668.3537670.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

[MS14] Rasmus Ejlers Møgelberg and Sam Staton. Linear usage of state. *Log. Methods Comput. Sci.*, 10(1), 2014. doi:10.2168/LMCS-10(1:17)2014.

[Pat01] Ross Paterson. A new notation for arrows. In Benjamin C. Pierce, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, pages 229–240. ACM, 2001. doi:10.1145/507635.507664.

[Pav13] Dusko Pavlovic. Monoidal computer I: basic computability by string diagrams. *Inf. Comput.*, 226:94–116, 2013. doi:10.1016/j.ic.2013.03.007.

[Pow02] John Power. Premonoidal categories as categories with algebraic structure. *Theor. Comput. Sci.*, 278(1-2):303–321, 2002. doi:10.1016/S0304-3975(00)00340-6.

[PR97] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Math. Struct. Comput. Sci.*, 7(5):453–468, 1997. doi:10.1017/S0960129597002375.

[PT99] John Power and Hayo Thielecke. Closed freyd- and kappa-categories. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 625–634. Springer, 1999. doi:10.1007/3-540-48523-6_59.

[Rom22a] Mario Román. Notes on Jeffrey's A Graphical View of Programs. Online, https://www.ioc.ee/~mroman/data/talks/premonoidalgraphicalview.pdf, March 2022.

[Rom22b] Mario Román. Promonads and string diagrams for effectful categories. In Jade Master and Martha Lewis, editors, *Proceedings Fifth International Conference on Applied Category Theory, ACT 2022, Glasgow, United Kingdom, 18-22 July 2022*, volume 380 of *EPTCS*, pages 344–361, 2022. doi:10.4204/EPTCS.380.20.

[Rom23] Mario Román. Monoidal Context Theory. PhD Thesis. *Tallinn University of Technology, available at https://mroman42.github.io/notes/papers/monoidal-context-theory.pdf*, 2023. Supervised by Paweł Sobociński.

[Sch01] Ralf Schweimeier. *Categorical and graphical models of programming languages*. PhD thesis, University of Sussex, UK, 2001. URL: https://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.366059.

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.

[Shu16] Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. URL: https://mikeshulman.github.io/catlog/catlog.pdf.

[Shu18] Michael Shulman. The 2-Chu-Dialectica construction and the polycategory of multivariable adjunctions. *arXiv preprint arXiv:1806.06082*, 2018. doi:10.48550/arxiv.1806.06082.

[SL13] Sam Staton and Paul Blain Levy. Universal properties of impure programming languages. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 179–192. ACM, 2013. doi:10.1145/2429069.2429091.

[Tod10] Todd Trimble. Coherence theorem for monoidal categories (nlab entry), section 3. discussion, 2010. https://ncatlab.org/nlab/show/coherence+theorem+for+monoidal+categories, Last accessed on 2022-05-10.

[UV08] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. In Jiří Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 263–284. Elsevier, 2008. doi:10.1016/j.entcs.2008.05.029.