Word problems on string diagrams

Antonin Delpeuch

New College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2021

Abstract

String diagrams are graphical representations of morphisms in various sorts of categories. The mathematical results which establish their soundness and completeness provide an elegant bridge between algebra and topology, equating a certain class of topological deformations to the equational theory of an algebraic structure. Beyond this conceptual appeal they also are practical tools to reason in the corresponding categories, making it easy to build intuition about the combinatorics of morphisms and efficiently communicate proofs to peers. Finally, they provide data structures to encode morphisms and manipulate them with computer programs such as proof assistants. Our contributions to the field explore three main directions.

First, we define a new class of string diagrams that we call *sheet diagrams* and prove their soundness and completeness for morphisms in free bimonoidal categories. This makes it possible to use string diagrams in situations involving two monoidal structures, one distributing over the other.

Second, the bulk of our work consists in studying how string diagram isotopy can be checked computationally. This is done by providing algorithms or hardness results for a particular class of decision problems, called *word problems*. Those word problems consist in determining whether two given diagrams can be related via a sequence of isotopy moves, whose nature depends on the algebraic structure at hand. We provide algorithms for the word problems for monoidal categories (or equivalently 2-categories or bicategories) and double categories. We also provide a hardness result for the word problem for braided monoidal categories, showing that this word problem is at least as hard as the unknotting problem, for which no polynomial-time solution is known to date.

Finally, we give a taste of how those techniques can be applied to real-world systems. We show how bimonoidal categories model the data transformation workflows offered by OpenRefine, an open source data wrangler focused on tabular data. This model has guided a refactoring of the architecture of the tool and suggests user interfaces to manipulate those workflows.

Word problems on string diagrams

Antonin Delpeuch
New College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Michaelmas 2021

Acknowledgements

It reads perhaps a bit bland, but it is true: my DPhil would likely not have been possible without the support of many people. This section is here to thank them, in no particular order. I want to thank Jamie for his tireless help and for believing in me even when I felt stuck in an impasse. I am also very grateful to my other supervisors Bob and Sam for helping me through the last steps of this journey.

I was lucky to be surrounded by eclectic, kind and open-minded fellow DPhil students, researchers and support staff in the Quantum Group. I thank them for their continuous help throughout these few years. I am immensely grateful to my friends from the Oxford University Ceilidh Band who welcomed me with open arms in their beautiful folky world. My friends from the British Human Power Club were also essential in keeping me insane throughout the degree. I thank my fellow activists from the Committee for the Accessibility of Publications in Sciences and Humanities, and all the signatories of the "No free view? No review!" pledge,[1] who helped me sustain a minimal viable faith in academia for the last few years. I am very grateful to my OpenRefine friends and colleagues for their patience while I write about abstract nonsense, and I look forward to working more with them now that this thesis has seen the light of day. I thank the courageous friends who went through earlier versions of this thesis and suggested many improvements.

I thank my friends of Kanthaus for their awesomness. And finally I thank my parents and siblings for their impatience to see this thesis in print.

Institutional

This DPhil was supported by a scholarship from the Engineering and Physical Sciences Reseach Council (EPSRC) and a Scatcherd European Scholarship from the University of Oxford.

[1] https://nofreeviewnoreview.org/

Abstract

String diagrams are graphical representations of morphisms in various sorts of categories. The mathematical results which establish their soundness and completeness provide an elegant bridge between algebra and topology, equating a certain class of topological deformations to the equational theory of an algebraic structure. Beyond this conceptual appeal they also are practical tools to reason in the corresponding categories, making it easy to build intuition about the combinatorics of morphisms and efficiently communicate proofs to peers. Finally, they provide data structures to encode morphisms and manipulate them with computer programs such as proof assistants. Our contributions to the field explore three main directions.

First, we define a new class of string diagrams that we call *sheet diagrams* and prove their soundness and completeness for morphisms in free bimonoidal categories. This makes it possible to use string diagrams in situations involving two monoidal structures, one distributing over the other.

Second, the bulk of our work consists in studying how string diagram isotopy can be checked computationally. This is done by providing algorithms or hardness results for a particular class of decision problems, called *word problems*. Those word problems consist in determining whether two given diagrams can be related via a sequence of isotopy moves, whose nature depends on the algebraic structure at hand. We provide algorithms for the word problems for monoidal categories (or equivalently 2-categories or bicategories) and double categories. We also provide a hardness result for the word problem for braided monoidal categories, showing that this word problem is at least as hard as the unknotting problem, for which no polynomial-time solution is known to date.

Finally, we give a taste of how those techniques can be applied to real-world systems. We show how bimonoidal categories model the data transformation workflows offered by OpenRefine, an open source data wrangler focused on tabular data. This model has guided a refactoring of the architecture of the tool and suggests user interfaces to manipulate those workflows.

Contents

1 Introduction

This chapter presents the scientific context and summarizes how our contributions articulate with it. We only aim to give a high-level overview. Readers who are unfamiliar with the concepts mentioned here can find precise definitions in Chapter 2.

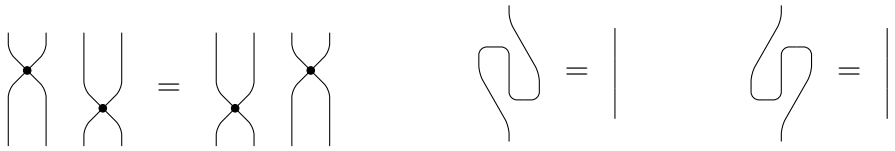
1.1 Graphical representations in mathematics

As pictured in popular culture, mathematics is all about writing arcane formulae and solving equations. Of course, in mathematical education, diagrams play an essential role in Euclidean geometry exercises and graphs are extensively used in real analysis, for instance. But many fields such as algebra or logic are perceived as mostly textual worlds.

This external perception of mathematics is fostered by some traditions in mathematical research communities. If graphical representations are to be used as part of a research project, they are too often treated as informal sketches that authors keep for themselves. Authors tend to communicate their proofs textually instead, the main goal being that a cautious reader can check the correctness of a proof almost mechanically, even if they lack the intuition behind it. This tendency is exacerbated by publishing constraints, as typesetting figures still is a time-consuming process. Diagrams also take away some precious space in articles which must often

fit in length constraints specified in pages. This reluctance for figures contributes to the image of mathematics as a dry, inaccessible and arrogant field.

String diagrams are particular sorts of diagrams used in category theory. Not only can they be used to build up intuition about a problem, but also act as a full-blown replacement for the traditional formula-based syntax to denote morphisms. One can therefore define structures and prove properties about them using string diagrams directly, without having to translate the expressions involved to formulae. Furthermore, some fundamental equations of widely used algebraic structures have a particularly compelling graphical counterpart. This is the case, for instance, for the exchange law for monoidal categories (Figure 1.1a) or the elimination of units and counits in an adjunction (Figure 1.1b). This unexpected outbreak of topology in an algebraic context is a sign that the said algebraic structures are natural and worth studying. It is also a great opportunity to challenge the textual habits of mathematical communication, by using diagrams as a primary communication medium and not just an illustrative one.



(a) Exchange law in monoidal categories **(b)** Elimination of units and counits for adjunctions

Figure 1.1

Beyond those useful features, string diagrams have the additional advantage of fostering interdisciplinary exchanges and applications. It is not uncommon for two different communities to rely on the same mathematical structures but use different names for them. This fragmentation comes at a cost, caused by the missed opportunities to reuse results and tools and exchange ideas more broadly. Analogies between domains can be easier to detect when they manifest themselves by a visual similarity between the diagrams used. Of course, one must be careful in checking that such similarities are not vacuous, so as to avoid drawing hasty parallels.

1.2 State of the art

String diagrams were introduced independently by Hotz (1965) and Joyal and Street (1988, 1991). Both established the equivalence between the equational theory of monoidal categories and a class of transformations of planar diagrams. This result is often referred to as a *coherence* theorem, but we prefer to call it a *soundness and completeness* theorem to avoid the confusion with results establishing that all pairs of morphisms of a certain sort are equal. After this seminal theorem, other soundness and completeness results have been proved, in more or less generality, for other sorts of categories. For instance, Shum (1994) gave such a result for tortile categories and Joyal et al. (1996) for traced monoidal categories. Selinger (2010) wrote a celebrated survey of results of this sort, which we do not reproduce here.

Interestingly, many of the results presented by Selinger have only been established in special cases, and some are even conjectured entirely. That does not prevent the category theory community to rely on these diagrams extensively in practice. It would be tempting to denounce this state of affairs as being dangerous, with researchers building up results on uncertain foundations. We will argue here that this is actually a healthy practice for the community. Indeed, while the use of string diagrams in scientific communication does rely on their correctness, this dependency is subtly different to that of a theorem explicitly invoked in a proof. String diagrams are generally used as convenience methods, as the authors would also be able to write down morphisms using formulae and be explicit about where each equality comes from. Authors use string diagrams as a language, with the expectation that their readers also master and trust this language. This is analogous to textual forms of communication, where the reader is also assumed to be able to master the language and to resolve ambiguities, abuses of notation or ellipsis for instance. The possibility to use a diagrammatic language even if the corresponding soundness and completeness theorem has not been established yet is also important because it encourages notational creativity, a healthy ingredient to help build up intuitions about new structures and overcome bureaucratic conventions which distract from the actual content.

Our work is not just concerned with soundness and completeness theorems for string diagrams. We are also interested in computational questions, and more specifically in the *word problem* for categorical structures. By this, we mean the decision problem consisting of determining if two morphism expressions in some categorical structure are equal up to the axioms of this structure. In this domain, we are only aware of few results, probably because the algebraic definitions themselves are still too recent to have attracted much computational attention.

The closest results we are aware of study word problems for higher categories. The foundational work of Burroni (1993) establishes the link between the word problem for an algebraic structure and the path problem in the next dimension. Later, Makkai (2005) showed decidability of the word problem for higher cells in strict ω-categories. His algorithm was recently implemented by Forest (2021). As we will sketch in Section 2.3.5, the fact that many flavours of monoidal categories arise as truncations of higher categories could mean that Makkai's result implies decidability of the word problem for all the structures listed in the periodic table. Unfortunately, strict ω-categories do not follow the periodic table, as the notion of equality is too strict for that. For instance, a braided monoidal category cannot be recast as the 2-cells and 3-cells of an ω-category which is 2-degenerate.

Recently, there has been sustained activity in the development of computer proof assistants for string diagrams, including *Quantomatic* (Dixon et al., 2010), *Globular* (Bar et al., 2016) and its successor *homotopy.io* (Heidemann et al., 2019). These tools let users derive isotopies between string diagrams by incrementally performing moves using a graphical interface, but lack methods to discover chains of those steps. Our string diagram isotopy algorithm could yield a geometrical notion of "tactic" for such a proof assistant, automatically finding isotopies between diagrams, or rearranging diagrams to normal form.

1.3 Overview of our contributions

Our contributions are threefold. The first chapter is concerned with bimonoidal categories, for which we offer a new string diagram calculus. The main theorem

equates our category of *sheet diagrams* with the free bimonoidal category generated by a bimonoidal signature:

Theorem 3.41. *The category of sheet diagrams on a signature Σ is bimonoidally equivalent to the free bimonoidal category on Σ.*

In other words, the diagrammatic language we propose is sound and complete for the structure of bimonoidal categories. A corollary of this theorem is a simpler proof of Baez's conjecture that the groupoid of finite sets and bijections is bi-initial in the 2-category of bimonoidal categories. We also present *SheetShow*, a web-based sheet diagram renderer which can generate two-dimensional projections of sheet diagrams from a purely combinatorial encoding of their geometry.

Then, we show how those diagrams can be used to give a complete axiomatization of a certain class of programs, inspired by the data model of a popular data wrangling tool, OpenRefine:

Theorem 3.48. *Let d, d' be sheet diagrams representing dataflow programs, both with one input sheet and one output sheet. Then $d = d'$ by our axiomatization if and only if given any valuation, d and d' induce the same functions.*

Our last chapter collects complexity results on word problems for various sorts of categories. The first section is concerned with non-symmetric monoidal categories and shows that their word problem is tractable:

Theorem 4.71. *The word problem for free monoidal categories, without equations imposed on the generators, can be solved in quadratic time in the number of edges and vertices of the diagram.*

We then extend this theorem to double categories:

Theorem 4.104. *The word problem for free double categories, without equations imposed on the generators, can be solved in quadratic time in the number of edges and vertices of the diagram.*

The proof of this theorem relies on a translation from free double categories to free 2-categories, which is of interest of its own:

Theorem 4.103. *A double signature can be translated to a signature for a 2-category, such that any morphism expression in the free double category can be translated to a morphism expression in the corresponding free 2-category. Moreover, this translation preserves equivalence.*

Finally, we tackle the word problem for braided monoidal categories and show that it is at least as hard as the unknotting problem, for which no polynomial algorithm is known to date:

Theorem 4.142. *The unknotting problem can be polynomially reduced to the word problem for free braided monoidal categories, without equations imposed on the generators.*

1.4 Structure of this thesis

Chapter 2 defines notions which are used throughout the thesis and aims to do so without assuming much background knowledge beyond elementary category theory. Definitions which we assume to be known can be found in Borceux (1994) for instance.

Chapter 3 gathers our results about bimonoidal categories. Section 3.3 defines sheet diagrams and proves their soundness and completeness for bimonoidal categories. Those results are being reviewed for the *Compositionality* journal (Comfort et al., 2020). Section 3.5 applies those string diagrams to provide an axiomatization of faceted dataflow programs. This part was presented at the Applied Category Theory 2019 conference (Delpeuch, 2019). It is worth noting that the application came before the theory: indeed, the diagrams were first used informally, in the particular case of our axiomatization of dataflow programs. They were then generalized to offer a complete calculus for bimonoidal categories.

Chapter 4 is independent of Chapter 3 and gathers our three results on word problems. Section 4.2 covers the case of monoidal categories (and equivalently,

2-categories). Section 4.3 extends the result to double categories by establishing the translation mentioned above. Finally, Section 4.4 gives the hardness result for the word problem for braided monoidal categories. This last section is independent of the first two. Each section in this chapter is taken from a corresponding article, respectively Delpeuch and Vicary (2018) (to appear in *Logical Methods in Computer Science*), Delpeuch (2020) (published in *Theory and Applications of Categories*) and Delpeuch and Vicary (2021) (to appear in the proceedings of the Applied Category Theory 2021 conference).

2
Background

Contents

This chapter gives a general introduction to the research domain. The string diagrams that are at the centre of this work represent morphisms in various sorts of categories, which we present in this chapter. Rather than aiming for completeness in their treatment, the goal is to give a sense of why these notions are so useful and pervasive in many applications of category theory, and to motivate the work by highlighting certain aspects of these concepts.

2.1 Monoidal categories

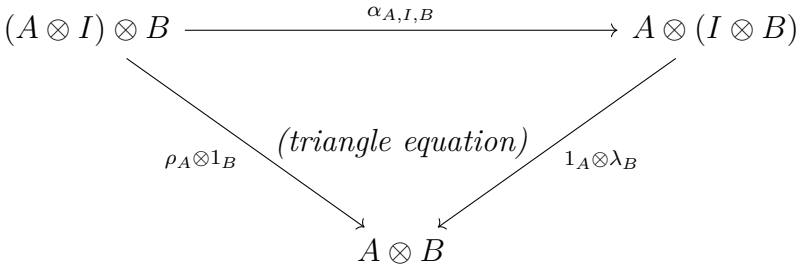
A category can be thought of as a family of typed processes (or functions, transformations) which can be composed together. Two processes (called *morphisms*) can only be composed if their types are compatible: the domain of the downstream process and the codomain of the upstream process must be the same *objects*.

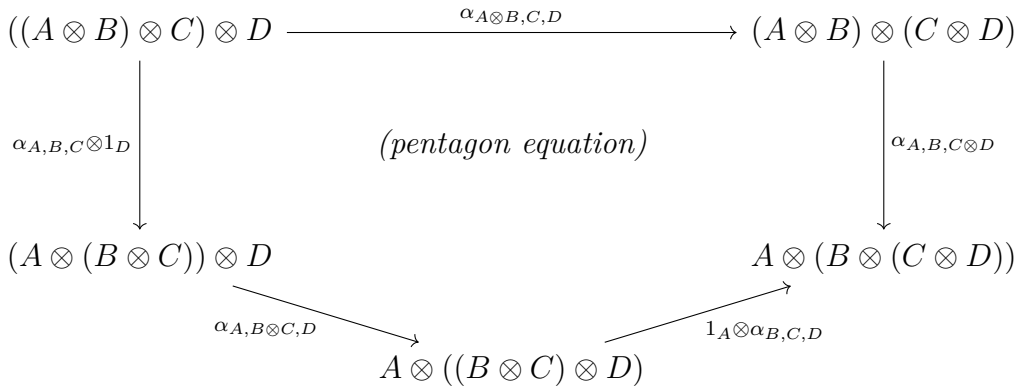
This built-in type safety is useful in many ways, but lacks the ability to represent processes with multiple inputs or outputs. From a programming perspective, functions must be able to consume multiple arguments and potentially return multiple values too. The notion of a monoidal category is one possible way to model this multiplicity, and is arguably the most widespread in applied category theory. Other approaches include operads (May, 1972) and polycategories (Szabo, 1975).

2.1.1 Weak monoidal categories

Informally, a monoidal category is a category in which objects can be concatenated. A process with multiple inputs will be represented as a morphism whose domain is the concatenation of the input objects. The concatenation of the input objects is itself an object. To turn this into a precise definition, we define a binary operator $\otimes$ on the objects, and require that it is associative and unital up to isomorphism. The precise definition was proposed by Mac Lane (1963) and refined by Kelly (1964).

Definition 2.1. *A **monoidal category** $\mathcal{C}$ is a category equipped with a bifunctor $\_ \otimes \_ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ such that $\otimes$ is associative up to natural isomorphism $\alpha_{A,B,C} : (A \otimes B) \otimes C \to A \otimes (B \otimes C)$ and has an object $I \in \mathcal{C}$ for which $\otimes$ is unital up to natural isomorphisms $\rho_A : A \otimes I \to A$ and $\lambda_A : I \otimes A \to A$. In addition, the triangle and pentagon equations below are required to hold.*





The isomorphism α is called the associator and ρ, λ are called unitors.

Let us show how this definition materializes with a concrete example. The category **Set** of sets and functions is monoidal for the cartesian product $\times$. Given sets A, B, C, we have an isomorphism $(A \times B) \times C \simeq A \times (B \times C)$ given by $((a, b), c) \mapsto (a, (b, c))$. This isomorphism is natural and satisfies the pentagon equation. The product has a unit $\{\star\}$, the one-element set, with isomorphisms $\rho_A : (a, \star) \mapsto a$ and $\lambda_A : (\star, a) \mapsto a$. The isomorphisms are again natural and satisfy the triangle equation. The last important condition to check is that $\_ \times \_ : \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ is a functor. Indeed, a monoidal product does not just define a monoid structure on the objects, but also on the morphisms. Requiring it to define a bifunctor (i.e. a functor whose domain is a product category) translates into the following properties:

$$1_A \otimes 1_B = 1_{A \otimes B} \tag{2.1}$$

$$(g \circ f) \otimes (h \circ k) = (g \otimes h) \circ (f \otimes k) \tag{2.2}$$

Equation 2.2 is called the bifunctoriality equation or exchange law, and holds as long as the left-hand side is defined (in which case the right-hand side automatically is). In the case of **Set** and the cartesian product, the product of two functions $f : A \to B$ and $g : C \to D$ is $f \times g : A \times C \to B \times D$ which is simply defined by $(f \times g)(a, c) = (f(a), g(c))$ for all $(a, c) \in A \times C$. Checking the equations above for this particular monoidal product is a good way to understand the meaning of the conditions, in particular for the bifunctoriality equation which can look a little cryptic at first. This completes our proof that the cartesian product defines a monoidal

structure on **Set**. More generally, if a category has products (in the sense of "limits of diagrams with two objects") then they form a monoidal structure in the same way.

Monoidal products are more general than categorical products, however. The canonical example for this is perhaps the category **Vect** of vector spaces and linear maps (over $\mathbb{C}$, for instance). The tensor product of vector spaces defines a monoidal structure on **Vect**, with the one-dimensional vector space I being the unit. This is interesting because the monoidal structure does not come from a categorical product, as can be seen from the fact that there cannot exist a map $\delta : A \to A \otimes A$ such that $\delta(v) = v \otimes v$ for all $v \in A$, as long as $\dim A > 0$. Such a duplication map would exist if the monoidal structure came from categorical products. The monoidal category $(\mathbf{Vect}, \otimes, I)$ is the basis of categorical quantum mechanics (Abramsky and Coecke, 2004) and this is made possible by the fact that monoidal structures do not assume copying or discarding to be possible.

Interestingly, there can be multiple monoidal structures on the same category. For instance, **Set** has another monoidal structure given by the disjoint union. Given two sets A, B one can form their disjoint union $A \sqcup B = \{1\} \times A \cup \{2\} \times B$. We have again an isomorphism $(A \sqcup B) \sqcup C \simeq A \sqcup (B \sqcup C)$ given by

$$\begin{aligned} (1, (1, a)) &\mapsto (1, a) \\ (1, (2, b)) &\mapsto (2, (1, b)) \\ (2, c) &\mapsto (2, (2, c)) \end{aligned}$$

The isomorphism is natural, associative and satisfies the pentagon equation. Similarly, the empty set is the unit for the disjoint union and the corresponding isomorphisms satisfy the required conditions. The bifunctoriality of the disjoint union can also be verified. Similar to the previous case, this is an instance of a more general fact: in a category with all coproducts, coproducts also form a monoidal structure.

The fact that both the cartesian product and the disjoint union are distinct instances of this notion of monoidal structure on the same category shows why monoidal structures are so pervasive. We have used two very different ways to

combine two objects together, and they still satisfy all the conditions required by the definition. In applications of category theory, monoidal structures can indeed be used in all sorts of situations: whether arguments are resources that are consumed or can be freely duplicated, whether arguments are all supplied to a function at once or one at a time, and so on. So although it is customary to use the $\otimes$ symbol to denote a monoidal structure, one should not forget that this structure is a priori much more general than the multiplicative conjunction of linear logic. See Clark et al. (2008); Coecke (2010); Ghani et al. (2016) for examples of applications of monoidal categories in linguistics, quantum physics and game theory respectively.

Category theory is often used to give semantics to syntaxes. This generally takes the form of functors from a syntactic category to a semantic category, which is a particular model of the theory. This requires the appropriate notion of functor, which must preserve properties of the syntax. Typically, monoidal structures are a priori not preserved by functors, hence the need for the following notion.

Definition 2.2. *Let $(\mathcal{C}, \otimes, I)$ and $(\mathcal{D}, \oplus, O)$ be monoidal categories and $F : \mathcal{C} \to \mathcal{D}$ a functor between them. The functor F is **monoidal** (or **strong monoidal**) when there exists a natural isomorphism $\eta_{A,B} : F(A \otimes B) \simeq F(A) \oplus F(B)$ and an isomorphism $\iota : F(I) \simeq O$, such that the following diagrams commute:*

$$\begin{array}{ccc}
F((A \otimes B) \otimes C) & \xrightarrow{F(\alpha_{\mathcal{C}})} & F((A \otimes (B \otimes C)) \\
\downarrow{\scriptstyle \eta_{A\otimes B,C}} & & \downarrow{\scriptstyle \eta_{A,B\otimes C}} \\
F(A \otimes B) \oplus F(C) & & F(A) \oplus F(B \otimes C) \\
\downarrow{\scriptstyle \eta_{A,B}\oplus 1_{F(C)}} & & \downarrow{\scriptstyle 1_{F(A)}\oplus \eta_{B,C}} \\
(F(A) \oplus F(B)) \oplus F(C) & \xrightarrow[\alpha_{\mathcal{D}}]{} & F(A) \oplus (F(B) \oplus F(C))
\end{array}$$

$$\begin{array}{ccc}
F(I \otimes A) & \xrightarrow{F(\lambda_{\mathcal{C}})} & F(A) \\
\downarrow{\scriptstyle \eta_{I,A}} & & \uparrow{\scriptstyle \lambda_{\mathcal{D}}} \\
F(I) \oplus F(A) & \xrightarrow[\iota\oplus 1_{F(A)}]{} & O \oplus F(A)
\end{array}
\qquad
\begin{array}{ccc}
F(A \otimes I) & \xrightarrow{F(\rho_{\mathcal{C}})} & F(A) \\
\downarrow{\scriptstyle \eta_{A,I}} & & \uparrow{\scriptstyle \rho_{\mathcal{D}}} \\
F(A) \oplus F(I) & \xrightarrow{1_{F(A)}\oplus\iota} & F(A) \oplus O
\end{array}$$

Note that a functor is monoidal with respect to specified monoidal structures, which is important since categories can have multiple monoidal structures as we have seen earlier.

2.1.2 Strict monoidal categories

When composing morphisms in a monoidal category, one often needs to insert associators ($\alpha_{A,B,C}$) or unitors (ρ_A, λ_A). This means that we need to track how our objects are bracketed at each stage of the composite and where the units are. This bookkeeping is a bit of a burden: in practice, mathematicians rarely make the structural isomorphisms explicit, since they distract from the real content. The problem is that a priori, the way we insert associators and unitors can change the overall value of a composite. The notion of a strict monoidal category solves this problem by requiring all the structural isomorphisms to be equalities.

Definition 2.3. *A **strict monoidal category** $\mathcal{C}$ is a category equipped with a bifunctor $\_ \otimes \_ : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ such that $\otimes$ is strictly associative and has a unit $I \in \mathcal{C}$.*

In this notion, the associativity and unitality of the monoidal product are equalities rather than isomorphisms. Strict monoidal categories are monoidal categories, because we can take the associators and unitors to be identities and all the required conditions are met automatically. When a monoidal category is not strict (or not assumed to be strict), we often describe it as a **weak monoidal category**.

Although many interesting monoidal categories are not strict, Mac Lane's coherence theorem provides a way to turn them into strict categories, in the following sense:

Theorem 2.4 (Mac Lane, 1963)**.** *Every monoidal category is monoidally equivalent to a strict monoidal category.*

This is one of the many possible formulations of this theorem, another common formulation being that in a free monoidal category, all pasting diagrams made out of associators, unitors and identities commute. This means that when working with a monoidal category, we can avoid having to manipulate associators and unitors in expressions by translating them to the corresponding strict monoidal category. We will extensively use this fact in the rest of this work, assuming strict monoidal structures in most of our results.

2.2 String diagrams

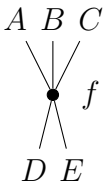
Even with associators and unitors being identities, monoidal categories give rise to a rich equational theory. For instance, given morphisms $a, b : I \to I$ (also called **scalars**), we have the following derivation, known as the Eckmann-Hilton argument:

$$
\begin{aligned}
a \circ b &= (a \otimes 1_I) \circ b && \text{(unitality of } \otimes) && (2.3)\\
&= (a \otimes 1_I) \circ (1_I \otimes b) && \text{(unitality of } \otimes) && (2.4)\\
&= (a \circ 1_I) \otimes (1_I \circ b) && \text{(bifunctoriality)} && (2.5)\\
&= a \otimes (1_I \circ b) && \text{(unitality of } \circ) && (2.6)\\
&= a \otimes b && \text{(unitality of } \circ) && (2.7)\\
&= (1_I \circ a) \otimes b && \text{(unitality of } \circ) && (2.8)\\
&= (1_I \circ a) \otimes (b \circ 1_I) && \text{(unitality of } \circ) && (2.9)\\
&= (1_I \otimes b) \circ (a \otimes 1_I) && \text{(bifunctoriality)} && (2.10)\\
&= b \circ (a \otimes 1_I) && \text{(unitality of } \otimes) && (2.11)\\
&= b \circ a && \text{(unitality of } \otimes) && (2.12)
\end{aligned}
$$

The derivation reveals multiple issues with the representation of morphisms as terms to reason about equivalence. First, the rewriting strategy used to derive the equality is not obvious: one needs to introduce identities by unitality in creative ways in steps 1, 2, 6 and 7. Therefore, it seems difficult to obtain a terminating and confluent rewriting system which would somehow normalize expressions in this presentation. Second, the bifunctoriality equation only holds when the domains and codomains of the morphisms involved are compatible: one cannot, in general, replace any expression $(g \otimes k) \circ (f \otimes j)$ by $(g \circ f) \otimes (k \circ j)$. Thus, we are required to keep track of the domains and codomains of all sub-expressions involved to understand which axiom can be applied. In the example above all domains and codomains are the monoidal unit I so the bifunctoriality equation could always be applied, but this is not true in general.

Rather than using textual formulae to represent morphisms, we can use string diagrams. The diagrams are graphical representations of morphisms in a monoidal category, which were proposed independently by Hotz (1965) and Joyal and Street (1988, 1991). We first give an informal introduction to the notation with a few examples.

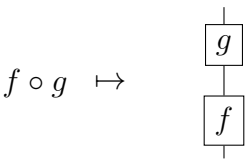
A morphism $f : A \otimes B \otimes C \to D \otimes E$ is represented by a vertex in the plane, with the edges above it corresponding to factors of its domain, and the edges below being factors of its codomain:



Throughout this thesis, morphisms will be read top-down as above. It is worth noting that some authors draw string diagrams bottom-up or left-to-right instead. For morphisms with the monoidal unit I as domain or codomain, the corresponding string diagrams do not have any edges on the corresponding side.

The identity morphism 1_A is represented as a straight line, which conveys the idea that the morphism simply forwards its input to its output, without interfering with it:

Given two morphisms f, g represented as string diagrams, we can obtain the string diagram for the composites $f \circ g$ (assuming compatible domains and codomains) and $f \otimes g$ as follows:



(a) Sequential composition

(b) Parallel composition

This is where string diagrams become useful, by exploiting two dimensions instead of one in the textual notation. The crucial observation is that with this notation, the bifunctoriality equation 2.2 becomes a simple matter of equating two different bracketings for the same diagram:

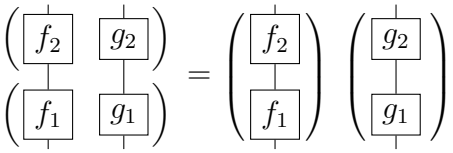


Figure 2.2: The bifunctoriality equation in string diagrams

We will see in the next section that beyond this particular case, one can actually forget how a particular diagram was built up, thanks to the guarantee that any other sequence of composites leading to the same picture will have the same meaning as a morphism.

To build up intuition about these diagrams, we give a few examples. Assume we have the following morphisms and objects:

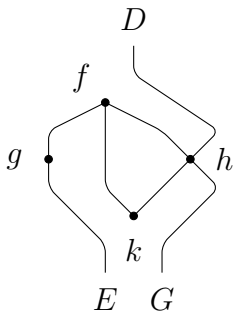
$$f : I \to A \otimes B \otimes C$$

$$g : A \to E$$

$$h : C \otimes D \to F \otimes G$$

$$k : B \otimes F \to I$$

We can form the following string diagram:



Note how the particular shape of each edge is unimportant: we choose to bend edges to allow for a compact diagram layout, but that does not change the meaning

of the diagram as long as edges "flow vertically" (meaning that their projection on the vertical axis is injective). Note the representation of morphism f (respectively k) which does not have any input (respectively no output).

Finally, we give the graphical equivalent of the Eckmann-Hilton argument given at the beginning of this section:

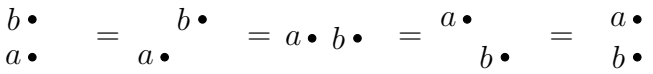


Figure 2.3: The Eckmann-Hilton argument in string diagrams

We hope that this representation provides a much clearer argument than the derivation using terms, showing why the presence of the second dimension introduced by the tensor product makes it possible to invert the order in which two scalars are composed.

2.2.1 Diagrams as mathematical objects

A crucial aspect of string diagrams is that they should be taken more seriously than illustrative pictures sketched on the back of an envelope. They are meant to provide a rigorous syntax for morphisms, as formal as terms. To support this, we need to give a mathematical definition of what string diagrams are, provide translations from terms to diagrams, and conversely. As a prerequisite to the definitions, we need some vocabulary around syntactic notions.

Definition 2.5. *A* ***monoidal signature*** Σ *is given by a set of* ***object symbols*** $\operatorname{Ob}\Sigma$*, a set of* ***morphism symbols*** $\operatorname{Mor}\Sigma$ *and domain and codomain functions* $\operatorname{dom}, \operatorname{cod} : \operatorname{Mor}\Sigma \to (\operatorname{Ob}\Sigma)^*$*, where* $(\operatorname{Ob}\Sigma)^*$ *is the set of finite words on* $\operatorname{Ob}\Sigma$*.*

For instance, we can define a monoidal signature Σ as:

$$\begin{aligned}\operatorname{Ob}\Sigma &= \{A, B, C\}\\ \operatorname{Mor}\Sigma &= \{f, g\}\\ \operatorname{dom}(f) &= [A, A]\\ \operatorname{cod}(f) &= [C]\\ \operatorname{dom}(g) &= [C]\\ \operatorname{cod}(g) &= [A, B]\end{aligned}$$

A monoidal signature can be interpreted in a monoidal category, in the following way.

Definition 2.6. *A **monoidal interpretation** i of a monoidal signature Σ into a monoidal category $(\mathcal{C}, \otimes, I)$ is given by a function $i_0 : \operatorname{Ob}\Sigma \to \operatorname{Ob}\mathcal{C}$ and a function $i_1 : \operatorname{Mor}\Sigma \to \operatorname{Mor}\mathcal{C}$ such that for each morphism symbol $f \in \operatorname{Mor}\Sigma$, $\operatorname{dom} i_1(f) = i_0(\operatorname{dom} f)$ and $\operatorname{cod} i_1(f) = i_0(\operatorname{cod} f)$ (where i_0 is extended to lists of object symbols by taking their monoidal product in $\mathcal{C}$).*

A monoidal interpretation $i : \Sigma \to \mathcal{C}$ can be composed with a monoidal functor $F : \mathcal{C} \to \mathcal{D}$, giving a monoidal interpretation of Σ into $\mathcal{D}$. With this notion, we are now equipped to discuss free monoidal categories.

Definition 2.7. *A monoidal category $(\mathcal{C}, \otimes, I)$ is **free** on a monoidal signature Σ via a monoidal interpretation $i : \Sigma \to \mathcal{C}$ when for any monoidal interpretation i' of Σ into a monoidal category $(\mathcal{D}, \oplus, O)$, there exists a monoidal functor from $F : \mathcal{C} \to \mathcal{D}$ such that $i' = F \circ i$, and F is unique up to natural monoidal isomorphism.*

See Joyal and Street (1988) for a more categorical and elegant definition of this notion and for the definition of a natural monoidal isomorphism (which is called a *tensor transformation* there). As always with notions of free algebraic structures, it is a consequence of this definition that there is a unique free monoidal category for a given signature, up to monoidal equivalence.

We are now ready to define string diagrams as topological objects, following Joyal and Street (1991). The full definition relies on a fairly long chain of definitions

which we do not reproduce here as it would be relatively tedious and uninformative. In the following definition, a and b are real numbers which correspond to the vertical positions of the input and output boundaries of the diagram.

Definition 2.8 (Joyal and Street, 1991)**.** *A* ***recumbent*** *(or* ***progressive****)* ***plane diagram*** *is an embedded graph* $\Gamma \hookrightarrow [a, b] \times \mathbb{R}$ *(see Joyal and Street (1991)) such that the projection of any edge on the vertical axis is injective.*

In the definition above, Γ is a topological graph and $[a, b] \times \mathbb{R}$ a section of the plane it gets embedded into.

Definition 2.9. *A* ***string diagram*** *on a monoidal signature* Σ *is a recumbent plane diagram where the vertices are annotated by morphism generators from* Σ *and wires are annotated by object generators, such that for each vertex the wires connected to it are annotated by the objects specified by the domain and codomain of the generator.*

We can then associate to any string diagram d a corresponding morphism $v(d)$ in the free monoidal category on Σ. This is achieved by breaking down a diagram into basic blocks of either generators or identities and composing them sequentially or in parallel according to how the basic blocks are laid out. The requirement for edges to be injective along the vertical axis ensures that there are no "cups" or "caps" which would not be interpretable without adjoints. The bifunctoriality equation and the associativity and unitality of compositions ensure that the resulting morphism does not depend on the order in which the blocks are composed, as shown in Figure 2.2. The details of this construction can be found in Joyal and Street (1988, 1991).

This interpretation gives us a translation from string diagrams to terms representing a morphism in a monoidal category. For the converse direction, the previous section shows that any term can be inductively translated to a diagram.

2.2.2 Joyal and Street's soundness and completeness theorem

The main result establishing the usefulness of string diagrams is the invariance of the interpretation as a morphism up to topological deformations.

Definition 2.10 (Definition 5 in Joyal and Street, 1988)**.** *A* ***deformation of recumbent graphs*** *is a deformation $h : \Gamma \times [0,1] \to [a,b] \times \mathbb{R}$ of planar graphs (see Definition 2.8) such that the image $\Gamma(t)$ of $h(-,t)$ is recumbent for all $t \in [0,1]$. A* ***deformation of string diagrams*** *is a deformation of the underlying recumbent graph, such that the annotating objects remain identical throughout the deformation.*

We can then state the invariance of the interpretation v defined in the previous section:

Theorem 2.11 (Theorem 3 in Joyal and Street, 1988)**.** *If $h : \Gamma \times [0,1] \to [a,b] \times \mathbb{R}$ is a deformation of string diagrams then the value $v(\Gamma(t))$ is independent of $t \in [0,1]$.*

The converse of the theorem is also true: if two expressions are equal as morphisms via the axioms of monoidal categories, then their string diagrams can be related by a deformation. Concretely, the result means that instead of manipulating terms to represent morphisms, we can simply rely on string diagrams to derive equalities in monoidal categories. For instance, Figure 2.3 can be taken not just as a depiction of the proof that $a \circ b = b \circ a$ for scalars, but an actual derivation on its own.

To finish this section, we give another view on Joyal and Street's theorem. What this theorem says is that string diagrams provide a syntax for monoidal categories. We can state this in a more categorical way, using the notion of free monoidal category generated by a signature.

Theorem 2.12. *The free monoidal category on a monoidal signature Σ has lists of object symbols from Σ as objects and string diagrams annotated with morphism symbols from Σ as morphisms, such that the boundaries of the string diagrams match the domains and codomains specified by the signature. Composition and tensor are respectively given by vertical and horizontal pasting, associators and unitors are identities.*

Another way to define the free monoidal category on a signature would be to freely generate object and morphism expressions as required by the axioms of monoidal categories, and quotient by the required equations. Therefore, stating that the free monoidal category can be defined as above in terms of string diagrams underlines their suitability as a syntax for monoidal categories.

2.2.3 Computing with string diagrams

While the diagrammatic representation is easy to manipulate at an intuitive level, one can argue that the associated topological notions are relatively complicated and precisely describing the objects involved is tedious. Graphical objects also seem harder to encode in a computer, making them of little use to solve the word problem.

In fact, it is possible to encode a string diagram more efficiently than by listing the explicit positions of its vertices and the trajectories of its edges. This relies on the notion of general position:

Definition 2.13. *A string diagram is in* ***general position*** *when none of its vertices share the same height.*

Any string diagram can be deformed slightly to be in a general position.

Lemma 2.14 (Joyal and Street, 1988)**.** *Given a string diagram* Γ*, there exists a diagram* Γ' *in general position and a deformation of recumbent diagrams between* Γ *and* Γ'*.*

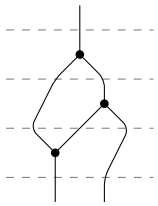
Proof. If two vertices are at the same height, then they can be slightly perturbed such that one is above the other. This can be done for each pair of vertices with recumbent transformations. □

Any string diagram in general position can be cut up in slices, each of which contains exactly one generator.

Lemma 2.15. *Given a string diagram in general position* Γ *containing* n *vertices, there exist heights* $h_1, \ldots, h_{n+1}$ *such that there is exactly one vertex between* h_i *and* h_{i+1}*.*

Proof. Let us call $y_1 < \cdots < y_n$ the heights of the n vertices in Γ. Any choice of h_i such that $h_1 < y_1 < h_2 < \cdots < h_n < y_n < h_{n+1}$ satisfies the property. □

A diagram in general position can be decomposed as a sequence of horizontal **slices**, each of which contains exactly one node, as below.



One can therefore encode the geometry of such a diagram as follows:

- The number of wires crossing the input boundary;
- The list of slices, each of which can be described by the following data:
  - The number of wires passing to the left of the node in the slice. We call this the **offset**;
  - The number of input wires consumed by the node;
  - The number of output wires produced by the node.

For the sample diagram above, this gives us the following encoding (with inputs at the top of the diagram):

```
inputs: 1
slices:
- offset: 0
  inputs: 1
  outputs: 2
- offset: 1
  inputs: 1
  outputs: 2
- offset: 0
```

```
inputs: 2
outputs: 1
```

On top of this geometrical structure, one can specify the object associated with each wire and the morphism associated with each vertex. We will use this computational structure in Chapter 4 in our solution to the word problem for monoidal categories.

2.3 Variants of monoidal categories

Monoidal structures come in all sorts of flavours and variants, and we review a few of them in this section. Some are monoidal structures with additional properties, while some are related to monoidal categories in a looser sense.

2.3.1 Symmetric monoidal categories

Many if not most of the monoidal structures of interest in applied category theory are symmetric, in the following sense.

Definition 2.16. *A monoidal category $(\mathcal{C}, \otimes, I)$ is **symmetric** when there is a natural isomorphism $s_{A,B} : A \otimes B \to B \otimes A$ such that $s_{A,B}^{-1} = s_{B,A}$ and satisfying the hexagon[1] identities:*

$$\sigma_{A,B\otimes C} = (1_B \otimes \sigma_{A,C}) \circ (\sigma_{A,B} \otimes 1_C)$$

$$\sigma_{A\otimes B,C} = (\sigma_{A,C} \otimes 1_B) \circ (1_A \otimes \sigma_{B,C})$$

Monoidal categories are often used as theories of resources (objects) and processes (morphisms) and it is common that the precise location or order of the resources consumed or produced by a morphism do not matter. In such cases it is natural to use a symmetric monoidal structure.

The string diagrams defined in the previous section can be adapted to the additional structure of symmetric monoidal categories. Concretely, it simply

[1] The name comes from the fact that when stated as commutative diagrams in the context of weak monoidal categories, they have six sides.

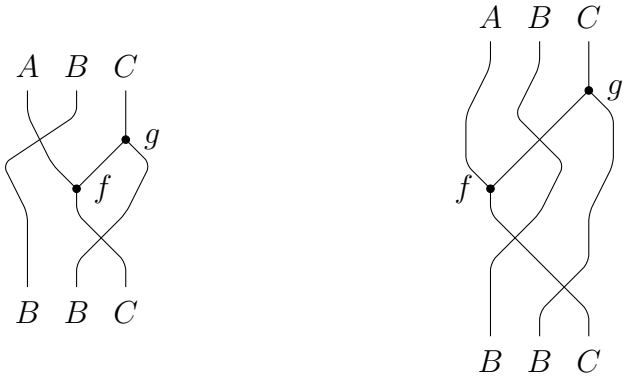


Figure 2.4: Two isomorphic symmetric monoidal string diagrams.

amounts to giving a special representation to the swap morphism $s_{A,B} : A \otimes B \to B \otimes A$, and adapting the notion of isotopy so that it captures the properties satisfied by this morphism. A string diagram in a symmetric monoidal category is given in Figure 2.4. The string diagram on the left represents the morphism $(1_B \otimes \gamma_{C,B}) \circ (1_B \otimes f \otimes 1_B) \circ (\gamma_{A,B} \otimes g)$, where $f : A \otimes A \to C$, $g : C \to A \otimes B$, and where $\gamma_{A,B} : A \otimes B \to B \otimes A$ is the symmetry.

Definition 2.17. *A **symmetric monoidal string diagram** on a symmetric monoidal signature Σ is an anchored progressive polarised diagram of Σ in the sense of Joyal and Street (1991).*

In the example signature Σ above, one can draw the string diagrams of Figure 2.4.

Definition 2.18. *An isomorphism of symmetric monoidal string diagrams $\phi : D \to D'$ is an anchored isomorphism of progressive polarised diagrams in the sense of Joyal and Street (1991).*

The two example diagrams in Figure 2.4 are isomorphic. Given a monoidal signature Σ, one can form a category $\mathbb{F}_s(\Sigma)$ whose objects are $(\mathrm{Ob}\,\Sigma)^*$ and morphisms are isomorphism classes of symmetric monoidal string diagrams on Σ.

Theorem 2.19 (Joyal and Street, 1991; Selinger, 2010)**.** *A well-formed equation between morphisms in the language of symmetric monoidal categories follows from the axioms of symmetric monoidal categories if and only if it holds, up to isomorphism*

of diagrams, in the graphical language. In other words, $\mathbb{F}_s(\Sigma)$ *is the free symmetric monoidal category on* Σ*.*

Note that the objects of the free symmetric monoidal category, $\mathrm{Ob}(\Sigma)^*$, correspond to elements of the free monoid on the object symbols of the signature. (The fact that the set of finite words is the free monoid can be seen as a one-dimensional analogue of the previous theorem.)

2.3.2 Cartesian categories

Another common feature of monoidal categories is the ability to duplicate or discard resources, typically when those resources are immaterial. In such cases, the monoidal category at hand is likely cartesian.

Definition 2.20 (Fox, 1976)**.** *A* ***cartesian category*** *is a symmetric monoidal category* $\mathcal{C}$ *equipped with a natural family of symmetric comonoids* $(\delta_A : A \to A \otimes A, \bot_A : A \to I)$ *such that* $\delta_{A\otimes B} = (1_A \otimes s_{A,B} \otimes 1_B) \circ (\delta_A \otimes \delta_B)$ *and the monoidal unit* I *is terminal. If these conditions are satisfied one may write the product as* $\times$ *instead of* $\otimes$*.*

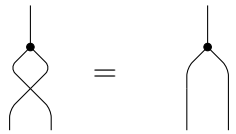
The comultiplication δ_A is the copying map and the counit $\bot_A$ is the discarding map. It can be checked that this definition of a cartesian category is equivalent to the usual one, where the product is defined as the limit of a two-point diagram (Fox, 1976; Heunen and Vicary, 2012). The idea behind defining a cartesian category as a symmetric monoidal category with extra structure is to obtain a graphical calculus for cartesian categories, by extending our string diagrammatic syntax to include the additional structure and properties.

In Figure 2.5 we represent the copying and discarding maps as explicit operations. The equations they satisfy can then be stated graphically in Figure 2.6. The presence of the monoidal symmetry in these axioms explains why cartesian categories are necessarily symmetric.

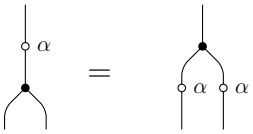
(a) Copying is unital for discarding

(b) Copying is associative

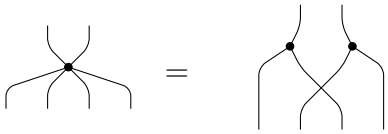
(c) Operations without outputs discard inputs



(d) Copying is symmetric



(e) Copying the output of an operation



(f) Copying a pair

Figure 2.6: Axioms of a cartesian structure in a symmetric monoidal category.

(a) Copying

(b) Discarding

(c) Arbitrary operation

Figure 2.5: Generators of a cartesian structure in a symmetric monoidal category.

String diagrams for cartesian categories are essentially directed acyclic graphs, and this graph-based representation is used in countless fields and implemented by many software packages. For instance, Figure 2.7 shows a *compositing* workflow in Blender3D[2], where the graph-based representation of the image transformation pipeline is manipulated directly by the user. In this representation, each output port of a morphism can have multiple edges connected to it: this is an alternative to using explicit copying morphisms, as we have done in the string-diagrammatic presentation given earlier, but both syntaxes are clearly equivalent.

[2] https://www.blender.org/

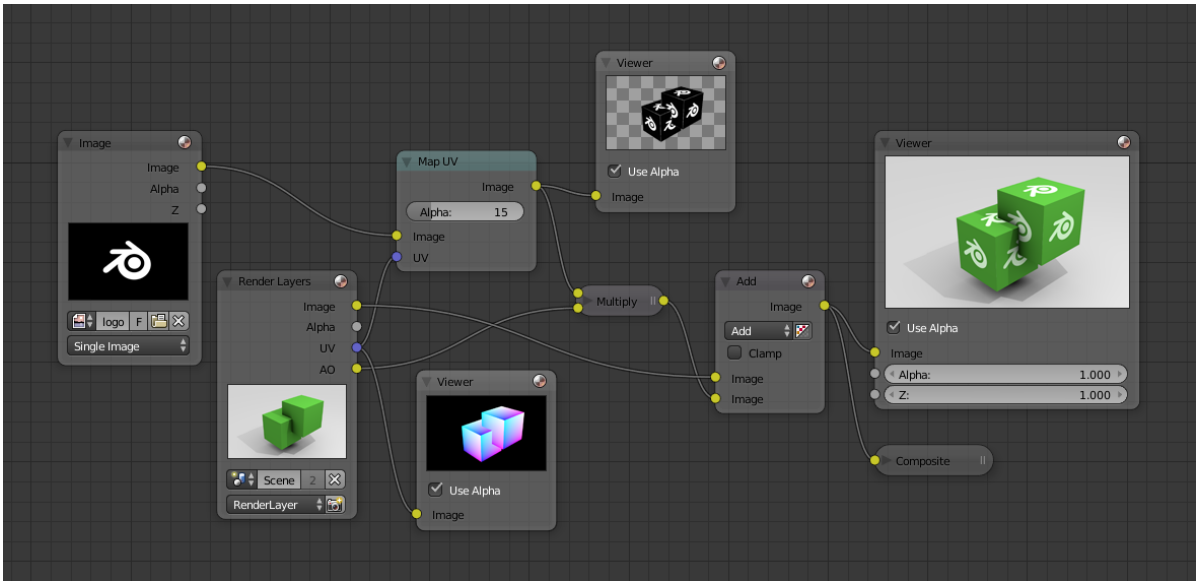


Figure 2.7: Constructing an image by composing modules in Blender3D.[3]

2.3.3 Braided monoidal categories

Another variant of monoidal categories that we find of interest are braided monoidal categories, which were introduced by Joyal and Street (1986, 1993). They are arguably not as common as symmetric monoidal categories in applied settings, but still find applications in, for instance, topological quantum computation. Their study is also motivated by the fact that they arise naturally in higher-categorical contexts, as we will explain in Section 2.3.5.

We assume the context of a strict monoidal structure but again, there exist weak versions of the following definitions, and coherence theorems show their equivalence with the strict definitions that we adopt here (Theorem 4 in Joyal and Street (1986)).

Definition 2.21. *A* ***braided monoidal category*** $\mathcal{C}$ *is a monoidal category* $(\mathcal{C}, \otimes, I)$ *equipped with a natural isomorphism* $\sigma_{A,B} : A \otimes B \to B \otimes A$*, satisfying the hexagon identities:*

$$\sigma_{A,B\otimes C} = (1_B \otimes \sigma_{A,C}) \circ (\sigma_{A,B} \otimes 1_C)$$

$$\sigma_{A\otimes B,C} = (\sigma_{A,C} \otimes 1_B) \circ (1_A \otimes \sigma_{B,C})$$

[3] Taken from https://docs.blender.org/manual/en/latest/compositing/introduction.html, CC BY-SA.

(a) String diagram for $\sigma_{A,B}$ **(b)** String diagram for $\sigma^{-1}_{A,B}$

Figure 2.8: Representation of braid morphisms as string diagrams.

Figure 2.9: The hexagon identities represented as string diagrams.

We use string diagrams for monoidal categories to represent morphisms in braided monoidal categories. Figure 2.8 shows the representation of the braid morphism and its inverse. Figure 2.9 shows the representation of the hexagon identities with this convention.

The soundness and completeness theorem of string diagrams for monoidal categories can be extended to the case of braided monoidal categories (Joyal and Street, 1991). This requires adapting again the notion of a string diagram, which is now three-dimensional, and the corresponding class of isotopies. We state the soundness and completeness theorem as formulated by Selinger (2010):

Theorem 2.22 (Theorem 5 in Selinger, 2010)**.** *A well-formed equation between morphisms in the language of braided monoidal categories follows from the axioms of braided monoidal categories if and only if it holds in the graphical language up to isotopy in 3 dimensions.*

(a) Reidemeister 2 move (σ is an iso) **(b)** Pull-through move (naturality of σ)

Figure 2.10: Equalities satisfied by braid morphisms.

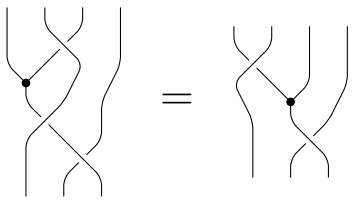
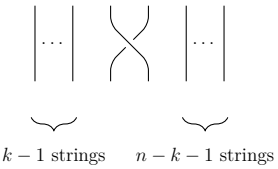
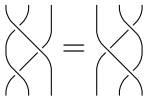


Figure 2.11: Two isotopic diagrams in a braided monoidal category.



(a) Representation of σ_k



(b) Representation of Equation 2.13

Figure 2.12: Graphical representation for the braid group.

The combinatorics of string braidings have been studied extensively, but more often from the perspective of group theory than category theory. The braid group was introduced by Artin (1947).

Definition 2.23. *The braid group on n strands B_n is the free group generated by generators $\sigma_1, \ldots, \sigma_{n-1}$ with equations*

$$\sigma_k \sigma_{k+1} \sigma_k = \sigma_{k+1} \sigma_k \sigma_{k+1} \qquad \text{for } 1 \leq k < n-1 \tag{2.13}$$

$$\sigma_i \sigma_j = \sigma_j \sigma_i \qquad \text{for } j - i > 1 \tag{2.14}$$

With this formalism, an element of the group represents a braid on n strings, while σ_k represents a positive braiding of the adjacent strings k and $k+1$, with no change on other strings. Its inverse σ_k^{-1} is the negative braiding on the same strings. Figure 2.12 shows how generators and equations of the braid group can be represented graphically. Equation 2.13 is called the Reidemeister type 3 move (or Yang-Baxter equation). It can be seen that it is a particular case of the pull-through move of Figure 2.10b, where the morphism being pulled through is a braid itself. Equation 2.14 corresponds to the exchange moves which hold in any monoidal category.

We can make the connection between this group-theoretic presentation and braided monoidal categories more precise. Given a monoidal signature, one can

generate the free braided monoidal category on it. By Theorem 2.22, this is the category of braided string diagrams whose vertices and edges are labelled by generating objects and morphisms respectively. Note that we are not imposing any additional equation between the generators: the only equations which hold are those implied by the braided monoidal structure itself.

Proposition 2.24 (Joyal and Street, 1993)**.** *The free braided monoidal category $\mathcal{B}$ generated by the signature $(\{A\}, \emptyset)$ is the braid category, i.e. $\mathcal{B}(A^n, A^n) = B_n$, the group of braids on n strands.*

Proof. This can be seen in string diagrams: a morphism in $\mathcal{B}$ can only be made of identities, and positive and negative braids. As a string diagram in a monoidal category, it can be drawn in general position, in which all braids appear at different heights. This can therefore be decomposed as a sequential composition of slices containing exactly one positive or negative braid. The number of wires between each slice remains constant given that braids and identities always have as many outputs as inputs: let us call this number n. Each of these slices corresponds to a generator or generator inverse in B_n. As explained above, the equations holding in $\mathcal{B}(A^n, A^n)$ and B_n are the same, hence the equality. □

Therefore, braided monoidal categories generalize the braid group by allowing for other morphisms than braids and identities.

2.3.4 Bicategories and 2-categories

So far we have listed a few notions which are special cases of monoidal categories. Let us now turn to a generalization of the notion: bicategories.

Definition 2.25. *A **bicategory** $\mathcal{C}$ consists of:*

- *A collection $\mathrm{Ob}\,\mathcal{C}$ of **0-cells**.*
- *For each pair of objects $A, B \in \mathrm{Ob}\,\mathcal{C}$, a category $\mathcal{C}(A, B)$ whose objects are the **1-cells** with domain A and codomain B. Given two 1-cells $f, g : A \to B$, the morphisms between them in $\mathcal{C}(A, B)$ are called **2-cells** with domain f and codomain g, respectively.*

- *For each object $A \in \mathrm{Ob}\,\mathcal{C}$, a distinguished 1-cell $1_A : A \to A$.*
- *For all objects $A, B, C \in \mathrm{Ob}\,\mathcal{C}$ a composition functor $\_ \circ \_ : \mathcal{C}(B,C) \times \mathcal{C}(A,B) \to \mathcal{C}(A,C)$.*
- *Natural isomorphisms witnessing the unitality of the composition functor: $1_B \circ \_ \simeq 1_{\mathcal{C}(A,B)}$ and $\_ \circ 1_A \simeq 1_{\mathcal{C}(A,B)}$ for all objects $A, B \in \mathcal{C}$.*
- *Natural isomorphisms similarly witnessing the associativity of the composition functor.*

The associators and unitors are required to satisfy the pentagon and triangle equations as in the definition of monoidal categories.

One example of a bicategory is **Cat**, the bicategory whose 0-cells are categories with functors as 1-cells and natural transformations as 2-cells. In fact, one can check that the natural isomorphisms required in the last two points of the definition are identities in this particular example. This motivates a stricter definition:

Definition 2.26. *A 2-category is a bicategory whose composition functor is strictly unital and associative.*

The reason why we are interested in bicategories and 2-categories is that they generalize monoidal categories, in the following sense.

Proposition 2.27. *Let $\mathcal{C}$ be a bicategory with a single 0-cell: $\mathrm{Ob}\,\mathcal{C} = \{\star\}$. Then one can form a monoidal category whose objects are the 1-cells $\mathrm{Ob}\,\mathcal{C}(\star,\star)$ and morphisms are the 2-cells of $\mathcal{C}$. The composition of morphisms is given by the composition in $\mathcal{C}(\star,\star)$ and the monoidal product is given by the composition functor in the definition above. Conversely, any monoidal category defines a bicategory in an analogous way.*

Similarly, 2-categories with a single object correspond to strict monoidal categories. Interestingly, the string diagrams for monoidal categories can be generalized to obtain string diagrams for 2-categories or bicategories. Instead of drawing

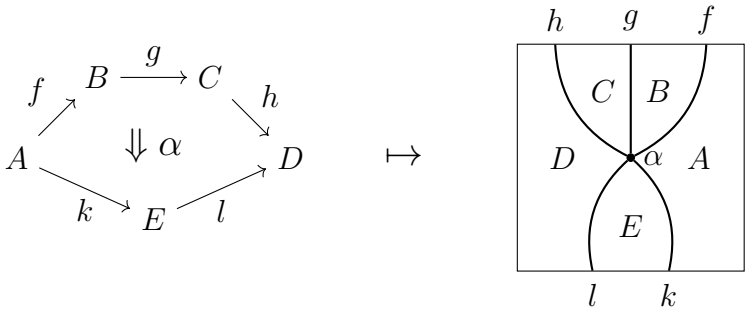


Figure 2.13: Obtaining a string diagram by duality from a commutative diagram.

diagrams to represent morphisms, we now draw diagrams for 2-cells. In the diagrams, nodes correspond to 2-cells, edges to 1-cells and regions to 0-cells. Therefore, the only difference is that we need to annotate the regions with our string diagrams with 0-cells, which is often done by colouring the regions. By doing so, we ensure that we make it syntactically impossible to compose two 1-cells with incompatible domains and codomains. In comparison, monoidal categories make it possible to consider the monoidal product of any two objects or any two morphisms.

In a sense, 2-categories also shed additional light on string diagrams by relating them to the classical pasting diagrams that category theorists generally use. Given a commutative diagram for a 2-cell, the corresponding string diagram can be obtained by taking the dual of the commutative diagram, as in Figure 2.13.

2.3.5 Higher categories

The notions of 2-category or bicategory suggest iterations of the idea: adding 3-cells to get 3-categories or tricategories, adding 4-cells to get tetracategories, and so on. They are called higher categories, as they consist of many layers of cells piling up on top of each other. The definitions are fairly involved, and we will not use them directly in our work, so we keep this section informal. The aim is only to convey the motivation behind some of our other results. Let us assume that we have a notion of n-category, for $n \geq 0$, which has 0-cells up to n-cells, similar to the case of 2-categories from the previous section. In particular, a 1-category is just a category, and a 0-category only has a collection of 0-cells, so it is essentially a set without any other structure. Just as we considered 2-categories which had

$k \backslash n$	0	1	2	3
0	set	category	2-category	3-category
1	$\{\star\}$	monoid	monoidal cat.	monoidal 2-cat.
2	$\vdots$	$\{\star\}$	comm. monoid	braided monoidal cat.
3		$\vdots$	$\{\star\}$	comm. monoid
4			$\vdots$	$\{\star\}$

Figure 2.14: The periodic table of n-categories.

only one 0-cell, we can do the same for higher categories. In fact, one can consider not just n-categories which have only one 0-cell: we can ask the first few layers of cells to all be trivial. More precisely:

Definition 2.28. *An n-category $\mathcal{C}$ is k-**degenerate** if it has only one p-cell for all $0 \leq p < k$.*

Just as a 2-category with only one 0-cell is a monoidal category, a k-degenerate n-category induces an $(n-k)$-category $\mathcal{D}$ by shifting k-cells down: the k-cells of $\mathcal{C}$ are the 0-cells of $\mathcal{D}$, and so on. This $(n-k)$-category has extra structure induced by the n-categorical structure. With this principle in mind, one can try to determine (or simply guess) what the induced structure is depending on n and k. This gives rise to the so-called periodic table of n-categories. Figure 2.14 gives a reduced version of it, which can be extended in various ways. Just like the periodic table of chemical elements, its contents were originally guessed and establishing the corresponding theorems can be a significant amount of work. Even stating the results properly is in itself non-trivial, since the desired result is generally not just about a correspondence between instances of the notions involved, but also the functors between them, the natural transformations between those, and so on.

Despite the difficulty in making its predictions fully precise and verified, the periodic table is of interest to us as a resource to discover variants of monoidal categories which somehow arise naturally, as simple cases of more complicated structures. Studying the word problem for the categories mentioned in the table is a way to approach the word problem for higher categories from simpler cases. In

particular, hardness or undecidability results on categories found in the periodic table would then directly transfer to more general versions of those categories. In terms of applications, the results then let us better understand what sort of automation one can hope for, in the context of proof assistants for higher categories such as homotopy.io (Heidemann et al., 2019).

3

Bimonoidal categories

Contents

The first four sections in this chapter are taken from the article *Sheet diagrams for bimonoidal categories* (Comfort et al., 2020), under review for the Compositionality journal. The last section is taken from the article *A complete language for faceted dataflow programs* (Delpeuch, 2019), presented at the Applied Category Theory 2019 conference.

3.1 Introduction

The previous chapter made the case for string diagrams as a convenient and rigorous syntax for morphisms in monoidal categories. However convenient this syntax might be, it is limited by the expressivity of the monoidal structure. It is common to work in settings where two monoidal structures are used; however, monoidal string diagrams are by definition a syntax for one particular monoidal structure. And a priori, there is no reason why one might expect that more than one monoidal structures interact well with each other. However, there are certain instances when there is some sort of useful interaction between both monoidal structures.

In this chapter we focus our attention to a particular type of distributivity between two monoidal structures on a category. Bimonoidal categories (also known as rig categories) are categories with a monoidal structure $\otimes$ and a symmetric monoidal structure $\oplus$, with natural isomorphisms

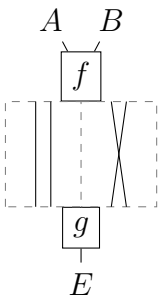
$$\delta_{A,B,C} : A \otimes (B \oplus C) \to (A \otimes B) \oplus (A \otimes C)$$

$$\delta^{\#}_{A,B,C} : (A \oplus B) \otimes C \to (A \otimes C) \oplus (B \otimes C)$$

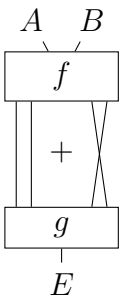
called *distributors*, distributing $\otimes$ over $\oplus$ from the left and the right, satisfying certain coherence laws. Many well-known categories have such a structure, for example **Set** with disjoint unions and cartesian products, or **Vect** with direct sums and tensor products. Some informal attempts have been made at drawing string diagrams for such categories. Consider for instance the following linear map, a morphism in **Vect**:

$$A \otimes B \xrightarrow{f} (C \otimes D) \oplus (C \otimes D) \xrightarrow{1_{C\otimes D} \oplus \gamma_{C,D}} (C \otimes D) \oplus (D \otimes C) \xrightarrow{g} E$$

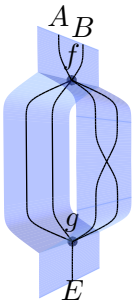
where $\gamma_{C,D} : C \otimes D \to D \otimes C$ is the symmetry for $\otimes$. Authors have used various informal conventions to represent such a morphism as a diagram:



(a) Duncan (2009)



(b) James and Sabry (2012)



(c) Staton (2015)

These conventions all communicate the structure of a morphism to readers, but do not a priori enjoy a soundness and completeness theorem. In this chapter we develop the formal theory of diagrams for bimonoidal categories (also known as rig categories, semiring categories). We provide a definition of the class of diagrams (that we call *sheet diagrams*), their deformations and a soundness and completeness theorem for them. Our sheet diagrams follow the three-dimensional style used by Staton (2015), retrospectively justifying their use as formal reasoning tools. The central theorem for this result is as follows:

Theorem 3.41. *The category of sheet diagrams on a signature Σ is bimonoidally equivalent to the free bimonoidal category on Σ.*

Sheet diagrams represent morphisms in a bimonoidal category in a normal form, as a "sum of products" with $\otimes$ pushed to the inside and $\oplus$ pushed to the outside by the distributors, similar to the normal form of a polynomial, or disjunctive normal form in logic. This is intended as a compromise, making sheet diagrams both easier to visualise and also easier to typeset, in return for which the tensor product of sheet diagrams is a rather complicated derived operation. It seems also likely that one could define a class of string diagrams which does not require normalizing objects as a sum of products, but this would also come with significant difficulties and is therefore left for future work.

Given that our diagrams are three-dimensional, it is natural to wonder whether they could be related to other classes of three-dimensional diagrams already known, such as those for Gray categories (Hummon, 2012). In fact, our diagrams cannot

be recast as surface diagrams for Gray categories because their boundaries are not necessarily globular. We could consider surface diagrams for double monoidal categories (which have not been formally defined to the best of our knowledge), which do not require globularity. While it might be possible to recast our sheet diagrams as such surface diagrams, the benefit of doing so is not immediately clear as no coherence theorem for such diagrams is known and some composition operations of those cubical surface diagrams would not make sense in the bimonoidal case.

Bimonoidal categories have found applications in a variety of fields: probability theory (Fritz and Perrone, 2018), quantum information (Staton, 2015), dataflow computations (see Section 3.5), game theory (Hedges, 2018), and reversible computation (James and Sabry, 2012). They are also studied in K-theory (Guillou, 2009; Gomez, 2009). Sheet diagrams could potentially be used in each of these fields, but one important obstacle for using these diagrams is the difficulty of typesetting and manipulating them. We introduce a web-based tool called SheetShow,[1] which renders sheet diagrams as vector graphics based on a purely combinatorial description of their geometry. We give an overview of the data structures of this tool in Section 3.3.4, which are based on our data structures for monoidal categories laid out in Section 2.2.3.

A corollary of this theorem is a simpler proof of Baez's conjecture that the groupoid of finite sets and bijections is bi-initial in the 2-category of bimonoidal categories.

Finally, we show how those diagrams can be used to give a complete axiomatization of a certain class of programs, inspired by the data model of a popular data wrangling tool, OpenRefine. The central theorem for this section states soundness and completeness of our axiomatization:

Theorem 3.48. *Let d, d' be sheet diagrams representing dataflow programs, with one input sheet and one output sheet. Then $d = d'$ by our axiomatization if and only if given any valuation, d and d' induce the same functions.*

[1] Available at https://wetneb.github.io/sheetshow/ (web app) and https://github.com/wetneb/sheetshow (source code)

Beyond this result, the motivation for this axiomatization is to open up possibilities to visualize and manipulate workflows more easily in the tool itself. Work to implement this in the OpenRefine tool is ongoing.

3.2 Bimonoidal categories

Definition 3.1. *A **bimonoidal category**, or **rig category**, is a category $\mathcal{C}$ with a monoidal structure $(\mathcal{C}, \cdot, I)$ and a symmetric monoidal structure $(\mathcal{C}, \oplus, O)$ with natural isomorphisms called the **left and right distributors**:*

$$\delta_{A,B,C} : A(B \oplus C) \to AB \oplus AC$$
$$\delta^{\#}_{A,B,C} : (A \oplus B)C \to AC \oplus BC$$

*and isomorphisms called the **left and right annihilator**:*

$$\lambda^*_A : OA \to O$$
$$\rho^*_A : AO \to O$$

satisfying the coherence conditions given in Appendix A.

For instance, **Set** is bimonoidal when equipped with the monoidal structures $(\mathbf{Set}, \times, \{\star\})$ and $(\mathbf{Set}, \sqcup, \emptyset)$, where $\times$ is the cartesian product and $\sqcup$ is the disjoint union. Similarly, $\mathbf{Vect}_k$ (the category of vector spaces on a field k) is bimonoidal for $(\mathbf{Vect}, \otimes, k)$ and $(\mathbf{Vect}, \oplus, O)$.

One important question when defining categorical structures such as this one is whether they satisfy coherence. By coherence, we mean that any two parallel morphisms generated by the structural isomorphisms $\delta, \delta^{\#}, \lambda^*, \rho^*$ and those of the monoidal structures are equal. For bimonoidal categories, this unfortunately does not hold. The symmetry of $(\mathcal{C}, \oplus, O)$ makes this impossible, as the isomorphisms $\gamma_{A,A}$ and $1_A \oplus 1_A$ are parallel and distinct.

However, a restricted form of coherence for bimonoidal categories was proved by Laplaza (1972). It applies when the domain (or equivalently codomain) of the parallel pair of morphisms is *regular*, which essentially means that no two

occurrences of the same object generator can be swapped by a symmetry. We will make this precise in Section 3.3.1 as Theorem 3.13. We finish this section with a semi-strictification theorem for bimonoidal categories.

Definition 3.2. *A bimonoidal category is* ***left-semistrict*** *(resp.* ***right-semistrict****) if both monoidal structures are strict and* $\delta^{\#}_{A,B,C}$ *(resp.* $\delta_{A,B,C}$*) is an identity for all* A, B, C*. It is* ***semistrict*** *if it is either left- or right-semistrict.*

Note that we cannot reasonably require a bimonoidal category to be both left- and right-semistrict at the same time. Indeed, if both distributors are identities, then by coherence axiom (IV) in Appendix A, one can deduce that the symmetry of $(\mathcal{C}, \oplus, O)$ would be equal to the identity, which is undesirable. However, partial strictification is always possible, as the following theorem establishes.

Theorem 3.3 (Guillou, 2009)**.** *Any bimonoidal category is equivalent to a semistrict one.*

The aim of the following section is to characterize the free bimonoidal category on a signature as a certain category of diagrams quotiented by certain topological equivalences. In a sense, this will generalize the results of Joyal and Street (1991) for monoidal categories and for symmetric monoidal categories, as the diagrammatic calculi for both non-symmetric and symmetric monoidal categories can be recovered as special cases of our sheet diagram calculus.

3.3 Sheet diagrams

3.3.1 Bimonoidal signatures

Given a set X, let $\mathrm{R}(X)$ denote the set of expressions generated by the two binary operators $\cdot$ and $\oplus$, the two nullary symbols I and O, and elements of X as nullary symbols. For instance, $\mathrm{R}(X)$ contains expressions such as $A \cdot (O \oplus B)$ or $A \cdot (B \oplus (I \oplus A))$. In other words, $\mathrm{R}(X)$ is the set of bimonoidal object expressions on generators X.

Definition 3.4. *A **bimonoidal signature** Σ consists of a set $\mathrm{Ob}\,\Sigma$ of **object symbols**, a set $\mathrm{Mor}\,\Sigma$ of **morphism symbols**, together with domain and codomain functions $\mathrm{dom}, \mathrm{cod} : \mathrm{Mor}\,\Sigma \to R(\mathrm{Ob}\,\Sigma)$. We write $f : \mathrm{dom}(f) \to \mathrm{cod}(f)$.*

One can then give a tautological definition of the free bimonoidal category such a signature generates.

Definition 3.5. *Given a bimonoidal signature Σ, define the **free bimonoidal category** $\overline{\Sigma}$ that it generates: the objects of $\overline{\Sigma}$ are given by $R(\mathrm{Ob}\,\Sigma)$; the morphisms are equivalence classes of morphism expressions built out of $\mathrm{Mor}\,\Sigma$, structural isomorphisms from Definition 3.1 and identities quotiented by the axioms of bimonoidal categories.*

This definition is not particularily easy to work with, since morphism expressions can easily get cluttered with structural isomorphisms due to the interplay between the three binary operations involved: $\circ$, $\cdot$ and $\oplus$. By defining sheet diagrams for bimonoidal categories, we will provide a more practical description of $\overline{\Sigma}$, where the bimonoidal axioms are interpreted as topological deformations.

Similarly, Definition 3.4 is the most general form of a bimonoidal signature, but again not the most convenient to work with. Just like monoidal signatures normalize the domains and codomains of their morphism symbols by forgetting the bracketing of their products, we will introduce a similar normalization for bimonoidal signatures. First, by Theorem 3.3 we can assume up to equivalence that both the additive and multiplicative monoidal structures are strict. We therefore omit the associators and unitors in what follows.

Definition 3.6. *An expression $\phi \in \mathrm{R}(X)$ is **normalized** when it is a sum of products of generators (elements of X). Each summand can have no factors (in which case the summand is simply I) and the sum can have no summands, in which case $\phi = O$. A bimonoidal signature is normalized when all its domains and codomains of morphism symbols are normalized.*

For any bimonoidal signature, we want to define a normalized version of it. This requires a bit more care than in the monoidal case because there can be multiple ways to normalize an object expression. For instance, the expression $(A \oplus B)(C \oplus D)$ is equivalent to two normalized forms: $AC \oplus AD \oplus BC \oplus BD$ and $AC \oplus BC \oplus AD \oplus BD$.

Definition 3.7. *For any expression $A \in \mathrm{R}(X)$ we define its normal form $N(A) = \bigoplus_i A_i$ where A_i are products, by induction:*

- $N(O) = O$
- $N(I) = I$
- $N(A \oplus B) = N(A) \oplus N(B)$
- $N(A \cdot B) = \bigoplus_i \bigoplus_j A_i B_j$, *where $N(A) = \bigoplus_i A_i$ and $N(B) = \bigoplus_j B_j$ with A_i, B_j products of generators.*

For example, $N((A \oplus B)(C \oplus D)) = AC \oplus AD \oplus BC \oplus BD$.

Definition 3.8. *Let $A_1, \ldots, A_p$ and $B_1, \ldots, B_q$ be object expressions. We define an isomorphism $\Delta_{p,q} : (\bigoplus_i A_i)(\bigoplus_j B_j) \to \bigoplus_i \bigoplus_j A_i B_j$ by repeated application of the distributor δ. Formally, the definition is by induction on p and q:*

- *For $p = 0$, $\Delta_{0,q} = \lambda^*_{\oplus_j B_j} : O(\bigoplus_j B_j) \to O$*
- *For $p = 1$, $\Delta_{p,q}$ is obtained by repeated applications of the distributor δ. More precisely:*
  - *For $p = 1$ and $q = 0$, $\Delta_{1,0} = \rho^*_{A_1} : A_1 O \to O$*
  - *For $p = 1$ and $q > 0$,*

$$\begin{aligned} \Delta_{1,q} : A_1(\textstyle\bigoplus_{j=1}^{q} B_j) &= A_1(\textstyle\bigoplus_{j=1}^{q-1} B_j \oplus B_q) \\ &\xrightarrow{\delta_{A_1, \bigoplus_{j=1}^{q-1} B_j, B_q}} A_1 \textstyle\bigoplus_{j=1}^{q-1} B_j \oplus A_1 B_q \\ &\xrightarrow{\Delta_{1,q-1} \oplus 1_{A_1 B_q}} \textstyle\bigoplus_{j=1}^{q-1} A_1 B_j \oplus A_1 B_q \\ &= \textstyle\bigoplus_{j=1}^{q} A_1 B_j \end{aligned}$$

- *For $p > 1$, $(\bigoplus_i A_i)(\bigoplus_j B_j) \to_\delta (\bigoplus_{i=1}^{p-1} A_i)(\bigoplus_j B_j) \oplus A_p(\bigoplus_j B_j)$. We can apply $\Delta_{p-1,q}$ on the left-hand side and $\Delta_{1,q}$ on the right-hand side.*

Definition 3.9. *For any object expression $A \in \mathrm{R}(X)$ we define its normalization morphism $n_A : A \to N(A)$ by induction:*

- $n_O = 1_O$
- $n_I = 1_I$
- $n_{A \oplus B} = n_A \oplus n_B$
- $n_{A \cdot B} = \Delta_{p,q} \circ (n_A \cdot n_B)$ *where p and q are the number of summands in $N(A)$ and $N(B)$ respectively.*

Note that every normalization morphism is an isomorphism.

Definition 3.10. *Given any bimonoidal signature Σ, we can build a normalized signature $N(\Sigma)$ where all domains and codomains are normalized by N:*

$$\begin{aligned} \mathrm{Ob}\, N(\Sigma) &= \mathrm{Ob}\, \Sigma \\ \mathrm{Mor}\, N(\Sigma) &= \{f' : N(\mathrm{dom}\, f) \to N(\mathrm{cod}\, f) \mid f \in \mathrm{Mor}\, \Sigma\} \end{aligned}$$

Theorem 3.11. *For any bimonoidal signature Σ, the categories $\overline{\Sigma}$ and $\overline{N(\Sigma)}$ are bimonoidally isomorphic.*

Proof. We define a functor $U : \overline{\Sigma} \to \overline{N(\Sigma)}$, which is the identity on objects and translates a morphism in $\overline{\Sigma}$ to $\overline{N(\Sigma)}$, by structural induction on the morphism expression as follows. For the base case, for any ϕ in $\mathrm{Mor}\, \Sigma$, define $U(\phi) := n^{-1}_{\mathrm{cod}\, \phi} \circ \phi' \circ n_{\mathrm{dom}\, \phi}$. Moreover, for any object A, define $U(1_A) := 1_A$ as required. And similarly, for all the structural natural isomorphisms of a bimonoidal category, define $U(\delta_{A,B,C}) := \delta_{A,B,C}$, $U(\delta^{\#}_{A,B,C}) := \delta^{\#}_{A,B,C}$ and so on. For the inductive case, consider some morphisms $\phi_1 : A \to B$, $\phi_2 : B \to C$ in $\overline{\Sigma}$ for which $U(\phi_1)$ and $U(\phi_2)$ are already defined. Then define $U(\phi_2 \circ \phi_1) := U(\phi_2) \circ U(\phi_1)$. Similarly, consider some morphisms $\phi_1 : A \to B$, $\phi_2 : C \to D$ in $\overline{\Sigma}$ for which $U(\phi_1)$ and $U(\phi_2)$ are already defined. Then define $U(\phi_1 \oplus \phi_2) := U(\phi_1) \oplus U(\phi_2)$ and $U(\phi_1 \cdot \phi_2) := U(\phi_1) \cdot U(\phi_2)$.

Similarly, define a functor $V : \overline{N(\Sigma)} \to \overline{\Sigma}$ which is the identity on objects and whose definition on objects is completely analogous, by structural induction on the signature of $\operatorname{Mor} N(\Sigma)$ and the axioms of a bimonoidal category. For the base case, this functor takes generators to $V(\phi') := n_{\operatorname{cod}\phi} \circ \phi \circ n^{-1}_{\operatorname{dom}\phi}$. The rest of the induction is defined as before.

The only thing left to prove is that the two assignments between Σ and $\overline{N(\Sigma)}$ are well-defined; as they preserve the bimonoidal structure and are inverse to each other by construction. That is to say, we have to show that they preserve the equivalence relations on morphisms induced by the axioms of a bimonoidal category. But since both assignments only replace generators by expressions, the equivalence relation induced by the bimonoidal structure is automatically preserved. The two functors are bimonoidal (in the sense of Appendix A.1) and mutually inverse. □

As a consequence, we will only consider normalized signatures in what follows, as this will not restrict the generality of our results. Let us now turn to coherence. As mentioned earlier, coherence does not hold for bimonoidal categories in general, but only when some conditions on their domain (or equivalently codomain) are met.

Definition 3.12 (Laplaza, 1972)**.** *An object $A \in \operatorname{Ob}\Sigma$ is **regular** when all the summands in $N(A)$ are distinct (they are all different lists of generators) and for each summand of $N(A)$, its factors are all different (it is a product of distinct generators).*

For instance, $AB \oplus BA$ is regular, but $AB \oplus AB$ and $AA \oplus AB$ are not. Note that the second condition (each summand being a product of distinct generators) was required by Laplaza because they assumed the multiplicative monoidal structure to be symmetric. We only assume the additive structure to be symmetric, so this condition should be superfluous in our case. We keep it for the sake of accurate citation as we will be able to accommodate this artificial restriction later on, when using the following theorem:

Theorem 3.13 (Laplaza, 1972)**.** *Let A, B be regular objects of $\overline{\Sigma}$. For all morphisms $f, g : A \to B$ generated by structural isomorphisms of $\overline{\Sigma}$, $f = g$.*

We finish this section with a lemma on the normalization function N.

Lemma 3.14. *For all objects* $A, B, C \in \operatorname{Ob} \overline{\Sigma}$*,*

$$N(A \cdot N(B \cdot C)) = N(N(A \cdot B) \cdot C)$$

Proof. Let $N(A) = \bigoplus_i A_i$, $N(B) = \bigoplus_j B_j$, $N(C) = \bigoplus_k C_k$. Then

$$\begin{aligned} N(A \cdot N(B \cdot C)) &= N(A \cdot (B \cdot C)) \\ &= \bigoplus_i \bigoplus_{j,k} A_i(B_j C_k) \\ &= \bigoplus_{i,j} \bigoplus_k (A_i B_j) C_k \\ &= N((A \cdot B) \cdot C) \\ &= N(N(A \cdot B) \cdot C) \end{aligned}$$

□

3.3.2 Defining sheet diagrams

In this section, we assume a fixed normalized bimonoidal signature Σ. Because bimonoidal categories are symmetric monoidal categories for their additive structure, we can already use string diagrams for symmetric monoidal categories to reason about bimonoidal categories. Such a diagrammatic language treats the multiplicative structure as opaque, but we will see in this section how to extend the language to take the second monoidal structure into account as well.

The first step consists in defining a monoidal signature which we will use for our symmetric monoidal diagrams.

Definition 3.15. *The monoidal signature* Γ *is given by*

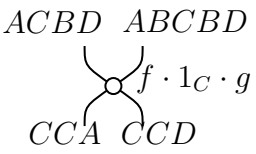
$$\begin{aligned} \operatorname{Ob} \Gamma &= (\operatorname{Ob} \Sigma)^* \\ \operatorname{Mor} \Gamma &= (\operatorname{Mor} \Sigma \cup \{1_A | A \in \operatorname{Ob} \Sigma\})^* \\ \operatorname{dom}[f_1, \ldots, f_n] &= N[\operatorname{dom}(f_1) \cdot \ldots \cdot \operatorname{dom}(f_n)] \\ \operatorname{cod}[f_1, \ldots, f_n] &= N[\operatorname{cod}(f_1) \cdot \ldots \cdot \operatorname{cod}(f_n)] \end{aligned}$$

where $\operatorname{dom}(1_A) = \operatorname{cod}(1_A) = A$.

We are slightly abusing notation in the previous definition: The normalized domains and codomains are sums of products, which we interpret as lists of lists, that is, as lists of object symbols of Γ.
The signature Γ generates a symmetric monoidal category $\mathcal{C}$ for the additive structure. Object generators are multiplicative products of bimonoidal generators, therefore objects in $\mathcal{C}$ are sums of products of bimonoidal generators.

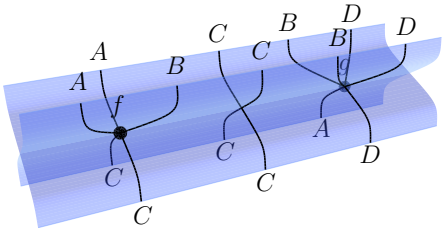
Similarly, morphism generators in Γ are multiplicative products of morphism generators in Σ, except that we also allow for identities to be part of the product. The domain of a morphism generator is obtained by normalizing the product of the domains of its components.

For instance, consider object generators A, B, C, D and bimonoidal morphism generators $f : A \oplus AB \to C$ and $g : BD \to A \oplus D$. We can form $f \cdot 1_C \cdot g \in \operatorname{Mor}\Gamma$. Its domain is $N((A \oplus AB) \cdot C \cdot BD) = ACBD \oplus ABCBD$ and its codomain is $N(C \cdot C \cdot (A \oplus D)) = CCA \oplus CCD$. When drawn as a string diagram generator, it looks as follows:



This is not very informative, since only the additive monoidal structure is reflected by the diagram. The multiplicative structure is left unanalyzed because it is internal to the object and morphism generators.

To fix this issue, we extrude our monoidal diagram into a sheet diagram. Edges of our monoidal diagram become sheets, and vertices become seams. On the sheets, we can draw wires whose connectivity reflects which factor of the morphism generator they are coming from:



Informally, each factor of the morphism generator corresponds to a node on the seam, in the same order. These nodes are represented here by small black spheres, except for the middle one which corresponds to an identity, which we leave unmarked. This idea can be generalized to arbitrary diagrams. This section defines these diagrams and the associated class of topological transformations.

Definition 3.16. *Let* $[f_1, \ldots, f_n] \in \operatorname{Mor} \Gamma$ *and let* $\operatorname{dom}[f_1, \ldots, f_n] = \bigoplus_{j=1}^{p} \bigotimes_{k=1}^{q_j} A_{jk}$ *be its domain. For each* j, k *we define the* ***origin*** *of* A_{jk} *in* $[f_1, \ldots, f_n]$ *as the index* $1 \leq i \leq n$ *such that* A_{jk} *occurs in* $\operatorname{dom} f_i$*. If* A_{jk} *occurs in the domain of more than one morphism factor, this can be made unambiguous by adding indices to the object symbols involved in the domains before normalization. Similarly, the origin of an object symbol in the codomain of a morphism generator is defined.*

Definition 3.17. *Given a symmetric monoidal string diagram* S *on* Γ*, a* ***sheet diagram*** D *for* S *is a collection of topological manifolds with boundaries in* $\mathbb{R}^3$*:*

- *for each vertex* $v \in \mathbb{R}^2$ *of* S*, there is a* ***seam*** $\{v\} \times [0,1] \subset \mathbb{R}^3$*. Let* $[f_1, \ldots, f_n]$ *be the morphism generators associated with* v*. For each* i *we pick an* $x_{vi} \in (0,1)$ *such that* $x_{v1} < \cdots < x_{vn}$ *and add a* ***node*** (v, x_{vi})*, which is included in the vertex's seam;*

- *for each edge* $e \subset \mathbb{R}^2$ *of* S*, there is a* ***sheet*** $e \times [0,1] \subset \mathbb{R}^3$*. Let* $p_e : [0,1] \to \mathbb{R}^2$ *be a parametrization of* e *from source to target (top down). Let* $[A_{e1}, \ldots, A_{en}]$ *be the type associated to* e *in* S*. For each* i *we pick a parametrized segment* $\gamma_{ei} : [0,1] \to [0,1]$ *which gives rise to a* ***wire*** $w_{ei} : t \in [0,1] \to p_e(t) \times \gamma_{ei}(t)$ *included in the sheet for edge* e*. We require that for all* $t \in (0,1)$ *and* $i < j$*,* $\gamma_{ei}(t) < \gamma_{ej}(t)$*.*

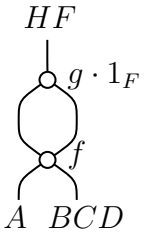
Furthermore, we require the following conditions:

1. *In* S*, if an edge* e *connects to a vertex* v *from above (into its domain), then for all wires* w_{ei} *on the sheet corresponding to* e *in* D*,* $\gamma_{ei}(1) = x_{vj}$ *where* j *is the origin of* A_{ei} *in the domain of* v*;*

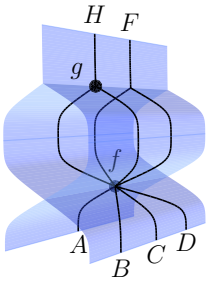
2. *In S, if an edge e connects to a vertex v from below (out of its codomain), then for all wires w_{ei} on the sheet corresponding to e in D, $\gamma_{ei}(0) = x_{vj}$ where j is the origin of A_{ei} in the codomain of v.*

*The **skeleton** of D is S. The set of sheet diagrams on Γ is denoted by $D(\Gamma)$.*

Let us consider an example of a sheet diagram.



(a) A string diagram S on Γ.



(b) A sheet diagram based on S.

Obtaining a sheet diagram from a string diagram only requires picking node positions on each seam, and trajectories of the wires on each sheet, such that the two boundary conditions are satisfied. The geometry of the seams and sheets is directly inherited from that of the string diagram itself. The boundary conditions ensure proper typing of the diagram. Geometrically, there is a projection from a sheet diagram to its skeleton.

Note that nothing interesting happens on sheets themselves: wires flow vertically, without being able to cross, from the bottom seam to the top seam of the sheet (or the diagram boundaries). However, for seams which have only one input sheet and one output sheet, we will use the convention of not drawing the seam between the two sheets, informally treating them as the same sheet. This makes it possible to draw monoidal string diagrams for the multiplicative product "on the sheets". In the following example we mark the seams with dashed red lines, which will be omitted in the future:

(a) Skeleton **(b)** Sheet diagram

Definition 3.18. *Let D be a sheet diagram. Its* ***domain*** $\operatorname{dom} D$ *is the domain of its skeleton, and similarly for its* ***codomain*** $\operatorname{cod} D$*.*

Definition 3.19. *Let D_1 and D_2 be sheet diagrams such that $\operatorname{cod} D_1 = \operatorname{dom} D_2$. The sheet diagram $D_2 \circ D_1$ is constructed by stacking D_1 on top of D_2, binding sheets and wires at the boundary with linear interpolation.*

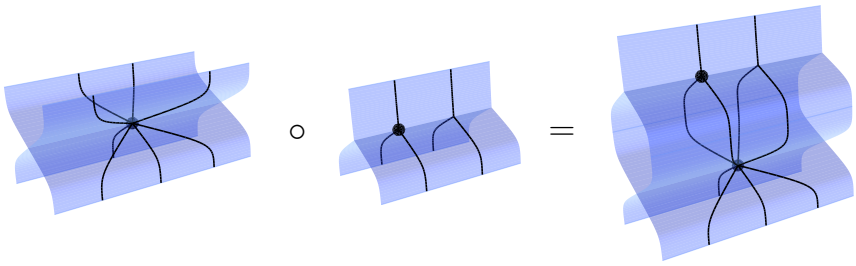


Figure 3.4: Example of composing sheet diagrams with $\circ$.

Definition 3.20. *Let D_1 and D_2 be sheet diagrams. The sheet diagram $D_1 \oplus D_2$ is constructed by adjoining D_2 to the right of D_1.*

Note that we have $S(D_2 \circ D_1) = S(D_2) \circ S(D_1)$ and $S(D_1 \oplus D_2) = S(D_1) \oplus S(D_2)$.

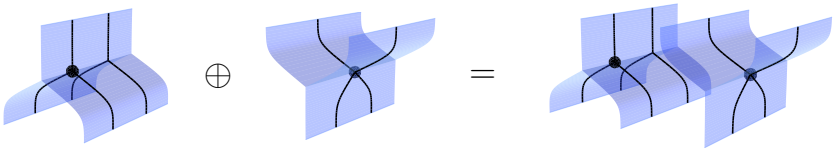


Figure 3.5: Example of composing sheet diagrams with $\oplus$.

To obtain a bimonoidal category of sheet diagrams, the last binary operation we need to define is the tensor product. This is more intricate since it is not naturally represented by the structure of the skeleton. We start by defining the

whiskering of a diagram with an object, in other words a tensor product where one of the factors is an identity.

Definition 3.21. *Let $A \in \operatorname{Ob}\Sigma$ and D be a sheet diagram. The **left whiskering** of D by A, denoted by $A \cdot D$, is obtained from D by:*

- *adding a node n_v on each seam corresponding to vertex v in $S(D)$, before all other nodes on the seam;*
- *adding a wire w_e on each sheet corresponding to edge e in $S(D)$, before all other wires on the seam;*
- *replacing all symmetries $\gamma_{U,V}$ in $S(D)$ by the whiskered symmetry $\gamma_{AU,AV}$;*

*such that if sheet e connects to seam v, wire w_e connects to node n_v. We have $A \cdot D : N(A \cdot \operatorname{dom} D) \to N(A \cdot \operatorname{cod} D)$. The **right whiskering** is defined similarly, placing the new nodes and wires after the existing ones instead.*

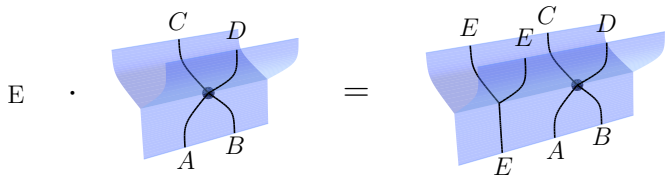


Figure 3.6: Example of the left whiskering of a sheet diagram.

We now turn to the general definition of the tensor product. We first need to define a family of isomorphisms to reorder summands.

Definition 3.22. *Let $p, q \in \mathbb{N}$ and $X_1, \ldots, X_p, Y_1, \ldots, Y_q \in \operatorname{Ob}\Gamma$. We define an isomorphism*

$$E_{XY} : \bigoplus_{i=1}^{p} \bigoplus_{j=1}^{q} X_i Y_j \to \bigoplus_{j=1}^{q} \bigoplus_{i=1}^{p} X_i Y_j$$

which reorders the summands as indicated by the commutation of sums, by repeated application of the symmetry for $\oplus$.

The definition of the tensor product of arbitrary diagrams exploits the exchange law to express it as a composition of whiskered diagrams.

Definition 3.23. *Let $f : \bigoplus_i A_i \to \bigoplus_j B_j$ and $g : \bigoplus_k C_k \to \bigoplus_l D_l$ be sheet diagrams, where $A_i, B_j, C_k, D_l \in \operatorname{Ob}\Gamma$. We can define the tensor product of D_1 and D_2 as:*

$$f \otimes g \coloneqq E_{BD}^{-1} \circ (\bigoplus_l f D_l) \circ E_{AD} \circ (\bigoplus_i A_i g)$$
$$: \bigoplus_i \bigoplus_k A_i C_k \to \bigoplus_j \bigoplus_l B_j D_l$$

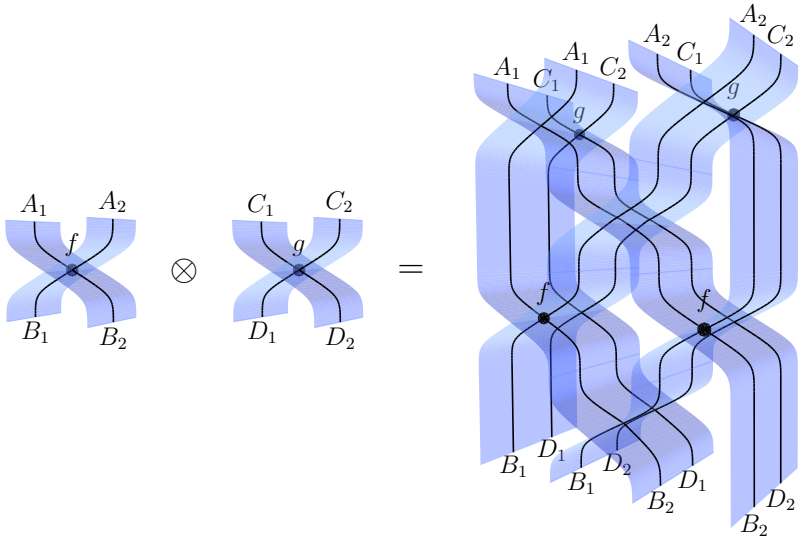


Figure 3.7: Example of the tensoring sheet diagrams.

Note that we made a choice here: $f \otimes g$ could have equivalently been defined as $(\bigoplus_j B_j g) \circ E_{BC}^{-1} \circ (\bigoplus_k f C_k) \circ E_{AC}$. In the next section, we will define a class of isotopies which will let us relate the two expressions (Lemma 3.38). Before that, we introduce one last product, that of morphism generators, for which we do not need to resort to whiskering.

Definition 3.24. *Let $f : \bigoplus_k A_k \to \bigoplus_l B_l$ and $g : \bigoplus_p C_p \to \bigoplus_q D_q \in \operatorname{Mor}\Gamma$. They are both lists of morphism generators in Σ or identities: $[f_1, \ldots, f_n]$ and $[g_1, \ldots, g_m]$. We define their tensor as*

$$[f_1, \ldots, f_n, g_1, \ldots, g_m]$$

In other words we concatenate the lists of generators and identities that compose them.

Lemma 3.25. *For all* $f, g \in \operatorname{Mor}\Gamma$, $\operatorname{dom} fg = N((\operatorname{dom} f)(\operatorname{dom} g))$ *and* $\operatorname{cod} fg = N((\operatorname{cod} f)(\operatorname{cod} g))$.

Proof. This is a direct consequence of the associativity of N (Lemma 3.14) and the definition of domains and codomain in Γ (Definition 3.15). □

3.3.3 Isomorphisms of sheet diagrams

We begin with a lemma on isomorphisms of symmetric monoidal string diagrams.

Lemma 3.26. *Given an anchored isotopy of progressive diagrams between diagrams* S_1 *and* S_2*, there is an open graph isomorphism* ϕ *between the open graphs defined by* S_1 *and* S_2*. In other words there are bijections between their sets of nodes, their sets of edges, and those respect the adjacency relations.*

Definition 3.27. *Given sheet diagrams* D_1, D_2 *with* $\operatorname{dom} D_1 = \operatorname{dom} D_2$ *and* $\operatorname{cod} D_1 = \operatorname{cod} D_2$*, a* ***regular isotopy of sheet diagrams*** *from* D_1 *to* D_2 *is given by an anchored isotopy of progressive polarised diagrams* $\alpha : S(D_1) \to S(D_2)$*, which gives an open graph isomorphism* ϕ *by Lemma 3.26, as well as:*

- *for each node* n_{vi} *on a seam* v *in* D_1*, a continuous map* $x^*_{vi} : [0,1] \to [0,1]$ *such that* $x^*_{vi}(0) = x_{vi}$ *and* $x^*_{vi}(1) = x_{\phi(v)i}$*, where* $n_{\phi(v)i} \in D_2$*;*
- *for each wire* w_{ei} *on a sheet* e *in* D_1*, a continuous isotopy* $\gamma^*_{ei} : [0,1] \times [0,1] \to [0,1]$ *such that* $\gamma^*_{ei}(0,t) = \gamma_{ei}(t)$ *and* $\gamma^*_{ei}(1,t) = \gamma_{\phi(e)i}(t)$ *for all* $t \in [0,1]$*, where* $w_{\phi(e)i} \in D_2$

Finally, at each time $t \in [0,1]$*, the sheet diagram made of the skeleton* $\alpha(t)$*, the node positions* $x^*_{vi}(t)$ *and the wire paths* $\gamma^*_{ei}(t)$ *is required to be a valid sheet diagram.*

Lemma 3.28. *Regular isotopy of sheet diagrams preserve their interpretations as bimonoidal morphisms.*

Proof. As the interpretation of a sheet diagram D is the interpretation of its skeleton $S(D)$, this is a simple consequence of Theorem 2.19. □

However, regular isotopy of sheet diagrams does not capture the entire equational theory of bimonoidal categories: the exchange law for the multiplicative monoidal structure is missing. For instance, the following diagrams have the same interpretation, but they are not regularly isotopic:

Their interpretations are equal by the exchange rule for the multiplicative product:

$$\begin{aligned}(1_B \cdot f) \circ ((g \cdot 1_C) \oplus (g \cdot 1_D)) &= (1_B \cdot f) \circ (g \cdot (1_{C \oplus D})) \\ &= (g \cdot 1_E) \circ (1_A \cdot f)\end{aligned}$$

Therefore, we need to broaden our class of isomorphisms to capture multiplicative exchange too.

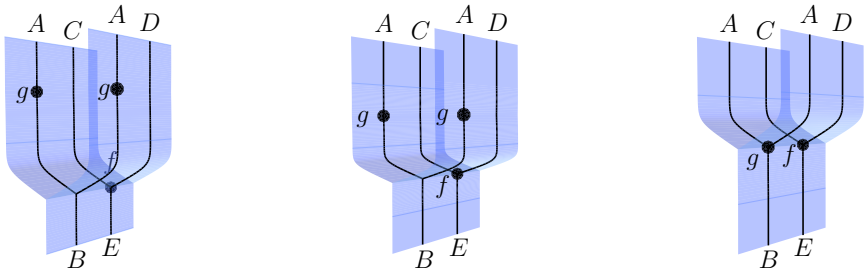
Definition 3.29. *Let $f : \bigoplus_i A_i \to \bigoplus_j B_j$ and $g : \bigoplus_k C_k \to \bigoplus_l D_l$ be morphism generators in Γ (diagrams with a single seam), where $A_i, B_j, C_k, D_l \in \operatorname{Ob}\Gamma$. A **tensor merge** from $\alpha = E_{BD}^{-1} \circ (\bigoplus_l f D_l) \circ E_{AD} \circ (\bigoplus_i A_i g)$ to $\beta = fg$ is a function $\gamma : [0,1] \to D(\Sigma)$ (where $D(\Sigma)$ is the set of sheet diagrams on Σ) such that:*

- *$\gamma(0) = \alpha$ and $\gamma(1) = \beta$;*
- *for all $0 < t < 1$, the restriction of γ on $[0, t)$ is a regular isotopy of sheet diagrams*
- *for each seam $s \in \alpha$, $\lim_{t \to 1} s(t)$ is the unique seam of β;*
- *for each sheet $s \in \alpha$ with one end on the boundary of the diagram, $\lim_{t \to 1} s(t)$ is the unique sheet in β connected to the same boundary at the same ordinal position;*

- *for each sheet $s \in \alpha$ not connected to the boundary, $\lim_{t\to 1} s(t)$ is the unique seam of β;*
- *for each node $n \in \alpha$ on a seam s, $\lim_{t\to 1} n(t)$ is a node in the unique seam of β, with the same ordinal position;*
- *for each wire w on a sheet $s \in \alpha$ that connects to the boundary, $\lim_{t\to 1} w(t)$ is a wire on $\lim_{t\to 1} s(t)$ with the same ordinal position.*

*Similarly, one can define a tensor merge from $(\bigoplus_j B_j g) \circ E_{BC}^{-1} \circ (\bigoplus_k f C_k) \circ E_{AC}$ to fg. Finally, a **tensor explosion** $\gamma : \alpha \to \beta$ is a tensor merge in reverse, i.e. when $t \mapsto \gamma(1-t)$ is a tensor merge.*

For example, the following are steps of a tensor merge:



(a) $\gamma(0)$ **(b)** $\gamma(\frac{1}{2})$ **(c)** $\gamma(1)$

We can extend the notions of tensor merges and tensor explosions to wider contexts, where the seams to merge or explode are parts of a larger diagram. To that end, we use the notion of *sheet diagram with a hole*, which is simply a sheet diagram where an occurrence of another sheet diagram has been removed. We denote by $C(x)$ such a diagram, where x is a free variable, and by $C(\alpha)$ the sheet diagram obtained by inserting the sheet diagram α in place of the hole.

Definition 3.30. *Let $\gamma : \alpha \to \beta$ be a tensor merge and $C(x)$ be a sheet diagram with a hole, such that $C(\alpha)$ is a valid sheet diagram. Since α and β have the same domain and codomain, $C(\beta)$ is also a valid sheet diagram. The function $C(\gamma) : [0,1] \to C(\Sigma)$ defined by $C(\gamma) : t \mapsto C(\gamma(t))$ is called a **tensor merge in context**. Similarly, we define tensor explosions in context.*

Lemma 3.31. *For all tensor merges or explosions in context $C(\gamma) : C(\alpha) \to C(\beta)$, $C(\alpha)$ and $C(\beta)$ are equal as bimonoidal morphisms.*

Proof. By Definition 3.29, the start and end diagrams of tensor merges or explosions are equal by multiplicative exchange. By composition, this extends to contexts. □

We can now define our most general notion of isotopy for sheet diagrams.

Definition 3.32. *A **bimonoidal isotopy** between sheet diagrams D_1, D_2 is a function $\gamma : [0,1] \to D(\Sigma)$ such that $\gamma(0) = D_1$, $\gamma(1) = D_2$ and for all $t \in [0,1]$, there exists $\varepsilon > 0$ such that on $[t-\varepsilon, t]$, γ is either a regular isotopy or a tensor merge, and on $[t, t+\varepsilon]$, γ is either a regular isotopy or a tensor explosion.*

Lemma 3.33. *Bimonoidal isotopy preserves the interpretation of diagrams.*

Proof. Since $[0,1]$ is connected, it is enough to show that the interpretation of $\gamma(t)$ is locally constant for all $t \in [0,1]$. By Lemma 3.28, the interpretation is constant during regular isotopies and by Lemma 3.31, tensor merges and explosions in context also preserve interpretation. □

Lemma 3.34. *Composition, sum and tensor of sheet diagrams all respect bimonoidal isotopy.*

Proof. Given two sheet diagrams α, β and bimonoidal isotopies $\gamma : \alpha \to \alpha'$, $\gamma' : \beta \to \beta'$, we obtain a bimonoidal isotopy from $\alpha \oplus \beta$ to $\alpha' \oplus \beta'$ by first running γ while β stays still, then running γ' while α' stays still. Note that we do not run both transformations in parallel because our definition of bimonoidal isotopy only allows for one tensor merge or explosion at a time. The case for the composition of diagrams is similar. By Definition 3.23, the diagram $\alpha \otimes \beta$ contains in general multiple copies of α and β: we obtain an isotopy by running γ on each copy of α in sequence, and then γ' on copies of β. □

Definition 3.35. *The **category of sheet diagrams** $D(\Sigma)$ has sums of products of object symbols from Σ as objects, and equivalence classes of sheet diagrams under sheet diagram isotopy as morphisms. Domains and codomains are given*

by Definition 3.18, composition by Definition 3.19. It has a symmetric monoidal structure $\oplus$ given by Definition 3.20.

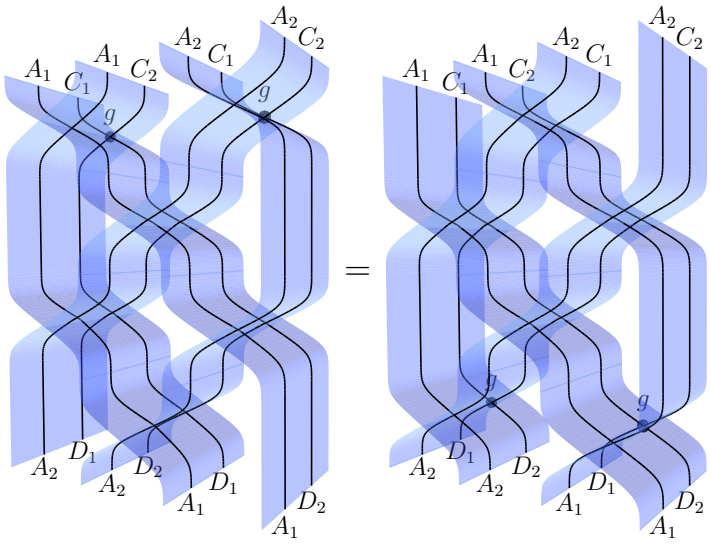
To equip $D(\Sigma)$ with a multiplicative monoidal structure, we need to show that our tensor product (Definition 3.23) satisfies the exchange law. Tensor merges and explosions are only defined for morphism generators in Γ (single seams), not arbitrary diagrams, so we cannot just use one tensor merge followed by one tensor explosion in general.

Lemma 3.36. *Any diagram $f \in D(\Sigma)$ can be written in general position, such that no two seams or additive symmetries are at the same height, up to bimonoidal isotopy. It can therefore be expressed as a sequential composition of **slices**, which are sums of at most one seam or additive symmetry and a finite number of identities.*

Proof. This is a straightforward generalization of the same result for symmetric monoidal string diagrams, which can be found in Joyal and Street (1991). □

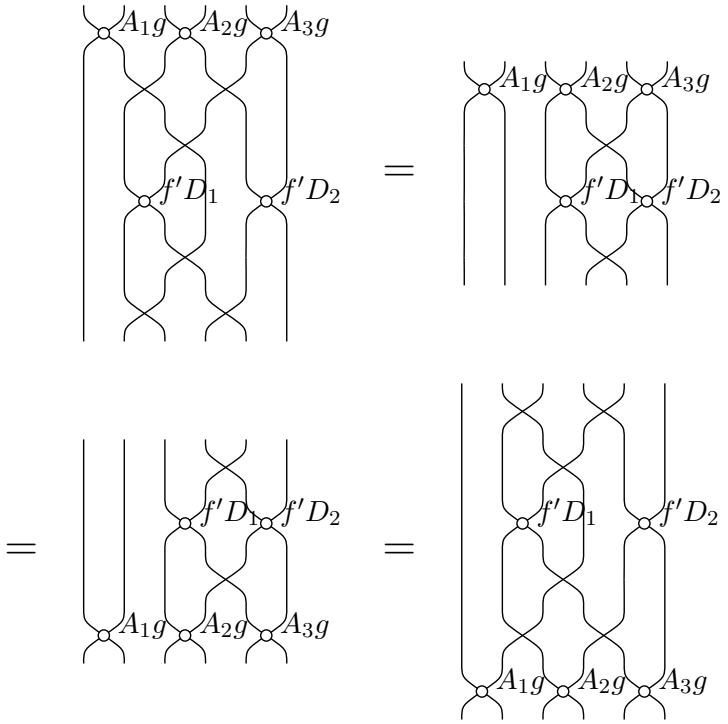
Lemma 3.37. *Let $f : \bigoplus_i A_i \to \bigoplus_j B_j$ and $g : \bigoplus_k C_k \to \bigoplus_l D_l$ be slices, where $A_i, B_j, C_k, D_l \in \operatorname{Ob}\Gamma$. Then there is a bimonoidal isotopy between $E_{BD}^{-1} \circ (\bigoplus_l f D_l) \circ E_{AD} \circ (\bigoplus_i A_i g)$ and $(\bigoplus_j B_j g) \circ E_{BC}^{-1} \circ (\bigoplus_k f C_k) \circ E_{AC}$.*

Proof. We proceed by induction on the sum of numbers of identities in the summands of f and g. When there are no identities in f or g, there are three cases. If both f and g are seams, then the two expressions are isotopic via a tensor merge and explosion by construction. If both f and g are symmetries, then the two expressions only consist of symmetries and identities which induce the same permutation of the summands of their domain, so they are isotopic. Finally, if one of f and g is a seam and the other is a symmetry, let us assume by symmetry that $f = \gamma_{A_1,A_2}$ and g is a seam. The isotopy holds by pull through moves:



Notice that in this transformation nothing interesting is happening on the third dimension: in the future, we will resort to two-dimensional string diagrams for such isotopies.

Now for the inductive case, assume there is an identity in $f = 1_{A_1} \oplus f'$. The isotopy holds as follows:



The second equality uses the induction hypothesis on f' and g, other steps are

regular isotopies. □

Lemma 3.38. *Let $f : \bigoplus_i A_i \to \bigoplus_j B_j$ and $g : \bigoplus_k C_k \to \bigoplus_l D_l$ be sheet diagrams, where $A_i, B_j, C_k, D_l \in \operatorname{Ob}\Gamma$. Then there is a bimonoidal isotopy between $E_{BD}^{-1} \circ (\bigoplus_l f D_l) \circ E_{AD} \circ (\bigoplus_i A_i g)$ and $(\bigoplus_j B_j g) \circ E_{BC}^{-1} \circ (\bigoplus_k f C_k) \circ E_{AC}^{-1}$.*

Proof. Up to a regular isotopy, we can assume that f and g are in general position and therefore expressed as a sequential composition of slices. We can then apply Lemma 3.37 repeatedly, exchanging neighbouring slices of f and g until all slices of f are below all slices of g. □

Lemma 3.39. *$D(\Sigma)$ can be equipped with a monoidal structure $(D(\Sigma), \otimes, I)$, given on objects by $A \otimes B = N(A \cdot B)$ and on morphisms by Definition 3.23*

Proof. The product on objects is unital ($N(A \cdot I) = N(I \cdot A) = N(A)$) and associative by Lemma 3.14. By Lemma 3.38, the exchange law for $\otimes$ is satisfied, hence the result. □

Lemma 3.40. *$D(\Sigma)$ is bimonoidal with $(\otimes, I)$ distributing over $(\oplus, O)$.*

Proof. The monoidal structure $(D(\Sigma), \oplus, O)$ is given by Theorem 2.19, and the monoidal structure $(D(\Sigma), \otimes, I)$ is given by Lemma 3.39.

Since $N((A \oplus B)C) = N(AC) \oplus N(BC)$, we can define:

$$\delta^{\#}_{A,B,C} : (A \oplus B) \otimes C \to A \otimes B \oplus B \otimes C = 1_{N(AC) \oplus N(BC)}$$

For $\delta_{A,B,C} : A \otimes (B \oplus C) \to A \otimes B \oplus A \otimes C$, decompose $N(A) = \bigoplus_i A_i$ with $A_i \in \operatorname{Ob}\Gamma$. We have

$$N(A(B \oplus C)) = \bigoplus_i N(A_i(B \oplus C)) = \bigoplus_i N(A_iB) \oplus N(A_iC)$$

$$N(AC) \oplus N(BC) = \bigoplus_i N(A_iB) \oplus \bigoplus_i N(A_iC)$$

Therefore we define $\delta_{A,B,C}$ as the reordering map (composition of symmetries) from $\bigoplus_i N(A_iB) \oplus N(A_iC)$ to $\bigoplus_i N(A_iB) \oplus \bigoplus_i N(A_iC)$.

Since $N(OA) = N(AO) = O$ for all A, we define $\lambda^* : O \otimes A \to O$ and $\rho^*_A : AO \to O$ as 1_O (the empty sheet diagram).

We now need to check the coherence axioms of Appendix A. Let us consider the first axiom:

$$\begin{array}{ccc} A(B \oplus C) & \xrightarrow{\delta_{A,B,C}} & AB \oplus AC \\ {\scriptstyle 1_A \gamma'_{B,C}} \downarrow & \text{(I)} & \downarrow {\scriptstyle \gamma'_{AB,AC}} \\ A(C \oplus B) & \xrightarrow[\delta_{A,C,B}]{} & AC \oplus AB \end{array}$$

In $D(\Sigma)$, all sides of this square are composites of the additive symmetry γ. Therefore, it is sufficient to check that both paths induce the same permutation, by coherence for symmetric monoidal categories. Equivalently, one can also use coherence for regular objects of bimonoidal categories (Theorem 3.13) by choosing A, B and C as sums of generators all distinct. One can then obtain commutation for the general case by instantiation (substituting the generators by the actual summands). The other axioms can be treated in similar ways:

- (I), (V), (VI), (VIII), (IX) hold by bimonoidal coherence for regular objects;
- (III) holds since $\delta^{\#} = 1$ and $\gamma_{A,B} 1_C = \gamma'_{AC,BC}$ by definition;
- (IV) simplifies thanks to $\delta^{\#} = 1$ and holds by monoidal coherence for $(\oplus, O)$
- (VII) simplifies thanks to $\delta^{\#} = 1$ and holds by monoidal coherence for $(\otimes, I)$
- (X), (XII), (XIII), (XIV), (XVI), (XVII), (XVIII) hold as all sides equal 1_O
- (XIX), (XX), (XXI), (XXII) hold as all sides are identities
- (XXIII) and (XXIV) hold as $\delta_{I,A,B} = 1_{A \oplus B} = \delta^{\#}_{A,B,I}$

□

Theorem 3.41. *$D(\Sigma)$ and $\overline{\Sigma}$ are bimonoidally equivalent, i.e. $D(\Sigma)$ is the free bimonoidal category on Σ.*

Proof. The interpretation of diagrams is a well-defined function $[\cdot] : D(\Sigma) \to \overline{\Sigma}$ by Lemma 3.33 and is a bimonoidal functor by construction. For the reverse direction, by freeness of $\overline{\Sigma}$ there is a unique bimonoidal functor $F : \overline{\Sigma} \to D(\Sigma)$ mapping each generator in Σ to its representation in $D(\Sigma)$.

$$\overline{\Sigma} \underset{[\cdot]}{\overset{F}{\rightleftarrows}} D(\Sigma)$$

Let us show that these form an equivalence. First, $F \circ [\cdot]$ is the identity on objects and morphism generators and therefore by induction it is the identity on all morphisms. Second, $[\cdot] \circ F$ is not the identity but $n_A : A \to N(A)$ is a natural isomorphism from the identity to $[\cdot] \circ F$. Its naturality can be shown by induction on f:

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ {\scriptstyle n_A}\downarrow & & \downarrow{\scriptstyle n_B} \\ N(A) & \xrightarrow[F(f)]{} & N(B) \end{array}$$

For f a generator, the vertical sides are identities (by assumption that domains and codomains of generators are normalized, from Section 3.3.1). For f a structural isomorphism, the square commutes by regular coherence. By induction, it holds for all morphisms. □

3.3.4 Data structures for sheet diagrams

In this section we give a short primer on the declarative format used to represent diagrams in SheetShow[2], the tool we used to render all sheet diagrams in this chapter.

Sheet diagrams in bimonoidal categories are obtained by extruding symmetric monoidal string diagrams for the additive monoidal structure $(\mathcal{C}, \oplus, O)$. Therefore, our data structure for bimonoidal diagrams is based on that for monoidal diagrams, laid out in Section 2.2.3.

A bimonoidal diagram is described by:

[2] Available at https://wetneb.github.io/sheetshow/ and https://github.com/wetneb/sheetshow (source)

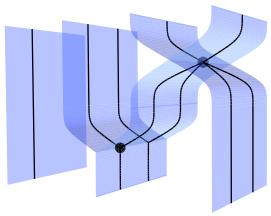
- The number of input sheets, and the number of input wires on each of these input sheets;
- The slices of the bimonoidal diagram, which are seams between sheets. They are each described by:
  - The number of sheets passing to the left of the seam. We call this, again, the **offset**;
  - The number of input sheets joined by the seam;
  - The number of output sheets produced by the seam;
  - The nodes present on the seam.

Each seam can have multiple nodes on it. Each of these can connect to some wires on each input sheet (not necessarily the same number of wires for each input sheet) and similarly for output sheets. We describe them with the following data:

- The number of wires passing through the seam without touching a node, to the left of the node being described. We call this the **offset** of the node;
- For each input sheet, the number of wires connected to the node;
- For each output sheet, the number of wires connected to the node.

For instance:

```
inputs: [ 1, 2, 1, 1 ]
slices:
- offset: 2
  inputs: 2
  outputs: 2
  nodes:
  - offset: 0
    inputs: [ 1, 1 ]
    outputs: [ 2, 2 ]
- offset: 1
  inputs: 1
  outputs: 2
  nodes:
  - offset: 0
    inputs: [ 1, 1 ]
    outputs: [ 1 ]
```



Again, additional metadata can be added on the geometry to annotate it with labels, types, and represent symmetries for the additive and multiplicative structures. For more details about these, consult SheetShow's documentation: https://sheetshow.readthedocs.org/en/latest/.

3.4 Baez's conjecture

Recently, a conjecture attributed to Baez was confirmed by Elgueta (2020), who showed that the groupoid of finite sets and bijections is bi-initial in the 2-category of bimonoidal categories. The category of finite sets has indeed a bimonoidal structure, where disjoint union of sets is the monoidal addition and cartesian product is the monoidal multiplication.

This result can also be obtained via string diagrams. Indeed, the free bimonoidal category on an empty signature, $\overline{\emptyset}$, is bi-initial. This is a direct consequence of the universal property: any bimonoidal functor from $\overline{\emptyset}$ to a bimonoidal category $\mathcal{C}$ is determined (up to equivalence) by the image of the generators of $\overline{\emptyset}$, but there are no such generators.

Therefore, to prove Baez' conjecture it is enough to characterize the free bimonoidal category on an empty signature. By Theorem 3.41, $\overline{\emptyset}$ is bimonoidally equivalent to $D(\emptyset)$. The objects of $D(\emptyset)$ are finite sums of the multiplicative monoidal unit. The morphisms of $D(\emptyset)$ are string diagrams on the empty signature. We can analyze the geometry of such string diagrams. All the sheets in such diagrams are empty: they cannot have any wires on them, since those wires would need to be annotated by an object from the signature. Similarly, the diagrams do not contain any seams, since each seam contains at least one node which is annotated by a morphism generator. Therefore, the diagrams are only made of identities of empty sheets and additive symmetries.

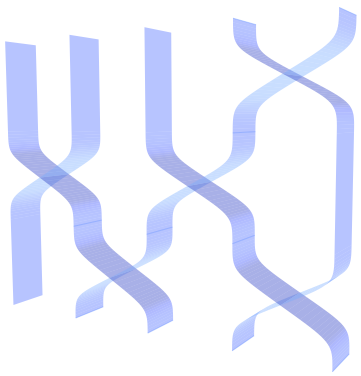


Figure 3.11: A sheet diagram on the empty signature.

Hence, a morphism in $D(\emptyset)$ induces a permutation of its domain, which is equal to its codomain. Furthermore, morphisms in $D(\emptyset)$ are equivalence classes of string diagrams up to bimonoidal equivalence. Two string diagrams induce the same permutation of their common domain when their skeletons are isomorphic as symmetric monoidal diagrams, and therefore when the diagrams are in bimonoidal equivalence. Therefore, the category $D(\emptyset)$ is equivalent to the groupoid of finite sets, hence the result.

3.5 Applications to dataflow programs

In this section, we give one application of bimonoidal categories in the domain of data processing. This is the application that motivated our work on sheet diagrams, as we first used those diagrams informally to data transformation pipelines. Therefore, this section follows the narrative of (Delpeuch, 2019), but it has been reformulated to use bimonoidal categories and the sheet diagrams we have just developed.

In the dataflow paradigm, data processing pipelines are built out of modular components which communicate via some channels. This is a natural architecture to build concurrent programs and has been studied in many variants, such as Kahn process networks (Kahn, 1974), Petri nets (Petri, 1966; Kavi et al., 1987), the LUSTRE language (Halbwachs et al., 1991) or even UNIX processes and pipes (Walker et al., 2009). Each of these variants comes with its own requirements on the precise nature of these channels and operations: for instance, sorting a

stream requires the module to read the entire stream before writing the first value on its output stream, which violates a requirement called *monotonicity* in Kahn process networks, but is possible in UNIX. Categorical accounts of these process theories have been developed, for instance for Kahn process networks (Stark, 1991; Hildebrandt et al., 2004) or Petri nets (Pratt, 1991; Meseguer et al., 1992).

In this section, we give categorical semantics to programs in Extract-Transform-Load (ETL) software. These three words refer to the three main steps of most projects carried out with this sort of system. Typically, the user extracts data from an existing data source such as a comma-separated values (CSV) file, transforms it to match a desired schema (for instance by normalizing values, removing faulty records, or joining them with other data sources), and loads it into a more structured information system such as a relational or graph database. In other words, ETL tools let users move data from one data model to another. Because the original data source is typically less structured and not as well curated as the target data store, these operations are also refered to as data cleansing or wrangling.

ETL tools typically let users manipulate their data via a collection of operations which can be configured and composed. The way operations can be composed, as well as the format of the data they act on, represent the main design choice for these tools: it will determine what sort of workflow they can represent naturally and efficiently. We will focus here on the tabular data model popularized the OpenRefine software (Huynh et al., 2019), a widely used open source tool popular in the linked open data and data journalism communities.[3] We give a self-contained description of the tool in Section 3.5.2.

We propose a complete categorical axiomatization for this data model, using a bimonoidal category. Using sheet diagrams, this gives rise to a three-dimensional diagrammatic language for the workflows, generalizing the widespread graph-based representation of dataflow pipelines. The semantics and the complete axiomatization provided make it possible to use this model to reason about workflow equivalence using intuitive graphical rules.

[3] See http://openrefine.org/, we encourage viewing the videos or trying the software directly, although this section should be readable with no previous knowledge of the tool.

This has very concrete applications: at the time of writing, OpenRefine has a very limited interface to manipulate workflows, where the various operations used in the transformation are combined in a simple list. Graph-based representations of workflows are already popular in similar tools but are not expressive enough to capture OpenRefine's model, due to the use of facets, which dynamically change the route followed by data records in the processing pipeline depending on their values. Our approach solves this problem by giving a natural graphical representation which can be understood with no knowledge of category theory, making it amenable to implementation in the tool itself.

3.5.1 Categorical semantics of dataflow

Symmetric monoidal categories model an elementary sort of dataflow pipelines, where the flow is acyclic and deterministic. This is well known in the applied category theory community: for instance, Coecke (2010) illustrates it by modelling cooking recipes by morphisms in such categories.

Informally, in a symmetric monoidal category $\mathcal{C}$, objects of $\mathcal{C}$ are stream types and morphisms are dataflow pipelines binding input streams to output streams. Pipelines can be composed sequentially, binding the outputs of the first pipeline to the second, or in parallel, obtaining a pipeline from both inputs to both outputs. The difference between food and data is that discarding the latter is not frowned upon: data streams can be discarded and copied, which makes the category cartesian.

3.5.2 Overview of OpenRefine

Let us now get into more detail about how OpenRefine works. Loading a data source into OpenRefine creates a *project*, which consists of a simple data table: it is a collection of rows and columns. To each row and column, a value (possibly null) is associated.

The user can then apply *operations* on this table. Applying an operation will change the state of the table, usually by performing the same transformation for each row in the table. Example of operations include removing a column, reordering

Family name	Given name	Donation
Green	Amanda	25€
Dawson	Rupert	12€
de Boer	John	40€
Tusk	Maria	3€

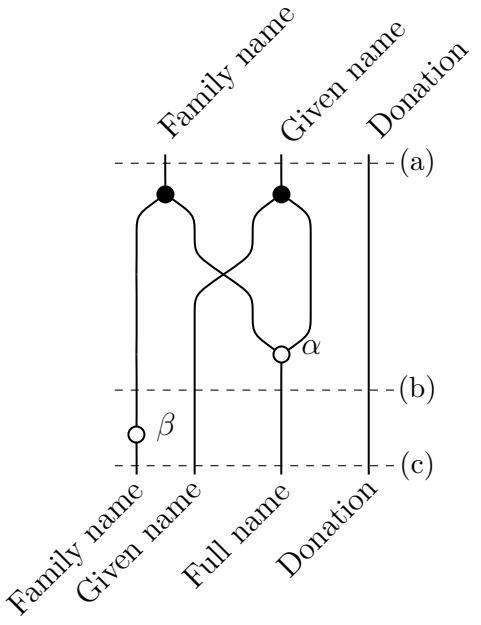
(a) The initial state of the project.

Family name	Given name	Full name	Donation
Green	Amanda	Amanda Green	25€
Dawson	Rupert	Rupert Dawson	12€
de Boer	John	John de Boer	40€
Tusk	Maria	Maria Tusk	3€

(b) Applying an operation to create the `Full name` column.

Family name	Given name	Full name	Donation
GREEN	Amanda	Amanda Green	25€
DAWSON	Rupert	Rupert Dawson	12€
DE BOER	John	John de Boer	40€
TUSK	Maria	Maria Tusk	3€

(c) Applying an operation to capitalize the `Family name` column.



(d) The workflow represented as a string diagram. Operation α is concatenation, operation β is capitalization.

Figure 3.12: Example of an OpenRefine project in its successive states, with the corresponding string diagram.

columns, normalizing the case of strings in a column or creating a new column whose values are obtained by concatenating the values in other columns. Users can configure these operations with the help of an expression language which lets them derive the values of a new column from the values in existing columns.

Unlike spreadsheet software, such expressions are fully evaluated when stored in the cells that they define: at each stage of the transformation process, the values in the table are static and will not be updated further if the values they were derived from change in the future. For instance, in the sample project of Figure 3.12, the first operation creates a `Full name` column by concatenating the `Given name` and `Family name` columns. Applying a second operation to capitalize the `Family name` column does not change the values in the `Full name` column.

Another difference with spreadsheet software, where it is possible to reference any cell in the expression defining a cell, is that OpenRefine's expression language only lets the user access values from the same row. For instance, in the same example project of Figure 3.12, spreadsheet software would make it easy to compute the sum of all donations in a final row. This is not possible in OpenRefine as this would amount to computing the value of a cell from the value of other cells outside its own row.

In other words, operations in OpenRefine are applied row-wise and are stateless: no state is retained between the processing of rows. It is therefore simple to parallelize these operations, as they amount to a pure *map* on the list of rows. This is a simplification: in reality, there are violations of these requirements (for instance, OpenRefine offers a sorting operation, and a *records mode* which introduces a restricted form of non-locality). Due to the limited space we do not review these violations here.

3.5.3 Elementary model of OpenRefine workflows

So far, OpenRefine fits neatly in the dataflow paradigm presented in Section 3.5.1. One can view each column of a project as a data stream, which can be assigned a type $t \in T$: in our example project, the first two columns are string-valued and the

third contains monetary values. These data streams are *synchronous*: the values they contain are aligned to form rows. An operation $\alpha \in O$ can be seen as reading values from some columns and writing new columns as output. Because of the synchronicity requirement, an operation really is just a function from tuples of input values on the columns it reads to values on the column it writes.

The schema of a table, which is the list of its column types, can be naturally represented by the product of the objects representing its column types. In the example of Figure 3.12, the initial table is therefore represented by $S \times S \times M$, where S is the type of strings and M of monetary values. Let us call $\alpha : S \times S \to S$ the first concatenation operation and $\beta : S \to S$ the second capitalization operation. Figure 3.12d shows a string diagram which models the workflow of Figure 3.12.

Definition 3.42. *The category $\mathcal{E}$ of table schema and elementary OpenRefine workflows between them is the free cartesian category generated by a set of datatypes D as objects and a set of operations O as morphisms.*

This modelling of OpenRefine workflows makes it easy to reason about the information flow in the project. It is possible to rearrange the operations using the axioms of a cartesian category to show that two workflows produce the same results. We could add some generating equations between composites of the generating operations, such as operations which commute even when executed on the same column for instance.

Without loss of generality, we can assume that the generating operations all have a single generating datatype as codomain, as the cartesian structure makes it possible to represent generic operations as composites of their projections. Under these conditions, morphisms of $\mathcal{E}$ can be rewritten to a normal form, illustrated in Figure 3.13.

Lemma 3.43. *Any morphism $m \in \mathcal{E}$ can be written as a vertical composite of three layers: the first one only contains copying and discarding morphisms, the second only symmetries and the third only generating operations (identities are allowed at each level).*

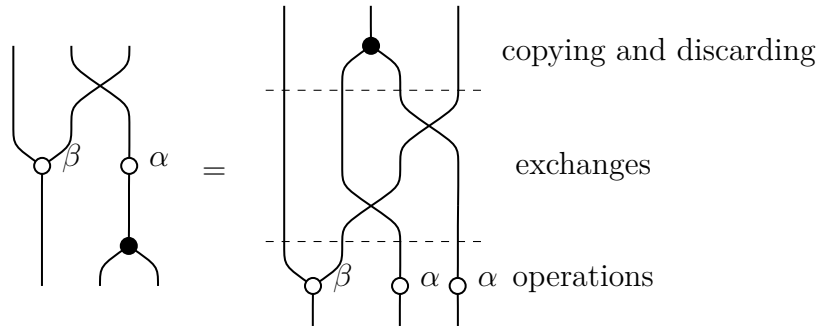


Figure 3.13: A diagram in $\mathcal{E}$ and its normal form.

All three slices in the decomposition above can be further normalized: for instance, the cartesian slice can be expressed in left-associative form, the exchange slice is determined by the permutation it represents, and the operation slice can be expressed in right normal form, as we will see in Section 4.2. This gives a simple way to decide the equality of diagrams in $\mathcal{E}$. Of course, deciding equality in a free cartesian category just amounts to comparing tuples of terms in universal algebra. We are only formulating it as a graphical rewriting procedure to lay down the methodology for the next section, where we develop a more elaborate model of OpenRefine workflows. For this model too, we will obtain a graphical method to decide the equality of diagrams, which is also be related to a term-based representation in the proof of Theorem 3.48.

3.5.4 Model of OpenRefine workflows with facets

One key functionality of OpenRefine that we have ignored so far is its *facets*. A facet on a column gives a summary of the value distribution in this column. For instance, a facet on a column containing strings will display the distinct strings occurring in the column and their number of occurences. A numerical facet will display a histogram, a scatterplot facet will display points in the plane, and so on.

Beyond the use of facets to analyze distributions of values, it is also possible to select particular values in the facet, which selects the rows where these values are found. It is then possible to run operations on these filtered rows only. So far

our operations ran on all rows indiscriminately, so we need to extend our model to represent operations applied to a filtered set of rows.

We assume from now on a set F of filters in addition to our set of operations O. Each filter $f \in F$ is associated with an object $T_f \in \mathcal{E}$, the type of data that it filters on. Each filter can be thought of as a boolean expression that can be evaluated for each value $v \in T_f$, determining if the value is included or excluded by the filter. The type T_f is not required to be atomic: for instance, in the case of a scatterplot filter, two numerical columns are read (one for each dimension of the scatterplot).

Definition 3.44. *Let $\mathcal{F}$ be the free co-cartesian category generated as follows. We denote by $[A_1, \ldots, A_n]$ the product of objects $A_1, \ldots, A_n$ in $\mathcal{F}$ to distinguish it from the product $\times$ in $\mathcal{E}$. For each object $T \in \mathcal{E}$, $[T] \in \mathcal{F}$ is a generator. Morphism generators are:*

(i) For each morphism $\alpha \in \mathcal{E}(T, U)$, there is a generator $[\alpha] : [T] \to [U]$.

(ii) For each filter f and object $U \in \mathcal{E}$, there is a generator $[f \times U] : [T_f \times U] \to [T_f \times U, T_f \times U]$.

For each object $T \in \mathcal{E}$, we call $J_T : [T, T] \to [T]$ and $E_T : [] \to [T]$ the comultiplication and counit provided by the co-cartesian structure.

The axioms satisfied by these generators are stated graphically in Figure 3.15, with the notations introduced in Figure 3.14. In addition to these axioms, we require that $[g] \circ [f] = [g \circ f]$ (which is tautological graphically). In other words, $\mathcal{E}$ embeds into $\mathcal{F}$ functorially (but that functor is not monoidal).

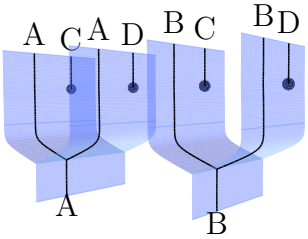
The definition above can be interpreted intuitively as follows. An object in $\mathcal{E}$ represents the schema of a table (the list of types of its columns). An object of $\mathcal{F}$ is a list of objects of $\mathcal{E}$, so it represents a list of table schemata. As will be made clear by the semantics defined in the next section, a morphism in $\mathcal{F} : [U, V] \mapsto [W, Z]$ should be thought of as a function mapping disjoint tables of respective schemata U and V to disjoint tables of respective schemata W or Z, and row-wise so: depending on its values, a row can end up in either of the output tables. This makes it therefore

possible to represent filters as morphisms triaging rows to disjoint tables. A filter $[f \times U]$ operates on tables of schema $T_f \times U$, and only reads values from the first component to determine whether to send the row to the first or second output table.[4] This treatment of a boolean predicate $A \to 2$ as a morphism $A \to A + A$ is similar to that of effectus theory (Cho et al., 2015). The comultiplication J_T is a union, merges two tables of identical schemata together.[5] The counit E_T is the empty table.

Proposition 3.45. *The category $\mathcal{F}$ is bimonoidal, with the additive monoidal structure being the co-cartesian structure of $\mathcal{F}$, and the multiplicative structure defined on objects by $[A_1, \dots, A_n] \times [B_1, \dots, B_m] = [A_1 \times B_1, \dots, A_1 \times B_m, \dots, A_n \times B_1, \dots, A_n \times B_m]$ and on morphisms as in Definition 3.23. Furthermore, it is cartesian too, making it a distributive category.*

Proof. Just like for sheet diagrams, we have defined the multiplicative product such that objects are always normalized, which is forced on us by the way we defined $\mathcal{F}$, by freely generating a co-cartesian structure on top of a cartesian structure. Therefore, the definition of the distributors is similar: the distributor $\delta[A] \times [B, C] \simeq [A \times B, A \times C]$ is the identity, and the distributor $\delta' : [A, B] \times [C] \simeq [A \times C, B \times C]$ consists of symmetries.

To show that $\mathcal{F}$ is cartesian, define the projection $\pi_1 : [A, B] \times [C, D] \to [A, B]$ by

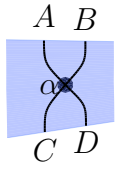


One can check that this satisfies the required properties.

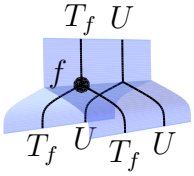
This shows that $\mathcal{F}$ is monoidal with its monoidal structures being respectively cocartesian and cartesian. This is known in the literature as a *distributive category*.

[4] The requirement to whisker our generators T_f by arbitrary objects comes from the two-step definition of our category, using two free constructions. It could be avoided with a free generation in a single shot.

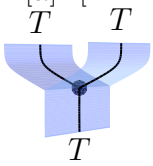
[5] In this model, row order does not matter in this model: tables are sets of rows.



(a) Operation $[\alpha] : [A \times B] \to [C \times D]$



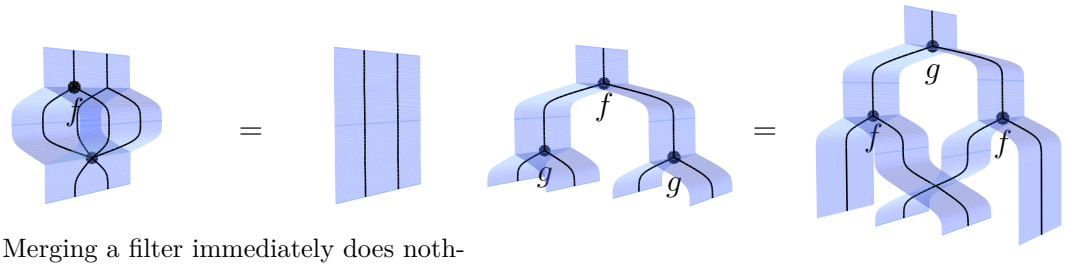
(b) Filter $[f \times U] : [T_f \times U] \to [T_f \times U, T_f \times U]$



(c) Union $J_T : [T, T] \to [T]$

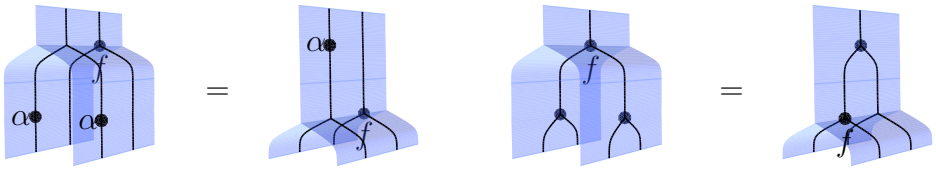
(d) Empty table: $E_T : [] \to [T]$

Figure 3.14: Generators of $\mathcal{F}$.



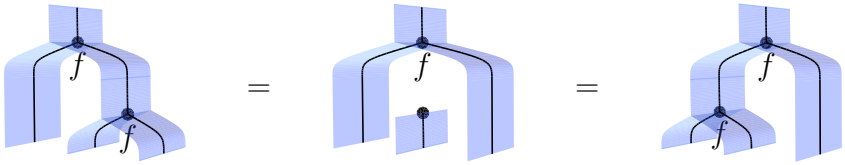
(a) Merging a filter immediately does nothing

(b) Filters commute even with a common column



(c) Disjoint filters and operations commute

(d) Copying and filtering commute



(e) Filters do not modify data

Figure 3.15: Axioms of $\mathcal{F}$.

We could alternatively have constructed $\mathcal{F}$ as a free distributive category, to the cost of making the relationship between the two models less clear. □

We can therefore represent morphisms in $\mathcal{F}$ as sheet diagrams as shown in Figure 3.14. Figure 3.15 states the relations satisfied by these generators using this convention.

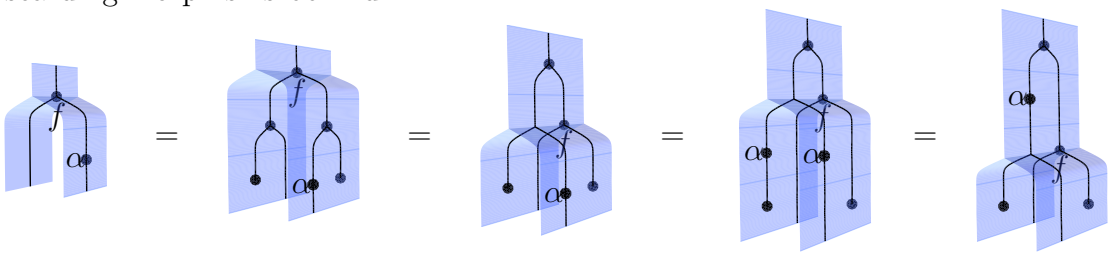
OpenRefine workflows with filters can be represented by morphisms $\mathcal{F}$. For

the converse, we first show that morphisms of $\mathcal{F}$ can be represented in normal form thanks to the following decomposition.

Lemma 3.46. *Let $m \in \mathcal{F}([A],[B])$ be a morphism with one input sheet and one output sheet. There exists a decomposition $m = z \circ y \circ x \circ [w]$ such that $w \in \mathcal{E}$, x only contains filters, y only contains discarding morphisms, and z only contains unions.*

Proof. First, any empty tables E_T in the diagram can be eliminated as co-cartesian units, just like discarding morphisms can be eliminated in the cartesian case (Section 3.5.3).

We then move all operations, copy morphisms and exchanges in $\mathcal{E}$ up to the first sheet. Operations and copy morphisms can be moved past unions and empty tables by the properties of the co-cartesian structure. Although Equation 3.15c can only be used for operations and filters applied to disjoint columns, it can be combined with Equation 3.15d to commute any operation and filter, possibly leaving discarding morphisms behind:



This lets us push all operations up, obtaining the first part of the factorization: $m = \phi \circ [w]$ with $w \in \mathcal{E}$ and ϕ consists of filters, unions, discarding morphisms and exchanges in $\mathcal{F}$.

Unions can be moved down by naturality, obtaining $m = z \circ \phi' \circ [w]$ where ϕ' only consists of filters, discarding morphisms and exchanges in $\mathcal{F}$. Then, all exchanges in ϕ' can be moved down by naturality and absorbed by z. Finally, all discarding morphisms can be moved down past the filters using Equation 3.15c. □

This decomposition can be used to show that all such morphisms arise from OpenRefine workflows, despite the fact that some generators cannot be interpreted as such individually. As stated, this lemma does not provide normal forms yet, as

the order of filters in x is not determined. We will see in the proof of Theorem 3.48 how this can be addressed.

3.5.5 Semantics and completeness

We can give set-valued semantics to $\mathcal{E}$ and $\mathcal{F}$ and obtain completeness theorems for our axiomatization of OpenRefine workflows.

Definition 3.47. *A **valuation** V is given by:*

(i) a set $V(T)$ for each basic datatype $T \in \mathcal{E}$;

(ii) a function $V(\alpha) : V(A) \to V(B)$ for each generator $\alpha \in \mathcal{E}(A, B)$, where $V(A)$ is the cartesian product of the valuations of the basic types in A;

(iii) a subset $V(f) \subset V(T_f)$ for each filter f.

Any valuation V defines a functor $V^* : \mathcal{F} \to \mathbf{Set}$ as follows:

$$\begin{aligned} V^*([A \times \cdots \times B, \ldots, C \times \cdots \times D]) &= (V(A) \times \cdots \times V(B) \sqcup \cdots \sqcup (V(C) \times \cdots \times V(D)) \\ V^*([\alpha]) &= V(\alpha) \\ V^*([f \times U]) &= ((x, u) \mapsto \mathrm{inj}_i(x, u)) \text{ with } i = 1 \text{ if } x \in V(f) \text{ else } i = 2 \\ V^*([J_T]) &= (\mathrm{inj}_i(x) \mapsto x) \\ V^*([E_T]) &= (\text{the initial morphism from the empty set}) \end{aligned}$$

Using the decomposition of Lemma 3.46, we can then show the completeness of our axiomatization for these semantics:

Theorem 3.48. *Let $d, d' \in \mathcal{F}([A], [B])$ be diagrams. Then $d = d'$ by the axioms of $\mathcal{F}$ if and only if $V^*(d) = V^*(d')$ for any valuation V.*

The proof of this theorem is given below. Broadly speaking, it goes by building a valuation where values are syntactic terms, such that a value encodes its entire own history through the processing pipeline. These syntactic values are associated with contexts which record the validity of filter expressions. The decomposition of Lemma 3.46 is then used to compute normal forms for diagrams, which can

be related to the evaluation of the diagram with the syntactic valuation. These normal forms can be computed using a simple diagramatic rewriting strategy, so this also solves the word problem for this signature.

Proof. We can check that all equations of Figure 3.15 preserve the semantics under any valuation, so if two diagrams are equivalent up to these axioms, then their interpretations are equal. For the converse, let us first introduce a few notions. We use a countable set of variables $V = \{x_1, x_2, x_3, \dots\}$.

Definition 3.49. *The set Θ of **terms** is defined inductively: it contains the variables V, and for each an operation symbol $\alpha \in O$ of input arity n and output arity p, it contains the terms $\alpha(t_1, \dots, t_n)[1], \dots, \alpha(t_1, \dots, t_n)[p]$. These terms represent the projections of the operation applied to the input terms.*

The set Θ_n of terms over n variables is the set of terms where only variables from $\{x_1, \dots, x_n\}$ are used. Given $t \in \Theta_n$ and $u_1, \dots, u_n \in \Theta_m$ we can substitute simultaneously all the x_i by u_i, which we denote by $t[u_1, \dots, u_n]$. For instance, let $t = \alpha(\beta(x_1, x_3)[2], x_1)[1]$ and $u_1 = x_3$, $u_2 = x_4$ and $u_3 = \gamma(x_1)[3]$. Then $t[u_1, u_2, u_3] = \alpha(\beta(x_3, \gamma(x_1)[3])[2], x_3)[1]$.

Definition 3.50. *An **atomic filter formula** (**AFF**) over n variables is given by a filter symbol f and terms $t_1, \dots, t_a \in \Theta_n$ where a is the arity of f. It is denoted by $f(t_1, \dots, t_a)$ and represents the boolean condition evaluated on the given terms.*

We denote by Φ the set of all atomic filter formulae. Similarly, Φ_n is the set of AFF over n variables.

Definition 3.51. *A **conjunctive filter formula** (**CFF**) over n variables is a given by a finite set $A \subset \Phi_n \times \mathbb{B}$ of pairs of atomic filter formulae and booleans, called clauses, such that no atomic filter formula appears with both booleans. Such a set represents the conjunction of all its clauses, negated when their associated boolean is false.*

Two CFF A and B are disjoint if they contain the same atomic filter formula with opposite booleans. Otherwise, we can form the conjuction $A \wedge B$, which is the CFF with clauses $A \cup B$.

We denote by Δ the set of CFF and Δ_n that of those over n variables. A CFF is represented as a conjuctive clause in boolean logic, such as $f(x_1, \alpha(x_3, x_2)[1]) \wedge \overline{g}(x_3)$.

Definition 3.52. *A **truth table** t on n variables and p outputs, denoted by $t : n \to p$, is a finite set of cases $c \in \Delta_n \times \Theta_n^p$, such that all the CFF are pairwise disjoint. This represents possible values for an object, depending on the evaluation of some filters.*

*Truth tables t, t' both on n variables and p outputs are **disjoint** if all the CFF involved are pairwise disjoint. The **union** of two disjoint truth tables t, t', denoted by $t \cup t'$, is the union of their cases. The **composition** of truth tables $t : n \to p$ and $t' : p \to q$, denoted by $t; t'$, is formed of the cases $(c_i \wedge c'_j, u'_{j,1}[u_{i,1}, \ldots, u_{i,p}], \ldots, u'_{j,q}[u_{i,1}, \ldots, u_{i,p}])$ for all $(c_i, u_i) \in t$ and (c'_j, u'_j) such that c_i and c_j are compatible.*

*A collection of truth tables $(t_i : n \to p)_i$ forms a **partition** if the CFF in them are all disjoint and their disjunction is a tautology. The **projection** of a truth table t with p outputs on its k-th component, $1 \leq k \leq k$, denoted by $t[k]$, is given by the cases $(c_i, u_{i,k})$ for $(c_i, u_i) \in t$. The **product** of truth tables $t : n \to p$ and $t' : n \to q$, denoted by $t \otimes t' : n \to p + q$, is given by the cases $(c_i \wedge c'_j, u_i, u'_j)$ for all $(c_i, u_i) \in t$ and $(c'_j, u'_j) \in t'$ such that c_i and c_j are compatible. Two truth tables $t, t' : n \to p$ are **equivalent**, denoted by $t \sim t'$, if all the cases in $t \otimes t'$ have value tuples of the form $(v_1, \ldots, v_p, v_1, \ldots, v_p)$.*

One can check that all the properties and operations on truth tables defined above respect the equivalence relation $\sim$: we will therefore work up to this equivalence in the sequel. We can represent truth tables by their list of cases:

$$f(x_1) \wedge g(\gamma(x_2)[1]) \mapsto (x_3, \alpha(x_1, x_1)[1])$$
$$f(x_1) \wedge \overline{g}(\gamma(x_2)[1]) \mapsto (x_2, \alpha(x_1, x_1)[1])$$
$$\overline{f}(x_1) \mapsto (\beta(x_2, x_3)[2], x_1)$$

With the syntactic objects just defined, we can now define semantics for $\mathcal{F}$ that are independent of any valuation. The morphisms will be families of truth tables, which can interpret the generators of $\mathcal{F}$.

Definition 3.53. *The category $\mathcal{T}$ is a symmetric monoidal category with $\mathrm{Ob}(\mathcal{T}) = \mathbb{N}^*$ (lists of natural numbers) and where the monoidal product is given by concatenation. A morphism $t \in \mathcal{T}([a_1, \ldots, a_n], [b_1, \ldots, b_m])$ is a collection $t_{i,j}$ of truth tables, $1 \leq i \leq n$ and $1 \leq j \leq m$, such that $t_{i,j}$ is of type $a_i \to b_j$, and for each i, $(t_{i,j})_j$ is a partition. Furthermore, we require that $m > 0$ unless $n = 0$.*

Given morphisms $t : [a_1, \ldots, a_n] \to [b_1, \ldots, b_m]$ and $u : [b_1, \ldots, b_m] \to [c_1, \ldots, c_p]$, the composite $t; u$ is given by $(t; u)_{i,k} = \bigcup_{1 \leq j \leq m} (t_{i,j}; u_{j,k})$.

The tensor product of $t : [a_1, \ldots, a_n] \to [b_1, \ldots, b_m]$ and $u : [c_1, \ldots, c_p] \to [d_1, \ldots, d_q]$ is the morphism $v : [a_1, \ldots, a_n, c_1, \ldots, c_p] \to [b_1, \ldots, b_m, d_1, \ldots, d_q]$ defined by $v_{i,j} = t_{i,j}$ for $i \leq n$ and $j \leq p$, $v_{i,j} = u_{i-n,j-m}$ for $i > n$ and $j > p$, and the empty truth table otherwise.

The identity 1 on $[a_1 \ldots, a_n]$ is given by $1_i = \top \mapsto (x_1, \ldots, x_n)$.

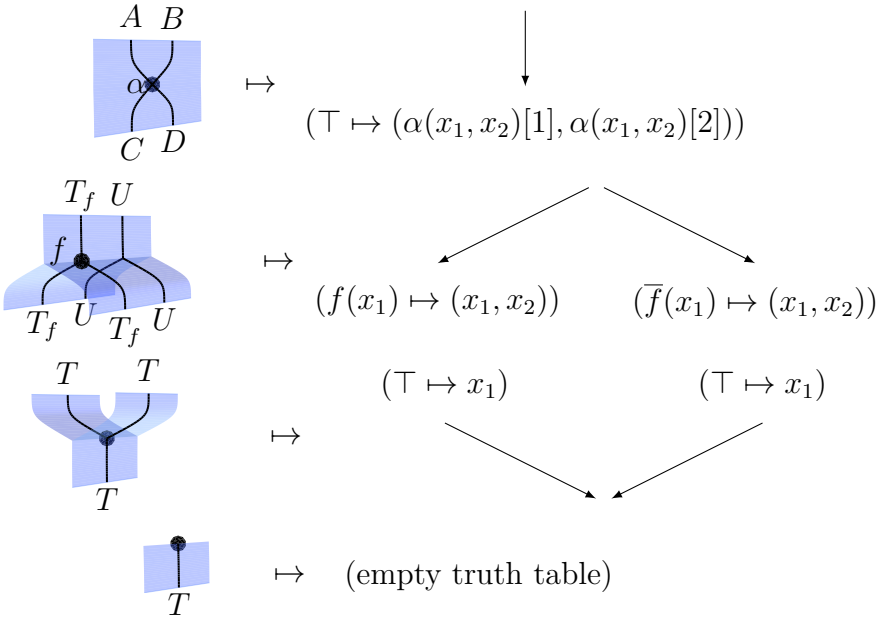
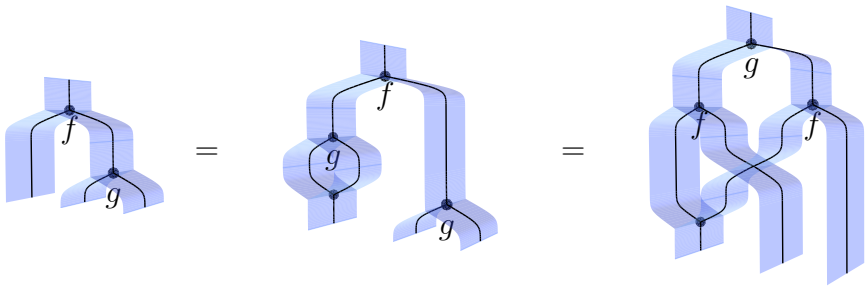


Figure 3.16: Definition of $P : \mathcal{F} \to \mathcal{T}$.

There is a functor $P : \mathcal{F} \to \mathcal{T}$ defined on objects by $P([A_1 \times \cdots \times A_n]) = [n]$ and on morphisms by Figure 3.16. One can check that it respects the axioms of $\mathcal{F}$.

Lemma 3.54. *The functor P is faithful.*

Proof. We show this by relating the image $P(d)$ of a diagram to its decomposition given by Lemma 3.46. As such, this decomposition does not give a normal form, as the order of the filters remains unspecified. However, successive filters can be swapped freely:



Let us pick an arbitrary order on Φ, the set of atomic filter formulae. In a diagram d decomposed by Lemma 3.46 as $z \circ y \circ x \circ [w]$, Each occurence of a filter in x can be associated with an AFF defined by the filter symbol for the filter and the term obtained from the wires read from w. Commuting filters as above does not change their corresponding AFF. Therefore, this determines an order on the filters occurring in x. We can rearrange the filters so that f appears above g if their corresponding AFFs are ordered accordingly. This will add new exchanges and unions in x, but we can use Lemma 3.46 a second time to push these to their part of the decomposition, as this procedure does not reorder the filters.

The rest of the decomposition can be normalized too: unions can be normalized by associativity, and any discarding morphism that is present in all sheets of y and discards a wire not read by any filter in x can be pushed up into w, which can be normalized as a morphism in $\mathcal{E}$.

From such a normalized decomposition, we can read out the truth table $P(d)$ directly. Each sheet in y corresponds to a case of $P(d)$, whose condition is determined by the conjunction of all the AFFs of the filters leading to it, with the appropriate boolean depending on the side of the filter they are on. Therefore, if $P(d) = P(d')$, then $d = d'$. □

Definition 3.55. *The **syntactic valuation** S is defined as follows. For each basic datype T, $S(T) = \Theta \times 2^\Phi + \{\bot\}$. In other words, a value can be either a term together with a context of true atomic filter formulae, or an inconsistent value $\bot$. For each facet f, $S(f) : (C, t) \mapsto f(t) \in C$: a facet is true if it belongs to the context.*

For each operation $\alpha : T_1 \times \cdots \times T_n \to U_1 \times \cdots \times U_m$,

$$(\alpha) : ((C, t_1), \ldots, (C, t_n)) \mapsto ((C, \alpha(t_1, \ldots, t_n)[1], \ldots, \alpha(t_1, \ldots, t_n)[m]))$$
$$\textit{anything else} \mapsto \bot$$

There is a functor $\Pi : \mathcal{T} \to \mathbf{Set}$, defined on objects by $\Pi([n_1, \ldots, n_p]) = (\Theta \times 2^\Phi + \{\bot\})^{n_1} \sqcup \cdots \sqcup (\Theta \times 2^\Phi + \{\bot\})^{n_p}$. Given a morphism $t : n \to p$, we define $\Pi(t)(\mathrm{inj}_i(x))$ as follows. If x contains any $\bot$ or if the contexts in it are not all equal, then $\Pi(t)(\mathrm{inj}_i(x)) = \mathrm{inj}_1((\bot, \ldots, \bot))$.[6] Otherwise, as the truth tables $(t_{i,j})_j$ form a partition, there is a single case (C, y) in all of them such that the associated CFF is true in the common context C. Let j be the output index of its truth table: we set $\Pi(t)(\mathrm{inj}_i(x)) = \mathrm{inj}_j(y[x])$. One can check that this defines a monoidal functor.

Lemma 3.56. *The functor Π is faithful.*

Proof. For simplicity, let us concentrate on the case of morphisms $t, t' : [n] \to [p]$: this is the only case that is actually needed to prove the completeness theorem, and the general case is similar. If $\Pi(t) = \Pi(t')$, then consider $t \otimes t' : n \to 2p$. For each case $(f, u, u') \in t \otimes t'$, with f a CFF and u, u' tuples of terms, $(f, u) = \Pi(t)(f, x_1, \ldots, x_n) = \Pi(t')(f, x_1, \ldots, x_n) = (f, u')$, so $u = u'$. Therefore, $t \sim t'$. □

Finally, combining Lemma 3.54 and Lemma 3.56, we obtain that $\Pi \circ P : \mathcal{F} \to \mathbf{Set}$ is faithful. But in fact $\Pi \circ P = S^*$, the functor arising from the valuation S. So, if two diagrams $d, d' \in \mathcal{F}$ give equal interpretations under any valuation V, then it is in particular the case for $V = S$, and by faithfulness of S^*, $d = d'$. □

[6] This is possible because we have assumed that codomains of morphisms in $\mathcal{T}$ are nonempty except for the identity on the monoidal unit.

We conjecture that this result generalizes to arbitrary morphisms in $\mathcal{F}$, with multiple input and output tables. However, all OpenRefine workflows have one input and one output table, so the theorem already covers these.

4 Word problems

Contents

4.1 Introduction

As we have seen in Chapter 2, equivalence of string diagrams is a geometrical notion, with two string diagrams being equivalent (that is, representing equal morphisms of the corresponding monoidal category) just when their string diagram representations are related by some kind of isotopy. For instance, for monoidal categories, two string diagrams are equivalent when there is a *recumbent isotopy* between them. This captures exactly the equational theory of free monoidal categories, without imposing any extra relation on the generators. Despite the wide applicability of string diagrams and their implementation in various software packages, there are no general results about the complexity of deciding equivalence of string diagrams. Understanding this problem is a prerequisite for many applications and is also a way to learn interesting facts about the combinatorics of those objects.

In this chapter, we will study various decision problems of this form: given two string diagrams, determine if they are isotopic and therefore represent the same morphisms. Those decision problems are called *word problems*, because they are generally expressed not in terms of diagrams but of formulae (hence *word*) in some algebraic structure. Given that translating between the string diagram representation and the term representation is a relatively simple process, the complexity results we obtain on string diagrams generally extend to term-based representations in a straightforward way.

It is worth noting that in many contexts, word problems consist in deciding the equality in some algebraic structure freely generated by some generators and relations between them. For instance, a fundamental result in computer science is that the word problem for monoids is undecidable, meaning that equality in a finitely generated and finitely presented monoid is in general undecidable (Post, 1947).

In our case, we will only address the case of word problems where no additional relations are imposed on generators: the only equalities which hold are those imposed by the ambient algebraic structure. The reason behind this choice is that

the algebraic structures we are concerned with are much richer than, say, monoids, so those simpler word problems are already worth studying on their own. Moreover, as soon as we allow arbitrary relations in the signatures which generate our free categories, the word problem becomes undecidable precisely for the same reason that the general word problem for monoids is undecidable (since the composition operator $\circ$ defines a monoid structure on the endomorphisms of any object in a category).

Outline We start this chapter by our first result in this field, providing algorithms for the word problem for monoidal categories.

Theorem 4.71. *The word problem for free monoidal categories, without equations imposed on the generators, can be solved in quadratic time in the number of edges and vertices of the diagram.*

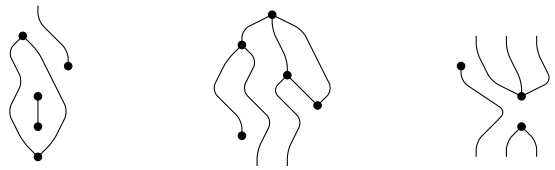
We then show how this approach can be extended to double categories.

Theorem 4.104. *The word problem for free double categories, witout equations imposed on the generators, can be solved in quadratic time in the number of edges and vertices of the diagram.*

Finally, the last section gives a hardness result for the word problem for braided monoidal categories.

Theorem 4.142. *The unknotting problem can be polynomially reduced to the word problem for free braided monoidal categories, without equations imposed on the generators.*

Implementations We did not attempt to implement any of the algorithms described in this thesis, as our interest in these decision problems mostly comes from an interest in determining their complexity from a theoretical standpoint, rather than developing applications out of those algorithms. That being said, we are aware of one implementation of our results on non-symmetric monoidal categories (de Felice et al., 2020), although they do not implement Theorem 4.71 but the conceptually simpler approach consisting in applying right exchanges until reaching a normal form.



(a) Disconnected **(b)** Connected **(c)** Boundary-connected

Figure 4.1: Connectedness for string diagrams

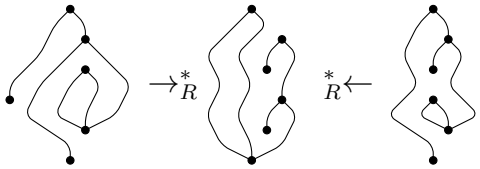


Figure 4.2: Two connected diagrams with the same right normal form

4.2 Non-symmetric monoidal categories

This section is taken from the article *Normalization for planar string diagrams and a quadratic equivalence algorithm* (Delpeuch and Vicary, 2018), to appear in *Logical Methods in Computer Science*.

In this section we take a first look at the complexity of the general equivalence problem for planar string diagrams[1] (henceforth simply *diagrams*), without additional axioms. This does not include all the features used by some applications of string diagrams (for example, braided monoidal categories which will be covered in Section 4.4), but it is already a nontrivial setting, and lays the groundwork for more elaborate settings.

Our main results are as follows. We write v for the number of vertices in a diagram, and e for the number of edges; also, we say that a diagram is *connected* when there is a path in the diagram between any two vertices, and *boundary-connected* when it is either connected, or every vertex has a path in the diagram to a boundary. See Figure 4.1 for examples of these notions.

[1] As seen in the first chapter, by Joyal and Street Joyal and Street (1991), this corresponds to the word problem for monoidal categories which are free on a given generating set of objects and morphisms. Furthermore, our results extend immediately to bicategories which are free on a given set of generating 1- and 2-morphisms, but we prefer to keep the discussion at the level of monoidal categories.

- For boundary-connected diagrams, we build a rewrite strategy that generates the equality relation, show it is strongly normalizing (Theorem 4.23), show it terminates after $O(v^3)$ steps (Theorem 4.29), show the normal forms can be constructed in $O(ve)$ time (Theorem 4.40), and show equality can be decided in $O(v + e)$ time (Corollary 4.80).

- For general diagrams, with no constraints on connectivity, we use the above results to derive a scheme that decides equality in $O(ve)$ time (Theorem 4.71).

- We show that the recumbency property listed above is unnecessary; that is, we show that two diagrams are recumbent isotopic, and hence equal, just when they are isotopic (Theorem 4.82).[2] This proves a conjecture of Selinger (2010).

This final result is attractive, since in practice the recumbency property is constraining, forcing the entire diagram to remain essentially "vertical" throughout the isotopy.

4.2.1 Combinatorial encoding of string diagrams

Our complexity results will be based on the combinatorial representation of string diagrams sketched in Section 2.2.3. In this section we give a more precise definition of the data structure involved and its properties, analyzing the cost of some basic operations on string diagrams. We will use the RAM model, used for instance by Hopcroft and Wong (1974), which assumes constant time random access to the working memory and constant time arithmetic operations on integers. This is a widespread model, which closely matches the architecture of today's computers, despite the fact that arithmetic operations on unbounded integers would not be implementable in constant time. We discuss the implications of this feature in Sections 4.2.6 and 4.2.7.

[2] This has a nice expression in categorical terms (Corollary 4.84): it says that for a monoidal signature Σ, the embedding functor from the free monoidal category on Σ to the free pivotal category on Σ is faithful.

As exposed in Section 2.2.3, the main idea is that a diagram is cut into slices given by Lemma 2.15, and each slice can be described by the following data:

- the number of wires at the top and bottom of the slice (which we will not directly encode as they are redundant with the neighbouring slices);
- the number of input and output wires for the generator in the slice;
- the horizontal position of the generator, described for instance by the number of wires passing to the left of the generator;
- the generator morphism itself, denoted by an identifier taken from the signature. For our purposes this will be omitted to simplify the presentation, as our results are applicable to any signature.

This encoding scheme serves as a formal combinatorial foundation for our results, although we will build most of our arguments at a more intuitive level with the corresponding graphical diagrams.

We give an example of a diagram, together with its encoding, in Figure 4.3. Note that in this example diagram, and in the other diagrams in the paper, we use small circles for the vertices, rather than boxes which are sometimes seen.

Definition 4.1. *Given $n \in \mathbb{N}$, we define the interval $[\![n]\!] = \{0, \dots, n-1\}$, a totally ordered set.*

Definition 4.2. *A **diagram** D is given by:*

- *$S(D)$, its number of **source edges**;*
- *$N(D)$, its **height**;*
- *For all $n \in [\![N(D)]\!]$, $H(D, n)$, the **vertex offset** at height n;*
- *For all $n \in [\![N(D)]\!]$, $I(D, n)$, the **number of inputs** at height n;*
- *For all $n \in [\![N(D)]\!]$, $O(D, n)$, the **number of outputs** at height n.*

From this data, we can deduce the number of edges at any horizontal level in the diagram. At the top boundary this is simply given by $S(D)$, and then as we browse the diagram towards the bottom end, each slice removes $I(D, n)$ and adds $O(D, n)$ edges. We define this formally as follows.

Definition 4.3. *For a diagram D, we define $\Delta(D, n) = O(D, n) - I(D, n)$ for all $n \in [\![N(D)]\!]$.*

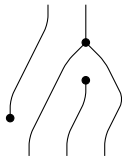
Definition 4.4 (Wires at each level)**.** *For a diagram D, we define $W(D, 0) = S(D)$ and $W(D, n + 1) = W(D, n) + \Delta(D, n)$ for all $n \in [\![N(D)]\!]$.*

The data structure we have defined so far lets us encode ill-defined diagrams, which happens when a given diagram slice does not contain enough edges for the next vertex to consume as inputs.

Definition 4.5. *A diagram D is **valid** when for all $n \in [\![N(D)]\!]$, we have $W(D, n) \geq H(D, n) + I(D, n)$.*

We now formalize the right and left exchange moves illustrated in Figure 4.4. All that needs to be checked is that the two vertices with adjacent heights share no common edges.

Definition 4.6. *For $n \in [\![N(D) - 1]\!]$, a diagram D **admits a left exchange move** at height n when $H(D, n + 1) \geq H(D, n) + O(D, n)$, and **admits a right exchange move** at height n when $H(D, n) \geq H(D, n + 1) + I(D, n + 1)$. A **right reduction** is a series of right exchange moves.*



$$
\begin{array}{rrr}
S(D) = 2 & N(D) = 3 & \\
H(D, 0) = 1 & I(D, 0) = 1 & O(D, 0) = 2 \\
H(D, 1) = 2 & I(D, 1) = 0 & O(D, 1) = 1 \\
H(D, 2) = 0 & I(D, 2) = 1 & O(D, 2) = 0
\end{array}
$$

Figure 4.3: *Example of a diagram D together with its encoding.*

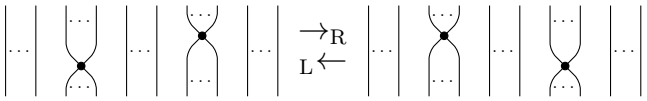


Figure 4.4: Right and left exchanges as rewrites on diagrams.

Definition 4.7. *For a diagram D which admits a left or right exchange move at height $n \in [\![N(D) - 1]\!]$, its **left exchange** $L^n(D)$ or **right exchange** $R^n(D)$, respectively, is defined to be identical to D, except at heights n, $n+1$ as follows:*

$$\begin{aligned}
H(L^n(D), n) &= H(D, n+1) - \Delta(D, n) \\
I(L^n(D), n) &= I(D, n+1) \\
O(L^n(D), n) &= O(D, n+1) \\
H(L^n(D), n+1) &= H(D, n) \\
I(L^n(D), n+1) &= I(D, n) \\
O(L^n(D), n+1) &= O(D, n) \\
H(R^n(D), n) &= H(D, n+1) \\
I(R^n(D), n) &= I(D, n+1) \\
O(R^n(D), n) &= O(D, n+1) \\
H(R^n(D), n+1) &= H(D, n) + \Delta(D, n+1) \\
I(R^n(D), n+1) &= I(D, n) \\
O(R^n(D), n+1) &= O(D, n)
\end{aligned}$$

Lemma 4.8. *For a valid diagram D which admits a left (or right) exchange move at height n, its left exchange $L^n(D)$ (or right exchange $L^n(D)$) is a valid diagram.*

Proof. D be a valid diagram which admits a right exchange at height n. We show that that $R^n(D)$ is valid again. The case of a left exchange is analogous.

We need to check that for each height $k \in [\![N(R^n(D))]\!]$, we have $H(R^n(D), k) + I(R^n(D), k) \leq W(R^n(D), k)$.

For $k < n$ or $k > n+1$, $H(R^n(D), k) = H(D, k)$, $I(R^n(D), k) = I(D, k)$ and $W(R^n(D), k) = W(D, k)$ so the inequality holds as D is valid.

For $k = n$, by definition of $R^n(D)$ we have $H(R^n(D), n) + I(R^n(D), n) = H(D, n+1) + I(D, n+1)$. As D admits a right exchange at height n, this is bounded by $H(D, n)$, so a fortiori $W(D, n)$.

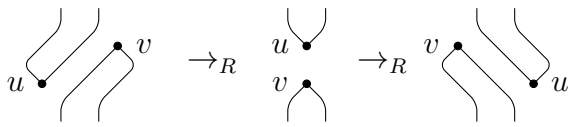
For $k = n+1$, $H(R^n(D), n+1) + I(R^n(D), n+1) = H(D, n) + O(D, n+1) - I(D, n+1) + I(D, n)$. Since D is valid, this is bounded by $W(D, n) + O(D, n+1) - I(D, n+1) = W(R^n(D), n+1)$. □

It is a consequence of Theorem 2.11 that right and left exchanges preserve the meaning of the diagram. With respect to our data structure described here, it is clear that the following operations can be performed in constant time, since they involve computing fixed formulae over the natural numbers, and testing a fixed number of inequalities:

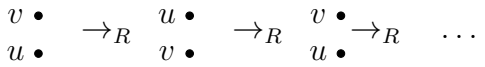
- checking whether a left or right exchange is admissible at a given height;
- given an admissible left or right exchange, computing the rewritten diagram in place.

Furthermore, the memory space needed to represent a diagram is linear in the number of vertices. We will use these observations as building blocks for our complexity arguments in the main part of the paper.

We write $\rightarrow_{\mathrm{R}}$ (respectively $\rightarrow_{\mathrm{L}}$) for the relation on diagrams given by a single right exchange (respectively left exchange). Figure 4.4 shows what those relations look like in general. We illustrate some interesting cases of these exchange moves. In degenerate cases where u and v have no inputs or outputs, it can be possible to apply two right exchanges in sequence to the same pair of vertices:



Furthermore, if there are no edges at all, then right exchanges can be applied indefinitely, which corresponds to the Eckmann-Hilton argument shown in Section 2.2.



Throughout this chapter we will use a braid notation to represent right reductions (series of right exchanges), such as in Figure 4.6 or Figure 4.7. These braidings represent the trajectories of the vertices as the reduction progresses, seen from the right-hand side of the diagram. Each crossing in the braid diagram corresponds to an exchange of two nodes in the string diagram.

Converting between representations of morphisms

The combinatorial encoding given in this section can be rendered into an actual diagram. Generating such a representation from our encoding involves computing suitable planar layouts for the vertices and edges respecting all the properties of this class of topological graphs. Various algorithms can be used to this end. In this work we use a simple layout strategy that enforces a constant vertical spacing between diagram levels and a constant horizontal spacing between the wires at each level.[3] Each level is horizontally centered based on the number of wires that cross it. Vertices are horizontally centered between their input and output ports. An example of such a rendered diagram can be found in Figure 4.3.

It is also possible to convert a morphism expression, i.e. a term denoting a morphism, into a combinatorial encoding of its string diagram in linear time.

Theorem 4.9. *Two morphisms expressions denote the same morphism if and only if the corresponding diagrams are related by a series of exchanges.*

Proof. Theorem 2.11 shows that two morphism expressions denote the same morphism if and only if their string diagrams are related by a deformation of recumbent graphs. Therefore we only need to show that string diagrams are related by a deformation if and only if their combinatorial encodings are related by a series of exchanges.

If two diagrams are related by a series of exchanges, they represent the same morphism as exchanges preserve denotation.

Conversely, let h be a deformation between diagrams Γ and Γ' in generic position, with $h(0) = \Gamma$ and $h(1) = \Gamma'$. We can assume that $h(t)$ is always in generic position except for a finite number of $t \in (0, 1)$: if it is not the case, translate each vertex v_i vertically by ϵ_i, uniformly for all $t \in [0, 1]$. The ϵ_i can be chosen to make sure no vertices remain at the same height for a non-trivial interval $t \in [u, v]$.

Furthermore, we can make sure that when $h(t)$ is not in generic position, only two vertices in $h(t)$ are at the same height. If this is not the case, the deformation

[3] It is simple enough to be programmed in LaTeX, so that our string diagrams are generated with this rendering process, directly from their combinatorial encodings.

can be modified to satisfy this condition by picking delays η_i for each vertex v_i, and delaying the movement of each vertex v_i by η_i over the course of the transformation. Again, the delays can be chosen collectively to ensure that at most two vertices occupy the same height at a given time.

Let $t_1 < \cdots < t_k$ be the instants at which $h(t)$ is not in generic position. For any other $t \in [0, 1]$ the combinatorial encoding of $h(t)$ is defined. By connectedness, the combinatorial encoding of $h(t)$ is constant for $t \in (t_i, t_{i+1})$ so this defines a sequence of diagrams $D_0, \ldots, D_k$. Since at most one pair of vertices exchange heights around instant t_i, D_i and D_{i+1} are related by a single exchange move or are equal. This gives the required sequence of exchanges between the source and target diagrams. □

Therefore, solving the word problem for free monoidal categories can be done by providing an algorithm to determine if two diagrams can be related by a series of exchanges. We will first show that right reductions form a terminating and confluent rewriting strategy on connected diagrams. Termination will be shown in Section 4.2.2 and confluence in Section 4.2.4.

4.2.2 Termination

To prove termination of right reductions on connected diagrams, we first introduce the class of linear diagrams. The general case of connected diagrams will then directly follow from our results on linear diagrams. In fact, we will see in Lemma 4.30 that linear diagrams exhibit the longest reductions.

Definition 4.10. *A diagram with n vertices is **linear** if it is connected, acyclic and has only two leaves (vertices connected to only one edge). We identify its vertices with the indices $1, \ldots, n$ such that 1 and n are the leaves, and k is connected to $k-1$ and $k+1$ for all $1 < k < n$.*

As the name indicates, linear diagrams have a path-like shape (since they are connected and cannot contain any branching). Figure 4.5 gives an example of a

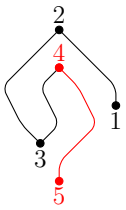


Figure 4.5: Example of a linear diagram, with final vertices in red

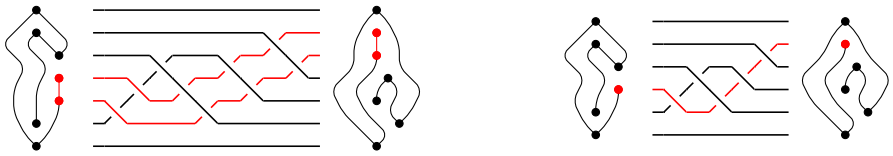


Figure 4.6: A collapsible reduction and its collapsed counterpart

linear diagram. The choice of the start and end of the indexing is arbitrary, as it can be reversed. We therefore assume that linear diagrams come with a chosen order.

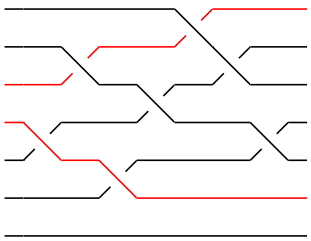
Definition 4.11. *In a linear diagram with n vertices, $n \geq 2$, the **final** vertices are the vertices $n-1$ and n.*

Definition 4.12. *In a linear diagram, the **final interval** is the set of vertices whose height is between the heights of the final vertices, including the final vertices themselves. If the final interval only consists of the final vertices, the diagram is **collapsible**.*

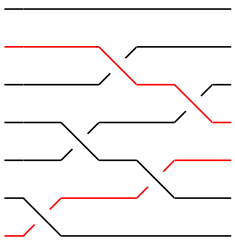
In Figure 4.5, vertices 1 and 3 are in the final interval, as well as the final vertices themselves, vertices 4 and 5.

Definition 4.13. *A right reduction is **collapsible** when its source and target are collapsible, and any exchange between a non-final vertex v and a final vertex f_1 is immediately followed or preceded by an exchange between v and the other final vertex f_2. In other words, all non-collapsible steps of the reduction are isolated.*

We call these reductions collapsible because as the final vertices move synchronously, they can be merged together: this defines a reduction on a shorter linear diagram. Figure 4.6 shows an example of a collapsible reduction with the final vertices in red, and the corresponding collapsed reduction on the shorter diagram.



(a) A funnel with collapsible source



(b) A funnel with collapsible target

Figure 4.7: Example of funnels

Definition 4.14. *Given a collapsible reduction on a linear diagram l of size n, the corresponding* ***collapsed reduction*** *is obtained by erasing vertex number n in l.*

Definition 4.15. *A right reduction of string diagrams $r : A \to_{\mathrm{R}}^{*} B$ is called a* ***funnel*** *when:*

- *each non-final vertex is exchanged at most once with a final vertex.*
- *if an exchange involves non-final vertices u and v, then both u and v are exchanged with a final vertex in the course of the rewrite, and these two final vertices are different.*

We are especially interested in the cases where the source or target of the funnel is collapsible, as in Figure 4.7. The name *funnel* comes from the shape of these reductions when depicted as braids: these are reductions where the final vertices converge or diverge from each other.

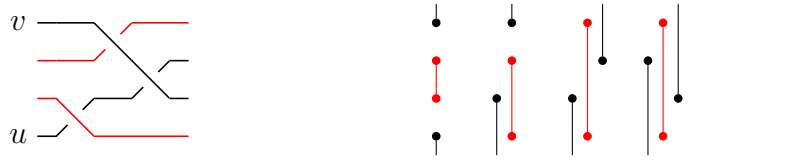
The following lemmas will establish various properties of funnels that we will need for the decomposition of Lemma 4.21.

Lemma 4.16. *Let $r : A \to_{\mathrm{R}}^{*} B$ be a funnel with A collapsible and $e : B \to_{\mathrm{R}} C$ be a right exchange of two non-final vertices u and v that are not touched by r. Then the reduction $r; e : A \to_{\mathrm{R}}^{*} B \to_{\mathrm{R}} C$ can be rearranged as $e'; r' : A \to_{\mathrm{R}} B' \to_{\mathrm{R}}^{*} C$, where e' exchanges u and v in A, and r' is a funnel.*

Proof. As u and v are not touched by r, the two reductions commute directly. □

Lemma 4.17. *Let $r : A \to^*_{\mathrm{R}} B$ be a funnel reduction where A or B is collapsible. Then, the trajectory of all non-final vertices is monotone in r, meaning that they trajectory does not contain both moves that make them go up and moves that make them go down.*

Proof. Let us assume by symmetry that the source A of the reduction is collapsible. Consider an exchange of non-final vertices u and v in r. By definition, u and v are exchanged with two different final vertices over the course of r. Because A is collapsible, this means that both u and v have entered the final interval earlier in the reduction, by being exchanged with the bottom and top final vertices (respectively). Figure 4.8 shows the general position of such an exchange.



(a) The general position of an exchange of final vertices in r.

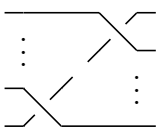
(b) Relative horizontal positions of nodes in r

Figure 4.8: Horizontal position of non-final nodes in a funnel.

As all the exchanges involved are right exchanges, u and v are on different sides of the final edge when they are exchanged: u is on the left and v is on the right of the final edge. This means that u necessarily goes up and v goes down. As this applies to all exchanges of non-final vertices, this means that the trajectory of both vertices is monotone. □

Definition 4.18. *An **interval right exchange** $i : A \to^*_{\mathrm{R}} B$ is a right reduction swapping a vertex x and a set of consecutive vertices $v_1, \ldots, v_k$ which is adjacent to x in A and B. The vertex x is exchanged first with v_1, then v_2, up to v_k.*

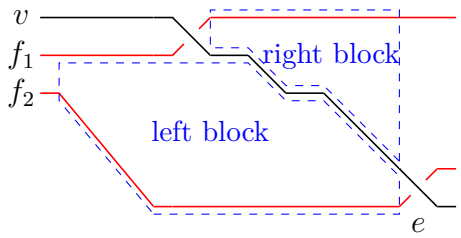
An interval right exchange looks like this:



Lemma 4.19. *Let $r : A \to_{\mathrm{R}}^{*} B$ be a funnel reduction with A collapsible and $e : B \to_{\mathrm{R}} C$ be an exchange of a non-final vertex v with a final vertex f_2, such that v is exchanged with the other final vertex f_1 in r. This gives a reduction path $r; e : A \to_{\mathrm{R}}^{*} B \to_{\mathrm{R}} C$. A reduction of the same length can be obtained: $i; r' : A \to_{\mathrm{R}}^{*} D \to_{\mathrm{R}}^{*} C$ where r' is a funnel reduction and i exchanges v with the final interval in A.*

Proof. By symmetry let us assume that f_1 is the highest final vertex, and f_2 is the lowest. Somewhere in r, v enters the final interval by being exchanged with f_1. By Lemma 4.17, the trajectory of v in r is monotone. In fact, because v ends up being adjacent to f_2 in B, v is exchanged exactly once with each non-final vertex that is exchanged with f_2 over the course of r.

Exchanges that do not involve v can be divided in two blocks: the ones that are on the right of the trajectory of v, and the ones that are on the left. The block on the right commutes with e because the vertices they exchange are disjoint, so we can permute the two.



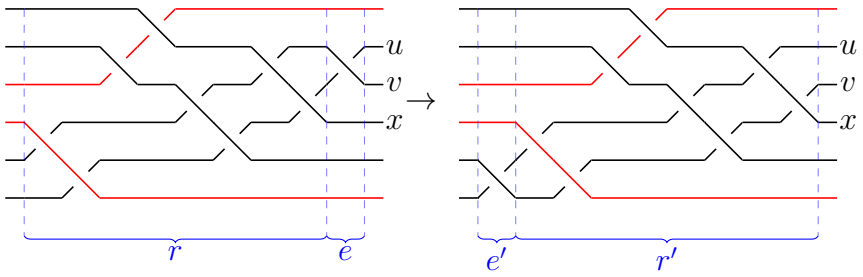
We now need to pull the block on the left through the exchanges involving v. Notice that v is the first vertex to be exchanged with f_1 over the course of r. This is because all other such vertices cannot be exchanged with v in f and v is adjacent to f_2 in B. Thus, the block on the left does not contain any exchange involving f_1: it only contains exchanges involving non-final vertices or f_2. By successive application of the Reidemeister type III move (which pulls one exchange through two other exchanges), we can therefore pull the left block through the trajectory of v. □

Lemma 4.20. *Let $r : A \to_{\mathrm{R}}^{*} B$ be a funnel reduction with A collapsible, followed by an exchange $e : B \to_{\mathrm{R}} B'$ of two non-final vertices u, v such that both vertices are*

exchanged with the same final vertex f in r. Then, the sequence $r; e$ can be rewritten as $e'; r' : A \to_{\mathrm{R}} A' \to_{\mathrm{R}}^* B'$ where e' exchanges u and v in A, and r' is a funnel.

Proof. We show that e can be pulled through all exchanges involving u or v in r. By symmetry, we will assume that the final vertex f exchanged with u and v is the lowest one, and that u is the vertex below v in B.

By induction, consider the last exchange in r that involves one of u or v and another vertex x. Because the trajectories of u and v always go up by Lemma 4.17, the trajectory of x goes down. As u and v are adjacent in B, this last exchange must be between u and x, and x must have been exchanged previously with v. Moreover, this previous exchange is necessarily the last one involving v (otherwise any later exchange with y would require a later exchange between y and u). Therefore, e can be pulled through the last exchanges involving u and v.



We perform these pull-through moves inductively, which eventually moves e' at the beginning of the reduction. The subsequent exchange the same nodes as r in the same order, so they form a funnel. □

Finally, the following lemma decomposes reductions on linear diagrams into two parts: a collapsible part and a funnel part. This decomposition is illustrated by Figure 4.11. As a collapsible reduction can be seen as a reduction on a shorter linear diagram, this will let us work inductively on the size of the linear string diagram.

Lemma 4.21. *Let $r : A \to_{\mathrm{R}}^* B$ be a reduction with A collapsible. Then r can be rearranged and decomposed as*

$$c; f : A \to_{\mathrm{R}}^* X \to_{\mathrm{R}}^* B$$

with c collapsible and f a funnel.

Proof. We construct the decomposition into collapsible and funnel parts by induction on the length of the rewrite r. For length 0, the result is clear. For length 1, there are two cases: if the exchange touches a final vertex, then it goes in the funnel part of the decomposition, otherwise it forms the collapsible part.

Assume we have a rewrite of length $k+1$. Use the induction hypothesis to decompose the first k exchanges:

$$c; f; z : A \to_{\mathrm{R}}^{*} X \to_{\mathrm{R}}^{*} B' \to_{\mathrm{R}} B$$

with c collapsible and f a funnel.

If $f; z$ is also a funnel, then this gives us the required decomposition. Otherwise, this funnelity can fail for multiple reasons.

First, it can be that z exchanges a final vertex v with a non-final vertex w that is already exchanged with a final vertex in f. In this case, by Lemma 4.19, we can rearrange $f; z$ into $i; f'$ where f' is a funnel and i exchanges v with the final interval. As the domain of i is collapsible, i is collapsible itself so we have the required decomposition.

Second, it can be that z exchanges two non-final vertices that are not exchanged with any final vertex in f. In this case, by Lemma 4.16, z commutes with f: we obtain $c; z; f : A \to_{\mathrm{R}}^{*} X \to_{\mathrm{R}} X' \to_{\mathrm{R}}^{*} B$, and $c; z$ is collapsible so we have the required decomposition.

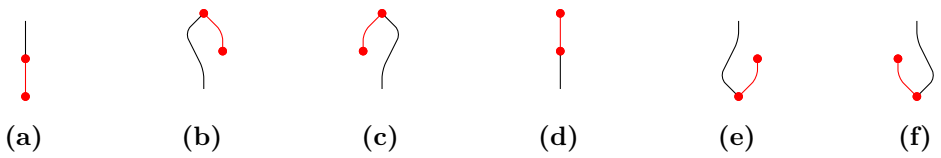
It cannot be the case that only one of the two non-final vertices z exchanges has been previously exchanged with a final vertex in f. This is because the heights of all vertices which have been exchanged with a final vertex lie in the final interval, and all other non-final vertices are outside the final interval.

Third, it can be that z exchanges two non-final vertices that are both exchanged in f with a final vertex. In this case, as we have assumed that $f; z$ is not final, it must be the vertices were exchanged with the same final vertex. We can therefore apply Lemma 4.20 and rearrange the rewrite into $e'; f'$ with e' exchanging the same non-final vertices as z and f' funnel. As e' is collapsible, this gives the required decomposition.

Finally, it cannot be the case that z exchanges the two final vertices, as final vertices can never be exchanged together since they are connected by an edge. □

Lemma 4.22. *Let $r : A \to_{\mathrm{R}}^{*} B$ be a right reduction on a linear diagram. Then r can be extended on some side such that its domain or codomain is collapsible.*

Proof. Our strategy to extend r depends on the topology of the final vertices. We know that vertex n is connected solely to $n-1$ and that $n-1$ is connected to both $n-2$ and n. Here are the possible ways these connections can happen:



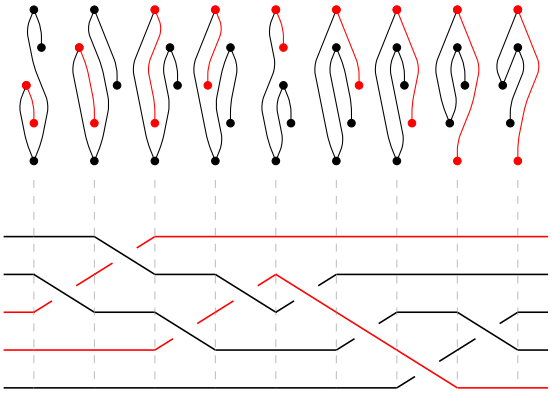
The orientation of the edges involved is preserved by the reductions so the same situation is observed in both A and B.

Consider situation (a). If the terminal layout B is not collapsible, non-final nodes are present between n and $n-1$. Some of them are on the left side of the edge connecting the final vertices and the others are on the right-hand side. Any two such nodes which are not on the same side of the final edge can be exchanged, so by appending a right reduction to r we can ensure that all the ones on the left are just below $n-1$, and all the ones on the right are just above n. Then, by adding further right exchanges, we can move these non-final nodes outside the final interval, leading to a collapsible configuration. This is illustrated in Figure 4.10a. In the situation illustrated in Figure 4.10b, we choose instead to prepend right exchanges before r: this is necessary to expel vertices nested inside the cap outside the final interval. The other cases are similar: in each of them, we can either prepend or append right exchanges to obtain a collapsible configuration. □

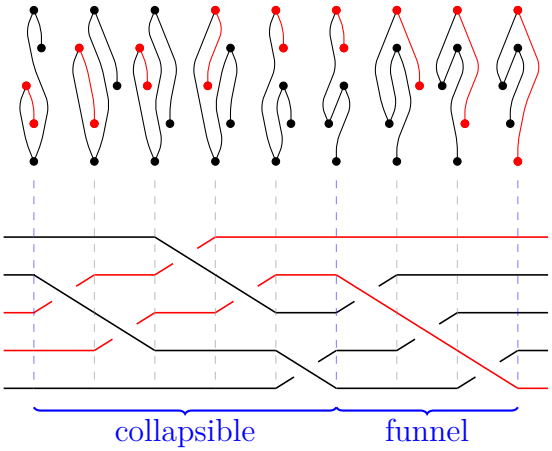
We can now show termination of right reductions. A finer analysis of the bound obtained on the length of reductions is presented in Section 4.2.3.

Theorem 4.23. *Right reductions are terminating on connected diagrams.*

Proof. We first show termination for linear diagrams. Notice that the length of a funnel reduction on a linear diagram of size n is bounded by $F(n) = O(n^2)$. This is because exchanges involving final vertices happen at most $O(n)$ times and exchanges involving only non-final vertices happen at most once per pair of non-final vertices by Lemma 4.17.



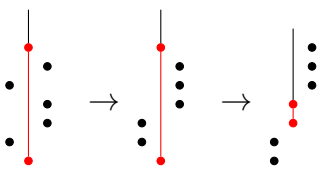
(a) Reducing a diagram to its normal form



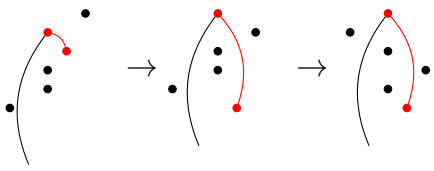
(b) Decomposition from Lemma 4.21

Figure 4.11: Decomposition into collapsible and funnel reductions

We can now define a bound $B(n)$ on the length of right reductions on linear



(a) Appending right exchanges



(b) Prepending right exchanges

Figure 4.10: Extending a reduction so that one end is collapsed

diagrams of size n, by induction on n. Consider such a reduction r. By Lemma 4.22, we can assume that one end of r is collapsible (by making r potentially longer). By Lemma 4.21, we can decompose r into a funnel part f and a collapsible part c. The collapsible part c gives rise to a collapsed reduction c', whose length is bounded by $B(n-1)$ by induction. Because an exchange involving the last vertex in the shorter diagram corresponds to two exchanges in the longer diagram, we obtain $|c| \leq 2B(n-1)$. By the observation above, $|f| \leq F(n)$. Hence, $|r| \leq 2B(n-1) + F(n) =: B(n)$. This shows termination of right reductions on linear diagrams.

We now move to the general case of connected diagrams. Assume by contradiction that there is an infinite reduction on a connected diagram. By the pigeonhole principle, there is a pair of vertices that are exchanged infinitely often. Consider a simple path between these two vertices and erase all vertices not visited by this path. The infinite reduction on the connected diagram induces an infinite reduction on the linear diagram, which contradicts termination on linear diagrams. □

Some diagrams are not connected as graphs but all their vertices are connected to a boundary. Theorem 4.23 can be extended to these cases.

Definition 4.24. *A diagram D is **boundary-connected** if it is connected or all vertices in D are connected to one of the two boundaries of the diagram.*

Figure 4.12a shows a diagram that is not connected (it has three connected components) but which is boundary-connected, since each component contains an open wire. Each vertex is therefore connected to either the top or bottom boundary of the diagram via these open wires.

(a) The original diagram D **(b)** The transformed diagram D'

Figure 4.12: Adding nodes on the boundaries to make a diagram connected

Corollary 4.25. *Right reductions on boundary-connected diagrams are terminating.*

Proof. Let D be boundary-connected. Consider the diagram D' obtained from D by adding two vertices b, t at the bottom and top boundaries, and adding two edges from b to t on each side of the diagram, as in Figure 4.12. Every edge connected to the boundary in D is connected to one of b, t in D', so D' is connected. Any right reduction on D induces a reduction of the same length on D', therefore right reductions on D terminate. □

4.2.3 Upper bound on reduction length

Beyond termination, we can use the same proof techniques to derive an asymptotic bound on reduction length. We first introduce a parametric cost on exchanges of linear diagrams:

Definition 4.26. *Given a reduction r on a linear diagram of size n and an integer w, the **cost** of r at weight w is $X + wY$, where X is the number of exchanges not involving vertex number n in r and Y is the number of exchanges involving vertex n in r.*

Lemma 4.27. *The maximum cost at weight w of a funnel with a collapsible end is $f(n, w) = O(n^2 + wn)$, where n is the length of the linear diagram.*

Proof. A funnel contains two types of exchanges. Those with final vertices account for at most $n - 2$ exchanges, because there is at most one for each non-final vertex. The ones with only non-final vertices are bounded by $O(n^2)$ as any pair of non-final vertices is exchanged at most once by Lemma 4.17. The bound follows from the definition of the cost. □

Theorem 4.28. *The maximum cost of a right reduction on a linear diagram is $O(n^3 + w \cdot n^2)$, where n is the size of the diagram.*

Proof. Let $g(n, w) = \sum_{k=1}^{n} f(k, w + n - k)$. We show that $g(n, w)$ bounds the cost of any right reduction on a linear diagram of size n. By Lemma 4.27, the desired bound will follow. We work by induction on n. For $n \leq 1$, no right

exchanges can be performed, so the bound holds. Consider a reduction $r : A \to_{\mathrm{R}}^{*} B$ on a linear diagram of size n. By Lemma 4.22, we can assume that A or B is collapsible (up to an extension which increases the cost of r). By Lemma 4.21, we can rearrange the exchanges in r to obtain a funnel and a collapsible reduction. By definition, the cost of the funnel part is bounded by $f(n, w)$. For the collapsible part, consider the reduction induced by merging the final vertices together: this gives a reduction on a diagram of size $n - 1$. Each exchange involving the last vertex in this induced reduction corresponds to an exchange of both final vertices in the original reduction, which has cost $w + 1$. Therefore, by induction, the cost of the collapsible part is bounded by $g(n - 1, w + 1)$. We therefore obtain the bound $g(n - 1, w + 1) + f(n, w) = g(n, w)$ on the cost of r at weight w. □

Theorem 4.29. *The maximum length of a reduction on a diagram of size n vertices is $O(n^3)$.*

Proof. By the same argument as Corollary 4.25 we can assume that the diagram is connected. Consider a connected string diagram D with v vertices. Pick a spanning tree on D and let D' be the string diagram obtained from D by removing all edges which are not in the spanning tree. Any reduction r_D on D induces a reduction of the same length $r_{D'}$ on D', so it is enough to bound the length of $r_{D'}$.

Pick an arbitrary vertex of D' as root for the tree and consider a depth-first search of D' from that root. This defines an envelope on the tree, which can be seen as a linear diagram L if we duplicate the nodes every time they are visited (see Figure 4.13). The length of this diagram is linear in the number of edges in D', which is linear in the number of vertices in D (since D' is a tree, it has one less edge than vertices, and its vertices are those of D).

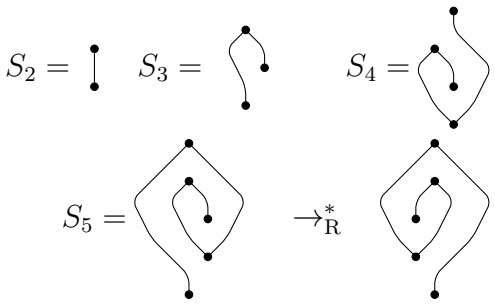
The right reduction $r_{D'}$ on D' translates to a right reduction r_L on L, where exchanging vertices x and y corresponds to exchanging all the copies of x and y in the same way. Therefore $|r_{D'}| \leq |r_L|$. By Theorem 4.28, $|r_L| = O(l^3)$ where l is the number of vertices in L. We also have that $l = O(n)$, as the linear envelope follows

each edge twice and the number of vertices in D' is one more than its number of edges. Combining this all, we obtain

$$|r_D| \leq |r_{D'}| \leq |r_L| = O(l^3) = O(n^3)$$

therefore obtaining the required bound. □

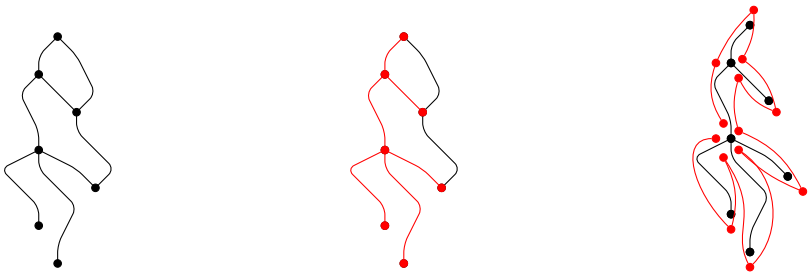
This asymptotic bound on reduction length is attained by a class of spiral-shaped diagrams:



Lemma 4.30. *For all n, the diagram S_n right reduces to its normal form in $\binom{n}{3}$ steps.*

Proof. A reduction of S_n to its normal form starts with $n-2$ exchanges of one end with the rest, followed by the reduction for S_{n-1} where the end weighs one more vertex. Therefore, the cost of a right reduction of S_n to its normal form is $s(n,w) = w(n-2) + s(n-1, w+1)$. We also have $s(2,w) = 0$ for all w. From this we obtain

$$s(n,w) = \frac{(n-1)(n-2)(n-3+3w)}{6}$$



(a) A connected diagram D **(b)** A spanning tree D' on D **(c)** A linear diagram L obtained from D'

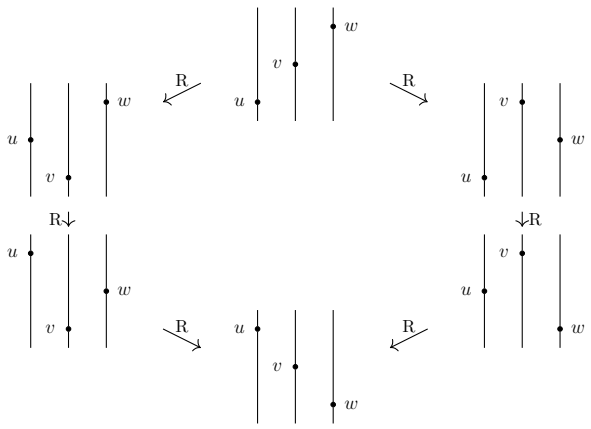
Figure 4.13: Transforming a connected diagram to a linear diagram

which gives $\binom{n}{3}$ for $w = 1$. □

4.2.4 Confluence

Lemma 4.31. *The right reduction relation is locally confluent.*

Proof. Let F, G, H be diagrams with $G_R \leftarrow F \rightarrow_R H$. If the two pairs of nodes exchanged in the two branches are disjoint, then the exchanges commute and we can close the diagram in one step: we have $H \rightarrow_{\mathrm{R}} K$ and $G \rightarrow_{\mathrm{R}} K$. Otherwise, the rewriting patterns overlap. There are nodes u, v and w in F, such that u and v are adjacent and are exchanged to obtain G, and v and w are adjacent and are exchanged to obtain H. The situation looks like this:



As u and v can be exchanged in F, there is no edge from the output of v to the input of u, and any edge going from the output of w to the input of u has to pass to the left of v. As v and w can be exchanged in F, there is no edge from the output of w to the input of v, and any edge going from the output of w to the input of u has to pass to the right of v, which is impossible by the previous observation, so there is no edge from w to u. Therefore, w and u can be exchanged both in G and H. In the resulting diagrams, we can then exchange (v, w) and (u, v) respectively, which closes the diagram. Note that the braids representation of both sides of the diagram correspond to the Reidemeister type 3 move. □

Theorem 4.32. *Right reductions are confluent and therefore define normal forms for diagrams under the equivalence relation induced by exchanges.*

Proof. By Theorem 4.29 the reduction is terminating and by Lemma 4.31 it is locally confluent, so by Newman's lemma, right reductions are confluent. Therefore, the right normal form for a given diagram can be obtained by applying any legal right exchanges until a normal form is reached. □

4.2.5 Computing normal forms

It follows from Theorem 4.32 and Theorem 4.29 that applying the right reduction rewrite strategy allows us to find normal forms in $O(v^4)$ time, where v is the number of vertices: we perform $O(v^3)$ exchanges, each of which can be found and performed in $O(v)$ time. In this section we show that this complexity can be improved, giving a procedure which constructs the normal form directly in $O(ve)$ time, where e is the number of edges.

Let D be a connected diagram in right normal form and $v \in D$ be a vertex. We analyze how a new vertex l can be added to D by connecting it to v only, such that l becomes a leaf in the new diagram. First, we need to choose whether to connect l to the domain or codomain of v. Assume for instance that we connect it to the domain of v. If v has k edges in its domain before the addition, there are $k+1$ possible positions for the new edge between l and v. Assume that such a position is chosen. The height of the vertex l in the new diagram must also be chosen, as shown in Figure 4.14. The following lemma shows that there is only one such choice such that the new diagram is in right normal form.

Lemma 4.33. *Let D' be a diagram obtained from D (where D is as above in normal form) by adding a leaf l connected to a vertex $v \in D$, at a determined side*

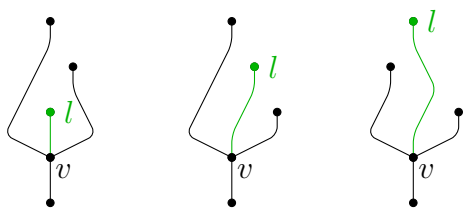


Figure 4.14: Possible vertical positions to grow a leaf l on v. Only the central diagram is in right normal form.

(domain or codomain) and position between existing edges on that side. There is a unique vertical position of l such that D' is in right normal form. Furthermore the horizontal position of l at this height is determined by the position between the existing edges of v that we grow it from.

Proof. Let us first show that there is a vertical position for l such that D' is in right normal form. First, pick an initial vertical position for l, such as the position immediately above or below v (depending on the orientation of the connection between v and l). Then, normalize by applying right exchanges. All the right exchanges involve l: otherwise, by contradiction, consider the first exchange not involving l. Removing l from its domain gives us D again (because the relative positions of vertices in D has not changed), and the exchange still applies to this diagram, which contradicts normality of D. This shows the existence of the vertical position and uniqueness follows from confluence. □

This observation already gives us a way to construct the right normal form of any acyclic connected diagram. For any tree, we can remove one leaf, compute the right normal form of the remaining tree recursively, and add the leaf at the height given by the lemma. However, this does not let us normalize cycles yet.

Definition 4.34. *A **simple face** in a string diagram is a simple edge loop whose inner region does not contain any other vertex or edge. (An edge loop is simple when no edge appears twice in the loop, and each vertex is visited at most once in the loop.)*

Definition 4.35. *Let p be an oriented path in a diagram. For each vertex v visited by p, we define the winding number of v as follows:*

Definition 4.36. *Given a simple face in a diagram D and an edge e in the face, the **mountain range** starting on e is the sequence of partial sums of winding numbers when visiting the face in direct rotation, starting from e.*

Figure 4.15 gives an example of a mountain range for an edge in a simple face. Because a cycle forms a closed loop in the plane, the winding numbers of its vertices sums up to two when visited in direct rotation. This means that a mountain range always stops two levels higher than it started.

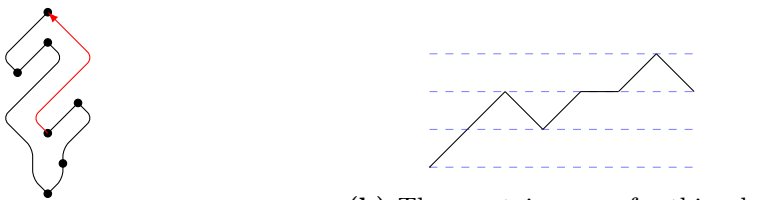
Definition 4.37. *An edge in a simple face is* ***eliminable*** *if the mountain range starting from it never reaches* 0 *after the first step.*

For instance, the edge above is eliminable, but its predecessor is not because the mountain range starts with a valley that goes at level -1 and then 0.

Lemma 4.38. *In any simple face there are exactly two eliminable edges.*

Proof. Pick an edge in the face and draw the mountain range ω for it. Let m be the minimum level it reaches after the first step. As ω starts at 0, $m \leq 1$. Consider the last edge to reach m, we will denote it by e_1. The part of ω to the right of e_1 never goes below $m+1$ by definition. When drawing the mountain range for e_1, the part of ω to the left of e_1 is shifted upwards by 2, so this part never goes below $2-m \geq 1$ when drawn as part of the mountain range for e_1. So e_1 is eliminable. Similarly, consider the last edge e_2 to reach 1 in the mountain range starting from e_1: it is also eliminable for the same reason. These are the only two edges which satisfy the criterion. □

Lemma 4.39. *Let D be a connected diagram in right normal form and e be an eliminable edge in a simple face of D. Then the diagram D' obtained from D by removing e is in right normal form.*



(a) An edge in a face

(b) The montain range for this edge

Figure 4.15: Example of a chosen edge in a face and its mountain range

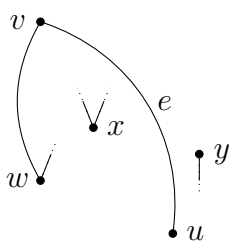
Proof. Consider such an edge. We first analyze what it means to be eliminable in geometrical terms. Let us call u the starting point of e and v its end point. We know that e is immediately followed by a left turn (winding number $+1$) at v. The next vertex where a rotation happens w also has winding number $+1$ (otherwise the number of rotations from e to the edge after w would be null). By symmetry let us assume that e points upwards when travelling in the direct orientation on the face.

There are three sorts of right exchanges that could potentially be enabled by removing e.

Exchanging u and v The first one would be exchanging the endpoints of e together, but this is impossible because of the left turn on v which imposes a horizontal ordering: no such right exchange can be made.

Exchanging u or v with another vertex x The second one would be exchanging one of the endpoints of e with another vertex. This other vertex must be in the interval between the endpoints (otherwise the exchange was already possible before). That is not possible for v because of the left turn on this vertex. For u, this would require having another vertex x immediately to the left of e with no edge linked from below. We will see in a later paragraph that this is not possible.

Exchanging two vertices x, y distinct from u and v Finally, the third case consists in exchanging two nodes x and y between u and v, x immediately to the left of e with no edge linked from below, and y immediately to the right of e with no edge from above. We will show that no such x exists.



Ruling out the existence of x Because e is the right boundary of the face, such an x must be a part of the boundary of the face. As part of this cycle, it has two edges coming from above. Browsing the cycle in the direct orientation can visit x in two directions: from left to right or from right to left.

If x is visited from left to right, this contradicts the fact that x is immediately to the left of e, because the interior of the face is contained between the two edges linked to x.

If x is visited from right to left, consider the path from w to x. It starts upwards and ends downwards, so it has odd winding number. As x itself is a right turn, this number cannot be negative: otherwise, travelling from e to the edge following x would have null or negative winding number, contradicting the assumption that e is eliminable. So, the path from w to x has positive winding number, and therefore one edge in this path is located between x and e, which contradicts the fact that x is immediately to the left of e. □

Theorem 4.40. *The right normal form of a boundary-connected string diagram in free monoidal categories can be computed in time $O(ve)$ where v is the number of vertices and e is the number of edges.*

Proof. Again we can restrict our attention to the case of connected diagrams thanks to the reduction of Figure 4.12. We construct the right normal form of any connected string diagram by induction on the number of edges. The initial case (no edge) is clear.

Given a diagram D, there are two cases. We can check in $O(v)$ if D has a leaf, in which case we remove this leaf and obtain a diagram D' with one less edge that we can inductively normalize. Then, by Lemma 4.33, we can deduce the right normal form for D, by inserting back the leaf at the unique spot which makes the diagram normalized. Such a spot can be found in $O(v)$ by applying right exchanges on the leaf as long as they are admissible. If D does not have any leaf, then it contains a cycle (acyclic graphs necessary have leaves) and so it has a simple face (pick a cycle and iteratively narrow it until it is a simple face). In that case, by Lemma 4.38,

there are two eliminable edges in this face. These can be identified in $O(v)$ thanks to the characterization via mountain ranges. We can remove one of them, obtaining diagram D'', and inductively normalize D''. By uniqueness of the normal form for D'' and by Lemma 4.39, the normal form for D'' can be obtained by normalizing D and then removing the edge. So the normal form for D can be reconstructed from the normal form for D'' by adding the edge back. This can also be computed in $O(v)$. We therefore obtain a normalizing algorithm with e induction steps, each of which takes $O(v)$ time, so the overall complexity is $O(ve)$. □

4.2.6 Extension to disconnected diagrams

The connectivity requirement is crucial to obtain termination of right reductions and therefore the right normal forms on which we relied on for our results. In this section, we extend our results to arbitrary diagrams. Our approach is to define a complete invariant for the exchange rule.

In general, a diagram can contain multiple connected components. Because we are dealing here with non-symmetric monoidal categories, the way these components nest into each other's faces matters as this tree structure is preserved by exchanges.

As the transformation described in Figure 4.12 is still applicable to disconnected diagrams, we consider a diagram D without input or output edges ($S(D) = 0$ and $W(D, N(D)) = 0$).

Definition 4.41. *For each level $h \in [\![N(D)+1]\!]$, we define symbols $(s_{h,k})_{0\leq k\leq D.N(h)}$. The symbol $s_{h,k}$ is the **spot** at height h and position k, which represents the empty space between the kth and $k+1$th edge at level h (including diagram boundaries for the extrema).) Similarly, we define symbols $(p_{h,k})_{1\leq k\leq D.N(h)}$. The symbol $p_{h,k}$ represents the intersection of the kth edge that crosses level h and level h itself: we call this a **place**. Vertices of the diagram are places too, and we represent the vertex between heights h and $h+1$ as v_h.*

Definition 4.42. *Two spots $s_{h,k}$ and $s_{h+1,k'}$ are **adjacent** when either $k = k'$ and $H(D,h) \geq k$ or $k + \Delta(D,h) = k'$ and $H(D,h) + I(D,h) \leq k$.*

Graphically, two spots are adjacent if they lie in neighbouring slices and they are in the same region of the diagram seen as a planar graph. We will formally define this notion of region by taking the connected closure of this adjacency relation.

Definition 4.43. *A **face** is a connected component of spots for the adjacency relation defined above.*

Definition 4.44. *Two places $p_{h,k}$ and $p_{h+1,k'}$ at consecutive levels are adjacent if they are on the same edge. Formally, this happens when either $k = k'$ and $H(D,h) > k$, or $k + \Delta(D,h) = k'$ and $H(D,h) + I(D,h) \leq k$. $H(D,h) \leq k < H(D,h) + I(D,h)$ and $H(D,h) \leq k' < H(D,h) + O(D,h)$. A place $p_{h,k}$ and a vertex v'_h are adjacent when Two places $p_{h,k}, p_{h,k'}$ at the same level are adjacent if $H(D,h) \leq k, k' < H(D,h) + I(D,h)$ or $H(D,h-1) \leq k, k' < H(D,h-1) + O(D,h-1)$.*

Definition 4.45. *A **component** is a connected component of places for the adjacency relation defined above.*

Lemma 4.46. *Any exchange $D \to D'$ induces bijections ϕ_F and ϕ_C between the faces and components of D and D'.*

Proof. Let n and $n+1$ be the levels exchanged. By symmetry we can assume it is a right exchange. Let us define ϕ_C by mapping each spot in D to a spot in D', such that the adjacency relation is respected. Let $s_{h,k}$ be a spot. If $h \leq n$ or $h > n+1$ (the spot lies in a slice that is untouched by the exchange) then $\phi_C(s_{h,k}) = s_{h,k}$. Otherwise, $h = n+1$. If $k \leq H(D, n+1)$ (the spot lies to the left of both nodes exchanged) then $\phi_C(s_{n+1,k}) = s_{h,k}$ again. If $k > H(D,n) + I(D,n)$ (the spot lies to the right of both nodes exchanged) then $\phi_C(s_{n+1,k}) = s_{h,k-\Delta(D,n)+\Delta(D,n+1)}$. If $k > H(D,n+1)$ and $k < H(D,n+1) + I(D,n+1)$ (the spot lies in one of the input branches of the node at height $n+1$) then $\phi_C(s_{n+1,k}) = s_{n,k}$ (the spot just above). Similarly if $k > H(D,n)$ and $k < H(D,n) + O(D,n)$ then $\phi_C(s_{n+1,k}) = s_{n+2,k+\Delta(D,n+1)}$ (the spot just below). Finally, if $k \geq H(D,n+1) + I(D,n+1)$ and $k \leq H(D,n)$ then $\phi_C(s_{n+1,k}) = s_{n,k-\Delta(D,n)+\Delta(D,n+1)})$. In each of these cases one can check that ϕ_C preserves the adjacency relationship for spots. The mapping for places ϕ_F can be defined similarly. □

Definition 4.47. *Given a level h, a spot $s_{h,k}$ and a place $p_{h,k'}$ are* ***neighbours*** *if $k = k'$ or $k + 1 = k'$. Furthermore, spots $s_{h,0}$ and $s_{h,W(D,h)}$ are* ***neighbours of the boundary****.*

Lemma 4.48. *There is a unique face containing spots which are neighbours of the boundary. We denote it by f_0.*

Proof. Any neighbour of the boundary is adjacent to the neighbours of the boundary above and below it. By assumption, there is only one spot at the source and target levels. Therefore, all neighbours of the boundary are connected together. □

Definition 4.49. *A component c neighbours a face f when there is a place $p \in c$ neighbouring a spot $s \in f$. We denote it by $c \diamondsuit f$.*

Neighbourhood is preserved by exchanges, in the following sense:

Lemma 4.50. *Let d, d' be diagrams where d' is obtained from d by exchanges. The bijections between faces and components of d and d' induced by the exchanges respect the neighbourhood relation.*

Proof. It suffices to show that any neighbourhood relation that holds between slices affected by an exchange also holds in the exchanged diagram. This can be achieved by simple inspection of the definition of neighbourhood and adjacency on spots and places. □

Definition 4.51. *Given a level h, two spots $s_{h,k}$ and $s_{h,k'}$ are* ***over-connected*** *if they are connected by a path of spots which never go below level h. Similarly, over-connectivity is defined for places too.*

Note that the term *over* should be understood visually: this means that the path goes through levels whose numerical indices are actually lower than h, not higher. Figure 4.16 shows an example of diagram where connected vertices are not over-connected. Over-connectivity is an equivalence relation on spots at the same level, and similarly for places.

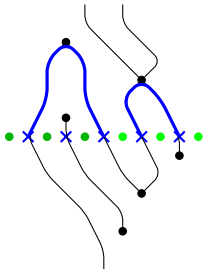


Figure 4.16: Over-connectivity on an example. Each blue path shows stays above the level h so each of them witnesses that their start and end points are over-connected. However, the start and end of the first blue path are not over-connected to the start and end of the second blue path, although they are connected.

Lemma 4.52. *Given a level h, assume that two distinct spots $s_{h,k}$ and $s_{h,k'}$ are over-connected. Furthermore, assume that $\forall k < i < k'$, $s_{h,i}$ is over-connected to neither $s_{h,k}$ nor $s_{h,k'}$. Then the places $p_{h,k}$ and $p_{h,k'-1}$ are over-connected.*

Proof. By induction on the distance h from the top of the diagram. The property holds trivially for the topmost slice as no two spots are over-connected at this level.

Assuming it holds at level h and consider spots $s_{h+1,k}$ and $s_{h+1,k'}$ over-connected. If they are both connected to the same spot $s_{h,k}$ then all places between them are over-connected, so the result holds. Otherwise, they are connected to different spots s_{h,k_0} and s_{h,k'_0} respectively which are over-connected. Let $s_{h,i_1}, \ldots, s_{h,i_q}$ be the spots between s_{h,k_0} and s_{h,k'_0} which are over-connected to s_{h,k_0} and s_{h,k'_0}. Each of these spots are not connected to any spot at level $h+1$, so the places neighbouring them are all connected via the same morphism between level h and $h+1$. In particular, p_{h,i_1-1} and p_{h,i_q} are neighbours.

Apply the induction hypothesis to the pairs (s_{h,k_0}, s_{h,i_1}) and (s_{h,i_q}, s_{h,k'_0}): p_{h,k_0} and p_{h,i_1-1} are over-connected and so are p_{h,i_q} and p_{h,k'_0-1}. Therefore p_{h,k_0} and p_{h,k'_0-1} are over-connected. □

Lemma 4.53. *Let f be a face, $f \neq f_0$. There is a component c such that for each level h containing a spot in f, the place to the left of the first spot in f at level h and the place to the right of the last spot in f at level h are both in c. Such a component c is therefore unique. We say that c **encloses** f, denoted by $f \prec c$.*

Proof. First, consider the highest level h_0 where f occurs. All spots in f at h_0 are not connected to any spot at the higher level, so their neighbouring places are all connected to the morphism above. Let c be their common component. For any further level h we prove the result by induction. Consider the first and last spots $s_{h,k}$, $s_{h,k'}$ which belong to f at level h. We show that $p_{h,k-1}$ belongs to c. Showing that so does $p_{h,k'}$ is similar. Let s_{h-1,k_0} be the leftmost spot neighboured by $s_{h,k}$ at level $h-1$. If s_{h-1} is the first spot s_{h,k_0} in f at $h-1$, then by induction p_{h-1,k_0-1} belongs to c and is connected to $p_{h,k-1}$, so $p_{h,k-1}$ belongs to c. Otherwise, let $s_{h-1,i_1}, \ldots, s_{h-1,i_p}$ be the spots in f to the left of s_{h-1,k_0}. We can apply Lemma 4.52 to s_{h-1,i_p} and obtain that p_{h-1,i_p} and p_{h-1,k_0-1} are connected. Furthermore, the $s_{h-1,i_1}, \ldots s_{h-1,i_p}$ are not neighbours of any spots at level h, so the edges separating them are all connected to the same vertex between h and $h-1$. So p_{h-1,i_p} and p_{h-1,i_1-1} are neighbours, and finally p_{h-1,k_0-1} and p_{h-1,i_1-1} are connected. By induction, p_{h-1,i_1-1} belongs to c, so so does p_{h-1,k_0-1}. □

Definition 4.54. *Given a face f, a spot $s \in f$ is maximal if it is at the highest level where spots of f occur. Similarly, it is minimal if it is at the lowest level where spots of f occur.*

Note that a face can have multiple maximal or minimal spots.

Lemma 4.55. *Maximal and minimal spots in a face only neighbour the enclosing component, or the boundary in the case of the root face.*

Proof. This is a direct consequence of the proof of Lemma 4.53. □

Definition 4.56. *Let f be a face. For each component c neighbour of f such that c does not enclose f, we say that f encloses c, denoted by $c \prec f$.*

The enclosure relation is preserved by exchanges as follows:

Lemma 4.57. *ϕ_F and ϕ_C respect $\prec$, i.e. $f \prec c \Leftrightarrow \phi_F(f) \prec \phi_C(c)$ and $c \prec f \Leftrightarrow \phi_C(c) \prec \phi_F(f)$.*

Proof. By Lemma 4.50 and because $c \prec f \Leftrightarrow c \diamondsuit f \wedge \neg(f \prec c)$, it is enough to show preservation of $f \prec c$.

Let $d \to_{\mathrm{R}} d'$ be a right exchange, f be a face in d enclosed by c. Let $s_h \in f$ be a maximal spot in f, and $s_l \in f$ be a minimal spot in f. If s_h is untouched by the exchange, then it is still maximal in $\phi_F(f)$, and $\phi_C(c)$ is still the only component neighboured by $\phi_F(f)$ at s_h's slice, so we have $\phi_F(f) \prec \phi_C(c)$. Similarly, if s_l is untouched by the exchange, $\phi_F(f) \prec \phi_C(c)$. If both s_l and s_h are touched by the exchange, then they are equal in d and f neighbours only c in d. By Lemma 4.50, $\phi_F(f)$ neighbours only $\phi_C(c)$. As $\phi_F(f)$ is not the root face in d', $\phi_F(f) \prec \phi_C(c)$. □

We next introduce an order on the faces enclosed by a component c. Let $N(c)$ be the right normal form of c, seen as a standalone diagram. The right reduction from c to $N(c)$ induces a bijection between the faces of c and those of $N(c)$.

Given two faces f, f' in $N(c)$, consider the leftmost maximal spots $s_{h,k}, s_{h',k'}$ of f and f'. We order f and f' by lexicographic order on the pairs $(h,k),(h',k')$. This defines an order $<$ on faces of $N(c)$ and therefore on faces of c.

Definition 4.58. *We inductively define the* ***structural tree*** *of faces and components. Given a face f, $T(f) = \{T(c) | c$ component enclosed by $f\}$. Given a component c, let $f_1, \dots, f_n$ be the set of faces enclosed by c, ordered with the order defined above. We set $T(c) = (N(c), T(f_1), \dots, T(f_n))$. Finally, the structural tree $T(D)$ of the entire diagram is $T(f_0)$.*

To make sure that this tree is finite, we must make sure that none of its nodes is a child of itself.

Lemma 4.59. *$\prec$ is well-founded.*

Proof. A diagram contains a finite number of components and faces. It is therefore enough to show that given a component c, it is impossible that $c \prec \cdots \prec c$. We first show that if $c \prec f \prec c'$, then at each level h where f appears, then for any place $p_{h,k} \in c$ there are places $p_{h,a}, p_{h,b} \in c'$ with $a < k < b$. This is a simple consequence of Lemma 4.53. Then, by induction, we extend this to the transitive closure of $\prec$, which shows the result. □

Lemma 4.60. *$T(D)$ is invariant under exchanges.*

Proof. By Lemma 4.57, $\prec$ is invariant under exchanges. The order on the faces enclosed by a given component is also invariant as it is defined on the right normal form of the component. □

Completeness of the structural tree

We show that the structural tree $T(D)$ of a diagram is a complete invariant for exchanges:

Theorem 4.61. *Two diagrams D, D' with no inputs and outputs are equivalent if and only if $T(D) = T(D')$.*

If D and D' are equivalent then $T(D) = T(D')$ by Lemma 4.60. To prove the converse, we introduce a few notions of diagram surgery, to manipulate components and faces.

Definition 4.62. *Given a diagram D and a spot $s \in D$, the **injection** of a closed diagram D' at s, denoted by $I_s^D(D')$, is obtained by inserting D' in place of s in D. Concretely, this means that the vertices of D' are inserted at the slice of s, shifted to the right by the number of edges to the left of s. Formally, $I_s^D(D')$ is defined as follows:*

$$S(I_s^D(D')) = S(D)$$

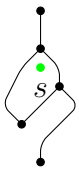
$$N(I_s^D(D')) = N(D) + N(D')$$

$$H(I_s^D(D'), h) = \begin{cases} H(D, h) & \text{if } h < a \\ H(D', h - a) + k & \text{if } a \leq h < b \\ H(D, h - N(D')) & \text{if } b \leq h \end{cases}$$

$$I(I_s^D(D'), h) = \begin{cases} I(D, h) & \text{if } h < a \\ I(D', h - a) & \text{if } a \leq h < b \\ I(D, h - N(D')) & \text{if } b \leq h \end{cases}$$

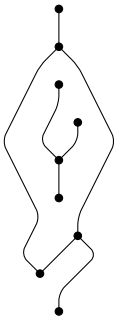
$$O(I_s^D(D'), h) = \begin{cases} O(D, h) & \text{if } h < a \\ O(D', h - a) & \text{if } a \leq h < b \\ O(D, h - D'.N) & \text{if } b \leq h \end{cases}$$

where $s = s_{a,k}$ and $b = a + N(D')$.



(a) The diagram D with a spot $s \in D$

(b) The diagram D'



(c) The diagram $I_s^D(D')$

Figure 4.17: Injection of a diagram in a face.

By abuse of language, if $x \in D'$ is a place, spot, face or component, then we denote again by x the corresponding place, spot, face or component in $I_s^D(D')$, as this is unambiguously defined.

Lemma 4.63. *Let D be a diagram and $s, s' \in D$ be spots in the same face $f \in D$. For all closed diagram D', $I_s^D(D') \simeq I_{s'}^D(D')$.*

Proof. Let us assume s and s' are adjacent. Let v be the vertex between the two slices containing s and s'. The vertices of D' in $I_s(D')$ can be successively exchanged with v, leading to $I_{s'}^D(D')$, which shows that $I_s^D(D') \simeq I_{s'}^D(D')$. By induction, this can be repeated for any adjacency path between two spots in the same face. □

Bearing in mind that this is only defined up to exchange, we can therefore write $I_f^D(D')$ to inject D' anywhere in the face f.

Lemma 4.64. *Injection respects exchanges on the outer and inner diagrams: If $D \simeq D'$ and $C \simeq C'$, then for any face $f \in D$ and its corresponding face $f' \in D'$, $I_f^D(C) \simeq I_{f'}^{D'}(C')$.*

Proof. Any exchange $C \to_R C'$ translates into a single exchange $I_f^D(C) \to_R I_f^D(C')$. So, by induction, if $C \simeq C'$, then $I_f^D(C) \simeq I_f^D(C')$. To show that injection respects equivalence on the outer diagram, let $s \in f$ and consider a single rewriting step

$D \to_R D'$. If s is not in the slice between the two vertices u and v being exchanged in D, then it corresponds to a spot $s' \in D'$. We have $s' \in f'$ and $I_s^D(C) \to_R I_{s'}^{D'}(C)$ in one step again. Otherwise, $N(D')$ exchanges are required to move u past D', one to exchange u and v, and $N(D')$ again to move v past D'. So $I_s^D(C) \simeq I_s^{D'}(C)$. So injections are compatible with exchanges both on the inner and outer diagram. □

Definition 4.65. *Let D be a diagram and $c \in D$ be a component. The **erasure** of c in D, denoted by $D - c$, is the diagram obtained by removing from D any vertex from c or its sub-components.*

Lemma 4.66. *Let D be a diagram and $c \in D$ be an acyclic component. Then there is a face $f \in D - c$ such that $D \simeq I_f^{D-c}(c)$.*

Proof. Pick a vertex $r \in c$: we will consider c as a tree rooted in r. By induction on this tree, we are going to gather all vertices around r, meaning that the heights of these vertices in the diagram form an interval.

Say that a vertex $v \in c$ is collapsed if the set of diagram heights of the vertices in its subtree form an interval. For any $v \in c$, we show that D is equivalent to a diagram D' where v is collapsed and such that the vertical order of vertices which are outside this subtree is preserved in D'.

If v is a leaf, it is always collapsed. Consider the case where v has children $u_1, \ldots, u_n$. Because c is acyclic, it is possible to exchange each child u_i with any vertex on a slice between u_i and v, so we can assume that the vertical positions of v and $u_1, \ldots, u_n$ form an interval. By induction, each u_i can be successively collapsed without changing the vertical order of vertices which are not in the subtree of u_i. Once this is done, the vertical position of vertices in the subtrees of the u_i and v form an interval, so v is collapsed. By doing so we have preserved the vertical ordering of vertices outside the subtree of v.

Therefore, there is a D' where r is collapsed. Let c' be the component corresponding to c in D'. Let $f' \in D'$ be the face enclosing c' in D'. We have $I_{f'}^{D'-c'}(c') = D'$. By invariance of injection up to exchanges (Lemma 4.64), $I_{f'}^{D'-c'}(c') \simeq I_f^{D-c}(c)$ for some the corresponding face $f \in D - c$. □

Definition 4.67. *Let D be a diagram and $f \in D$ be a face that does not neighbour the boundary. The* ***erasure*** *of f in D, denoted by $D - f$, is the diagram obtained by removing all spots in f and descendant faces. Formally, let $P(h, i, j) = |\{s_{h,k} | i \leq k < j, s_{h,k} \in f' \prec^* f\}|$. Then $D - f$ is defined as follows:*

$$S(D - f) = S(D)$$

$$N(D - f) = N(D)$$

$$H(D - f, h) = H(D, h) - P(h, 0, H(D, h) + 1)$$

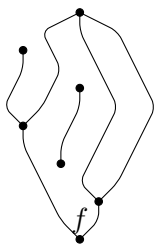
$$I(D - f, h) =$$

$$\begin{cases} 1 \text{ if } I(D,h) = 0 \text{ and } P(h, H(D,h), H(D,h)) = 1 \\ I(D,h) - P(h, H(D,h) + 1, H(D,h) + I(D,h) + 1) \\ \text{otherwise} \end{cases}$$

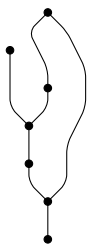
$$(D - f).O(h) =$$

$$\begin{cases} 1 \text{ if } O(D,h) = 0 \text{ and } P(h+1, H(D,h), H(D,h) + 1) = 1 \\ O(D,h) - P(h+1, H(D,h) + 1, H(D,h) + O(D,h) + 1) \\ \text{otherwise} \end{cases}$$

One can check that this defines a valid diagram.



(a) A face f in diagram D



(b) The diagram $D - f$

Lemma 4.68. *Let $s \in D$ be a spot. Then for any closed diagram c and face $f \in c$, $I_s^D(c - f) = I_s^D(c) - f$.*

Proof. This can be checked directly from the definitions of injections and erasures. □

Lemma 4.69. *Let D be a diagram and $f \in D$ be a face. For any diagram $D' \simeq D - f$, there is a diagram $D_0 \simeq D$ such that $D' = D_0 - f$.*

Proof. By induction on the length of the equivalence between D' and $D - f$, consider an exchange on $D - f$, exchanging vertices at height n and $n+1$. By symmetry we can assume it is a right exchange, and therefore $H(D-f,n) \geq H(D-f,n+1)+I(D-f,n+1)$. We show that $H(D,n) \geq H(D,n+1)+I(D,n+1)$, enabling a right exchange at the same heights in D. Applying this right exchange will give us D_0.

Let us first show that $H(D,n) \geq H(D,n+1)$. Indeed, assume by contradiction that $H(D,n) < H(D,n+1)$. We have $P(n+1,0,H(D,n+1)+1) = P(n+1,0,H(D,n)+1) + P(n+1,H(D,n)+1,H(D,n+1)+1)$. Because the wires to the left of $H(D,n)$ run undisturbed at levels n and $n+1$, we also have $P(n+1,0,H(D,n)+1) = P(n,0,H(D,n)+1)$. Therefore $P(n+1,0,H(D,n+1)+1) = P(n,0,H(D,n)+1) + P(n+1,H(D,n)+1,H(D,n+1)+1)$. Furthermore, $P(n+1,H(D,n)+1,H(D,n+1)+1) \leq H(D,n+1) - H(D,n)$, as can be seen from the definition of P. Combining these, we get:

$$\begin{aligned} H(D-f,n+1) &= H(D,n+1) - P(n,0,H(D,n)+1) - P(n+1,H(D,n)+1,H(D,n+1)+1) \\ &\geq H(D,n+1) - P(n,0,H(D,n)+1) - (H(D,n+1) - H(D,n)) \\ &= H(D-f,n) \end{aligned}$$

So $H(D-f,n+1) \geq H(D-f,n)$ which contradicts the exchangeability assumption in $D-f$. Hence $H(D,n) \geq H(D,n+1)$. We can therefore decompose $P(n,0,H(D,n)+1) = P(n,0,H(D,n+1)+1)+P(n,H(D,n+1)+1,H(D,n)+1)$.

Now, let us show that $H(D,n) \geq H(D,n+1) + I(D,n+1)$. By contradiction again, we assume that $H(D,n) < H(D,n+1) + I(D,n+1)$. Then, substracting $P(n+1,0,H(D,n+1)+1)$ from both sides we get

$$H(D,n) + P(n+1,0,H(D,n+1)+1) < H(D-f,n+1) + I(D,n+1)$$

. By the decomposition above this is equivalent to

$$H(D-f,n) + P(n,H(D,n+1)+1,H(D,n)+1) < H(D-f,n+1) + I(D,n+1)$$

And by the fact that $I(D, n+1) \leq I(D-f, n) + P(n, H(D, n+1)+1, H(D, n)+1)$, we get

$$H(D-f, n) \geq H(D-f, n+1) + I(D-f, n)$$

. This shows that the right exchange is admissible on D and completes the proof. □

Lemma 4.70. *Let D be a diagram and $c \in D$ be an arbitrary component (not necessarily acyclic this time). Then there is a spot $s \in D - c$ such that $D \simeq I_s^{D-c}(c)$.*

Proof. Let $f_1, \ldots, f_k$ be all the faces enclosed by c. Consider $D' = D - f_1 \cdots - f_k$. Let c' be the component corresponding to c in D': as a diagram, $c' = c - f_1 \cdots - f_k$. As c' does not enclose any face, c' is acyclic. By Lemma 4.66, we can gather c' in one spot: there is $s \in D' - c'$ such that $D' \simeq I_s^{D'-c'}(c')$. (In fact, because all the faces removed from D to obtain D' are enclosed by c, $D' - c' = D - c$.) Then, by Lemma 4.69, there is a $D_0 \simeq D$ such that $D_0 - f_1 \cdots - f_k = I_s^{D'-c'}(c')$. By Lemma 4.68, $I_s^{D'-c'}(c') = I_s^{D'-c'}(c - f_1 \cdots - f_k) = I_s^{D'-c'}(c) - f_1 \cdots - f_k$. Therefore, we obtain $D_0 - f_1 \cdots - f_k = I_s(c) - f_1 \cdots - f_k$ and finally $D_0 = I_s^{D'-c'}(c) = I_s^{D-c}(c)$. □

We can now prove Theorem 4.61, showing the completeness of the structural tree for exchanges.

Proof. Let C, D be diagrams such that $T(C) = T(D)$.

To each node n of $T(C)$ we can associate diagrams C_n (respectively D_n) obtained by erasing vertices not contained in the subtree below n in C (respectively D). We show by induction on n that $C_n \simeq D_n$.

If n is a leaf face node, then both C_n and D_n are empty diagrams and are therefore equivalent. If n is a leaf component node, then both C_n and D_n are acyclic connected diagrams with identical right normal forms, so they are equivalent.

If n is an internal face node, let $\{c_1, \ldots, c_m\}$ be its child components. By induction their corresponding diagrams in C and D are pairwise equivalent. We can apply Lemma 4.70 for each of them and express both C and D as iterated injections of the c_i in the empty diagram: therefore $C_n \simeq D_n$.

If n is an internal component node, let $(f_1, \ldots, f_m)$ be its child faces. Again, by induction their corresponding diagrams in C and D are pairwise equivalent. Moreover, the components corresponding to n in C and D have the same right normal form F. We can therefore obtain both C and D as iterated injections of the f_i in the faces of F, in the designated order. Therefore $C_n \simeq D_n$, which completes the proof. □

Word problem

We show how to compute the structural tree of a diagram, and therefore solve the word problem in the general case. Algorithm 1 scans the diagram in one pass and computes simultaneously the components and faces of the diagram, as well as the inclusion relation between them. Components and faces are defined as equivalence classes of *places* and *spots* under an adjacency relation, so we use two union-find data structures to represent them.

Algorithm 1 Algorithm to compute the faces, components, and relations between them.

initialize union-find data structures F for faces and C for components
initialize parent pointer arrays PF for faces and PC for components
for $h = 0$ to $N(D)$ **do**
  for $k = 0$ to $W(D, h)$ **do**
    if $s_{h,k}$ adjacent to $s_{h-1,k}$ and to $s_{h-1,k-\Delta(D,h-1)}$ **then**
      UNION$(F(h-1,k), F(h-1,k-\Delta(D,h-1)))$
      $F(h,k) \leftarrow F(h-1,k)$
    else if $s_{h,k}$ adjacent to $s_{h-1,k}$ only **then**
      $F(h,k) \leftarrow F(h-1,k)$
    else if $s_{h,k}$ adjacent to $s_{h-1,k-\Delta(D,h-1)}$ only **then**
      $F(h,k) \leftarrow F(h-1,k-\Delta(D,h-1))$
    else
      $F(h,k) \leftarrow$ a fresh face id
      $PF(h,k) \leftarrow (h-1, H(D,h-1))$
    end if
  end for
  for $k = 0$ to $W(D,h) - 1$ **do**
    Update components similarly
  end for
end for

Unions of faces are performed when scanning vertices with no output. Each of them costs $O(\log^* f)$, where f is the number of faces of the diagram and $\log^*$ is the log-star function. So total cost of all unions of faces is $O(v \log^* f)$. Scanning the diagram with the two loops takes $O(ve)$ operations, and checking if two spots or places are adjacent takes constant time. Therefore, the computation of the faces, components and their relations can be done in $O(ve)$.

Then, we apply the algorithm of Theorem 4.40 to compute the right normal form of each component, which can be done again in quadratic time.

Finally, the structural tree of the diagram is converted to an integer recursively in Algorithm 2, where we assume a coding function χ injectively mapping any tuple of integers to an integer (using an appropriate encoding). As the structural tree is a complete invariant for diagram equivalence, we obtain the following theorem.

A note of caution about this last step is that it does rely pretty crucially on the encoding of large data structures into integers and exploits the fact that the computational model lets us compare unbounded integers for free in constant time, which could be unrealistic for practical purposes. For implementations, this can be mitigated using hashing techniques which let compare large data structures quickly on average, rejecting different structures quickly and only resorting to full comparison when the hashes are equal.

Algorithm 2 Algorithm to recursively compute an integer representation for a structural tree.

if n is a face node **then**
  compute the integer representation of its children components recursively;
  sort the list of children components as l **return** $\chi(l)$
end if
if n is a component node with normalized root component c **then**
  sort the children faces by order of introduction in the normalized component c
  compute the integer representation of the children faces recursively as l, preserving the order; **return** $\chi(c, l)$
end if

Theorem 4.71. *The word problem for string diagrams in a monoidal category can be solved in $O(ve)$, where v is the number of vertices and e is the number of edges.*

4.2.7 Linear-time solution to the word problem in the connected case

In this section we show how the word problem can be solved in linear time for boundary-connected diagrams via a reduction to the problem of map isomorphism. In the disconnected case, the components enclosed in a face can spin around each other, so comparing two faces amounts to comparing their sets of components. Therefore, there is little hope to extend this result to the disconnected case.

We first recall some background notions of topological graph theory. We refer the interested reader to (Mohar and Thomassen, 2001) for a more in-depth treatment of these notions.

Background on planar maps

A multigraph is a set of vertices V and of edges E where each edge $e \in E$ is associated with a set of one or two vertices $V(e)$. In other words it is an undirected graph where multiple edges can exist between two vertices, and loops are allowed.

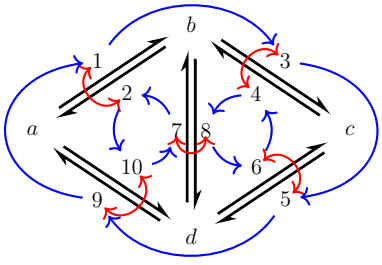
A planar map is a discrete representation of the embedding of a connected multigraph (seen as a topological space) in a surface.

Definition 4.72. *A **map** is a set Ω of **darts** (or half-edges) and two permutations x and y of Ω such that $x^2 = 1$, x has no stationary point, and the permutation group G generated by x and y is transitive (for any $a, b \in \Omega$ there is $g \in G$ such that $g(a) = b$).*

$$x = (1\ 2)(3\ 4)(5\ 6)(7\ 8)(9\ 10)$$

$$y = (2\ 10\ 7)(4\ 8\ 6)(1\ 3\ 5\ 9)$$

(a) A planar map given by two permutations



(b) Graphical representation where darts are numbered half-edges

Two maps are isomorphic when there is a bijection between their sets of darts respecting the permutations x and y of both maps.

In a map m, the cycles of x are called edges of m. The cycles of y are called faces of m. The cycles of xy are called vertices. The Euler characteristic of m is

$$\chi(m) = v - e + f$$

where v is the number of vertices, e of edges and f of faces. A map m is *planar* if $\chi(m) = 2$.

Any embedding of a multigraph in the plane gives rise to a planar map.

Theorem 4.73. *Jones and Singerman (1978) Any two embeddings of a multigraph in the plane are isotopic if and only if the corresponding planar maps are isomorphic.*

Theorem 4.74. *Hopcroft and Wong (1974) Determining if two planar maps are isomorphic can be decided in linear time.*

Again, the same note of caution about the comparison of unbounded integers in constant time applies to the latter result, but hashing should provide a satisfactory implementation in practice.

Our goal is to reuse this last result to solve the word problem for connected string diagrams. However, the word problems for string diagrams and for planar maps do not match: Figure 4.20 shows two string diagrams which are isotopic as planar maps but not equivalent as string diagrams.

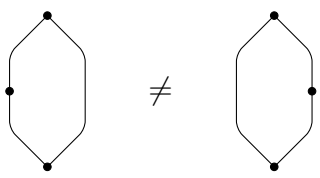


Figure 4.20: Two non-equivalent string diagrams which are isotopic as maps

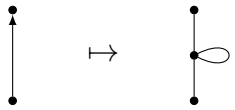
Directed planar maps

Maps are embeddings of undirected multigraphs. In this section, we introduce an analogous notion for directed multigraphs. A directed multigraph is a set of vertices V and a set of edges E, each edge being associated to a pair of vertices (s, t) (its source and target). A directed multigraph is connected if it is connected as an undirected multigraph.

Definition 4.75. *A directed map is a map (Ω, x, y) together with a choice of distinguished darts $D \subseteq \Omega$ such that exactly one dart in each cycle of x belongs to D.*

Two directed maps are isomorphic when they are isomorphic as maps and furthermore the bijection respects the distinguished darts. Similarly to Figure 4.20, there are directed maps which are isomorphic as undirected maps but not as directed maps.

Given a directed planar map M, we can define a planar map $\iota(M)$ by replacing each directed edge by an undirected graph which encodes the direction of the original edge:



Proposition 4.76. *Two directed planar maps M,M' are isomorphic if and only if the undirected planar maps $\iota(M)$ and $\iota(M')$ are isomorphic.*

Proof. If M and M' as isomorphic then clearly so are $\iota(M)$ and $\iota(M')$. Conversely, assume that $\iota(M)$ and $\iota(M')$ are isomorphic maps via an isomorphism ϕ. Say that a vertex $v \in \iota(M)$ is a *loop root* if a loop is rooted on v. Given the definition of ι, the image $\iota(u)$ of a vertex $u \in M$ cannot be a loop root, as any loop on u in M is translated to non-loop edges in $\iota(M)$. Therefore, there is a bijection between the loop roots of $\iota(M)$ and the edges of M. As loop roots are preserved by graph isomorphism, ϕ induces a bijection between the loop roots of $\iota(M)$ and $\iota(M')$, so we have a bijection ψ between the edges of M and M'. This bijection in turn determines a directed graph isomorphism between M and M'. For instance the

source vertex of an edge can be recovered from its loop root u: follow the edge which comes after the loop, when browsing incident edges of u in clockwise order. Similarly the target vertex can be recovered. Finally, as ϕ is a map isomorphism, the cyclic order of edges around vertices is preserved, so ψ is a directed map isomorphism between M and M'. □

Corollary 4.77. *Testing whether two acyclic directed planar maps are isomorphic can be done in linear time.*

Proof. The translation via ι can be computed in linear time so the problem reduces to deciding undirected planar map isomorphism, which is linear by Theorem 4.74. □

Proposition 4.78. *Two embeddings of connected directed multigraphs in the plane are isotopic if and only if the corresponding directed maps are isomorphic.*

From string diagrams to maps

We translate any string diagram D to a directed planar map $\gamma(D)$ by replacing each vertex by the gadget below. The original edges coming from D inherit their orientation from the string diagram (top to bottom), and we add two dangling edges for each vertex. These additional dangling edges are useful for vertices with only inputs or only outputs by blocking any cyclic permutation of these edges around the vertex.[4]

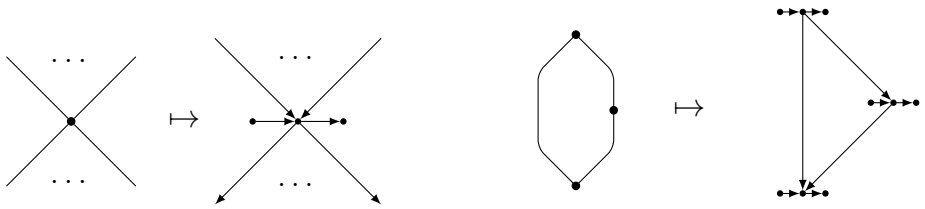


Figure 4.21: Translation of a string diagram to a directed map.

Theorem 4.79. *Any two connected diagrams are equivalent if and only if the induced directed maps are isomorphic.*

[4]These dangling edges are only useful for vertices with only inputs or only outputs but we choose to add them to all vertices for the sake of uniformity.

Proof. Exchanges on connected diagrams preserve the translation to directed maps so any two equivalent connected diagrams are mapped to isomorphic directed maps. For the converse direction, we can therefore assume that the two diagrams D, D' are in right normal form. Let ϕ be the map isomorphism between the corresponding directed maps $\gamma(D), \gamma(D')$. First, $\gamma(D)$ and $\gamma(D')$ have the same number of vertices and so do D and D'. Call n the number of vertices of D.

We prove by induction on n that $D = D'$. We reuse the induction technique introduced in Section 4.2.5: diagram D contains a leaf or a face.

If D contains a leaf l, this leaf is mapped to a vertex $\gamma(l)$ connected to three edges. As $\phi(\gamma(l))$ is also connected to three edges, there is a leaf $l' \in D'$ such that $\gamma(l') = \phi(\gamma(l))$. Because ϕ is an isomorphism of directed maps, the orientations of l and l' are the same: they are both single-input or both single-output vertices. Moreover, they are connected to their parent vertices at the same position in their list of inputs or outputs, thanks to the auxiliary edges added in the translation. Consider the diagrams E and E' obtained from D and D' by removing l and l' respectively. These diagrams are in right normal form. The isomorphism ϕ induces a map isomorphism between $\gamma(E)$ and $\gamma(E')$ so by induction $E = E'$. By Lemma 4.33, $D = D'$.

If D contains a face f, this face is mapped to a face $\gamma(f)$ in $\gamma(D)$. The face $\phi(\gamma(f))$ is itself the image of a face $f' \in D'$. Because ϕ preserves edge orientations, the mountain ranges of f and f' are equal. Let e be an eliminable edge in f and let e' be the preimage of $\phi(\gamma(e))$ in f'. By equality of the mountain ranges, e' is also eliminable in f'. By Lemma 4.39, removing e from D and e' from D' gives diagrams F and F' both in right normal form. Again we can apply the induction hypothesis to F and F', so $F = F'$, and therefore $D = D'$. □

Corollary 4.80. *The word problem for connected string diagrams can be solved in linear time.*

Proof. The translation γ from string diagrams to directed planar maps can be computed in linear time. The decision problem therefore reduces to the word

(a) A recumbent plane diagram **(b)** A locally recumbent plane diagram

Figure 4.22: Examples of topological diagrams.

problem for acyclic directed planar maps, which is solvable in linear time by Corollary 4.77. □

4.2.8 Recumbent isotopy

Joyal and Street's theorem relating diagram deformations to the axioms of monoidal categories (Theorem 2.11) requires the deformations to be recumbent. This means that at each stage of the deformation, the diagram's edges must remain upright, as shown in Figure 4.22. It was conjectured by Selinger (Selinger, 2010) that the recumbency condition can be weakened. For this weakening, the requirement that all wires must flow vertically can be dropped, but we must keep the requirement that wires stay connected to their endpoints from the same side. Figure 4.23 shows a counter-example for the conjecture without this last condition.

Figure 4.23: Arbitrary planar isomorphism does not respect morphism equality.

We now show how our reduction from string diagrams to planar maps can be used to prove Selinger's conjecture, generalizing Joyal and Street's Theorem 2.11. To extend this result to disconnected diagrams, we only need to extend the notion of directed map to disconnected cases.

Definition 4.81. *A **disconnected planar map** is defined recursively as a tree, as follows.*

- *A **face node** is a set of component nodes (possibly empty).*

- *A **component node** is a planar map m, an outer face $f_0 \in m$ and face nodes for each face $f \neq f_0$ of m.*

A disconnected planar map is given by its root face node, which has finite depth.

As this definition mirrors that of the structural tree of a diagram (Definition 4.58), it is straightforward to extend the translation of Section 4.2.7 to translate any diagram D to a disconnected planar map.

Equivalence of disconnected planar maps is defined by point-wise equivalence of the planar maps involved. By completeness of the structural tree for string diagrams (Theorem 4.61), two string diagrams are equivalent if and only if the corresponding disconnected planar maps are equivalent. For this reason, Theorem 4.73 can be extended to the disconnected case: two disconnected planar maps are equivalent if and only if their embeddings in the plane are isotopic. We therefore obtain the following theorem:

Theorem 4.82. *Two string diagrams are equivalent if and only if their translations as disconnected planar maps from Section 4.2.7 are isotopic.*

This generalizes Joyal and Street's result in the way hinted by Selinger's conjecture: the isotopy is unconstrained, although some gadgets have been added to enforce the preservation of the order of inputs and outputs around vertices.

This generalization has a clean statement in terms of planar pivotal monoidal categories (Selinger, 2010, Section 4.2). The coherence theorem for their graphical calculus is stated as follows (Selinger, 2010, Theorem 4.14).

Theorem 4.83 (Coherence for pivotal categories)**.** *A well-formed equation between morphisms in the language of pivotal categories follows from the axioms of pivotal categories if and only if it holds in the graphical language up to planar isotopy.*

We write *monoidal signature* for the generating data for a monoidal category which is free on objects and morphisms, and given a monoidal signature Σ, we write $\mathrm{M}(\Sigma)$ for the free monoidal category on Σ, and $\mathrm{P}(\Sigma)$ for the free pivotal category on Σ. Combining Theorems 4.82 and 4.83 then yields the following.

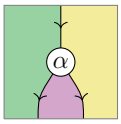
Corollary 4.84. *Given a monoidal signature Σ, the obvious embedding functor $F : \mathrm{M}(\Sigma) \to \mathrm{P}(\Sigma)$ is faithful.*

Proof. For $A, B \in \mathrm{Ob}(\mathrm{M}(\Sigma))$ and morphisms $f, g : A \to B$, Theorem 4.83 says that $F(f) = F(g)$ just when the string diagrams for f and g are isotopic. But then by Theorem 4.82, we also have $f = g$, and hence faithfulness. □

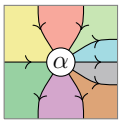
4.3 Double categories

This section is taken from the article *The word problem for double categories* (Delpeuch, 2020), published in *Theory and Applications of Categories*.

In this section we turn our attention to double categories (Ehresmann, 1963). Similarly to 2-categories and bicategories, those offer an axiomatization of the composition of planar structures, with two composition operations corresponding to the two dimensions of the plane. Informally, one can describe double categories by the shape of their string diagrams. Unlike 2-categories where the edges are required to flow along a specified direction (usually vertically), 2-cells in double categories can connect to both horizontal and vertical wires. Therefore they not only have a vertical domain and codomain, but also a horizontal domain and codomain. These definitions are made precise in Section 4.3.1.



(a) A morphism in a 2-category.



(b) A morphism in a double category.

At a first glance, double categories could be considered a more natural categorical axiomatization of planar systems, since they treat the two dimensions of the plane in a dual, interchangeable way. In comparison, the vertical and horizontal compositions in 2-categories are intrinsically different, forcing diagrams to flow in a specified direction. However, this uniform behaviour in two dimensions comes at a cost known as the *pinwheel problem*. Concretely, this problem manifests itself in the fact that not all planar arrangements of 2-cells can be composed, even if all local

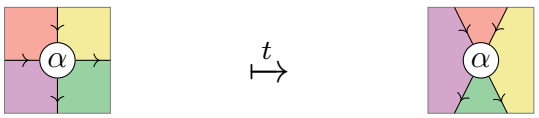
compatibility conditions are satisfied. For a diagram to be interpreted as a 2-cell it must be binary composable and this can fail if the diagram contains a so-called pinwheel, represented later in Figure 4.25.

A lot of work has already been dedicated to characterizing which arrangements of 2-cells can be composed in a double category, using order-theoretic representations of these arrangements (Dawson and Paré, 1993; Dawson, 1995). In this work, we focus instead on the word problem for 2-cells in double categories. Given two binary composable diagrams, we want to determine whether they represent the same 2-cell or not. Dawson et al. (2004) have studied this problem in the case of free extensions of double categories, showing for instance that the word problem can become undecidable with the addition of a single free 2-cell.

We study the word problem for free double categories, meaning that no equations are imposed on the generators. The only equations relating expressions in this context are the axioms of double categories. We introduce a correspondence between a free double category and a free 2-category, for which the word problem was solved in the previous chapter. We obtain as a result a quadratic time algorithm to determine if two double category diagrams are equivalent (Theorem 4.104).

Our solution to the word problem for double categories relies on a reduction to the word problem for 2-categories. Here again, the translation used is of its own interest, as it establishes a tight relation between the combinatorics of 2-categories and that of double categories. In fact, we will argue in Section 4.3.6 that free 2-categories should in a sense be preferred to free double categories, as they are simpler, equally expressive and do not suffer from the pinwheel problem.

The idea of the correspondence is very simple. In order to simulate the horizontal wires of a double category in a 2-category, we simply "rotate the string diagrams by $\frac{\pi}{4}$". In Section 4.3.3, we make this correspondence precise and show that it respects the notions of equivalences on both structures. This lets us solve the word problem for free double categories in Section 4.3.5.



This correspondence between free double categories and free 2-categories is motivated by the word problem but is of interest in its own right: it shows that one does not gain much by considering a free double category instead of the corresponding free 2-category. Reasoning in a 2-category avoids the pinwheel problem entirely as the validity of a string diagram in this structure can be checked locally. Section 4.3.6 shows how the translation could be extended to diagrams which include pinwheels, giving them a meaning in the free 2-category. This has also practical implications: one can use the translation to reason about double categories in proof assistants such as homotopy.io (Heidemann et al., 2019) which use a globular notion of n-category.

4.3.1 Double categories

Definition 4.85. *Let $\mathcal{C}$ be a category. An **internal category** in $\mathcal{C}$ consists of the following data:*

- *a pair of objects M, O, that we think of as the sets of morphisms and objects*
- *morphisms $d, c \in \mathcal{C}(M, O)$, intuitively the domain and codomain functions*
- *a morphism $\iota \in \mathcal{C}(O, M)$, taking an object to its identity map;*
- *a morphism $\mu \in \mathcal{C}(P, M)$, where P is the pullback (which is therefore required to exist) of $M \xrightarrow{d} O \xleftarrow{c} M$. This represents the multiplication of compatible pairs of morphisms.*

These morphisms are required to satisfy equalities, which correspond to the axioms of a category (associativity and unitality of composition, as well as equations for the domains and codomains of identities and composites).

The definition above is chosen such that an internal category in **Set** is a small category. The purpose of this concept is that its generality makes it possible to interpret it in other categories.

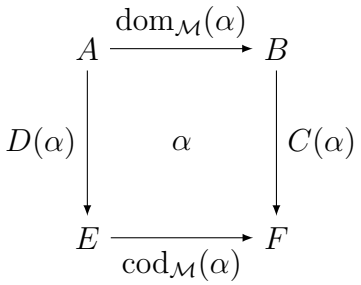
Definition 4.86. *A **double category** is an internal category in* **Cat**, *the category of small categories.*

This definition is concise and this conciseness justifies the interest in this structure, which was originally introduced by Ehresmann (1963). However, it is of little help to build intuition about the nature of such an object, so let us unfold its content. A double category consists of an object category $\mathcal{O}$ and a morphisms category $\mathcal{M}$, with functors $D, C : \mathcal{M} \to \mathcal{O}$, $I : \mathcal{O} \to \mathcal{M}$ and $M : \mathcal{P} \to \mathcal{M}$ where $\mathcal{P}$ is defined as above. We will call

- objects of $\mathcal{O}$ as **objects** of the double category;
- morphisms of $\mathcal{O}$ as **vertical morphisms** of the double category;
- objects of $\mathcal{M}$ as **horizontal morphisms** of the double category;
- morphisms of $\mathcal{M}$ as **2-cells** of the double category.

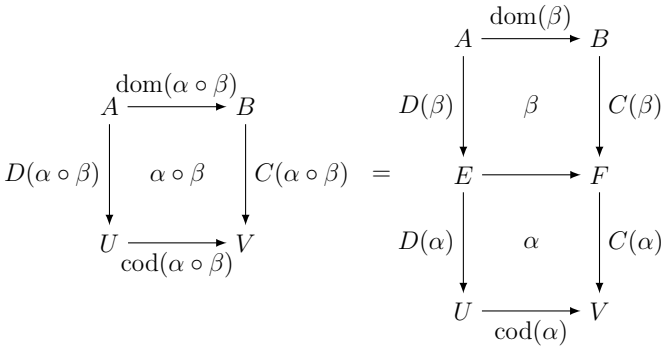
Initially, it can seem confusing that objects of $\mathcal{M}$ are thought of as morphisms. The reason for this is that by forgetting morphisms, i.e. taking the image of our internal category via the forgetful functor **Cat** $\to$ **Set**, we obtain an internal category in **Set**, i.e. a small category. We will call this the **horizontal category** of the double category. As its morphisms are the objects of $\mathcal{M}$, this justifies their name. These horizontal morphisms have as domains and codomains objects of $\mathcal{O}$. These objects are also involved in another category, namely $\mathcal{O}$ itself, that we will call the **vertical category** of the double category.

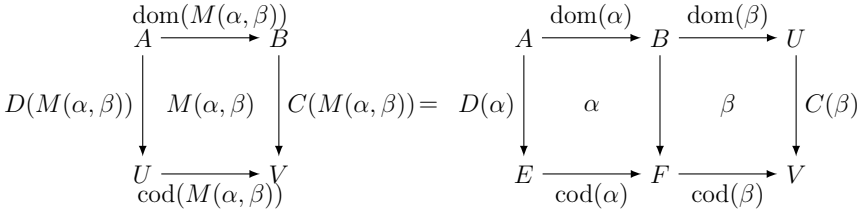
Any 2-cell α has two horizontal morphisms as domain and codomain, $\mathrm{dom}_{\mathcal{M}}(\alpha)$ and $\mathrm{cod}_{\mathcal{M}}(\alpha)$ as a morphism of $\mathcal{M}$. We will call these the **horizontal domain** and **codomain** of α. Furthermore, it is associated by the internal category structure to $D(\alpha)$ and $C(\alpha)$, which are vertical morphisms. We will therefore call these the **vertical domain** and **codomain** of α. Finally, the functoriality of D and C ensures that for instance $D(\mathrm{dom}_{\mathcal{M}}(\alpha)) = \mathrm{dom}_{\mathcal{O}}(D(\alpha))$ and similarly for C and cod. This suggests the representation of α as a square:



Although this diagram is similar to commutative diagrams used in category theory, we stress that it is here used in a more general sense, as composing horizontal and vertical morphisms does not make sense in general.

The 2-cells in a double category can be composed in two different ways. First, as morphisms of $\mathcal{M}$, two 2-cells can be composed if they have compatible horizontal domain and codomain. We call this the **vertical composition**. Second, the functor M defines a composition for 2-cells with compatible vertical domains and codomain, and we call this the **horizontal composition**. These compositions can be represented with diagrams:





The functoriality of M ensures that these two compositions are compatible: $(\alpha \star \delta) \circ (\beta \star \gamma) = (\alpha \circ \beta) \star (\delta \circ \gamma)$ for all 2-cells such that both sides of the equation

are defined. This means that the following diagram is unambiguous:

$$\begin{array}{ccccc} A & \longrightarrow & B & \longrightarrow & C \\ \downarrow & \beta & \downarrow & \gamma & \downarrow \\ D & \longrightarrow & E & \longrightarrow & F \\ \downarrow & \alpha & \downarrow & \delta & \downarrow \\ G & \longrightarrow & H & \longrightarrow & I \end{array}$$

Given that the horizontal and vertical compositions are also associative, it is natural to represent composite 2-cells as tilings of rectangles in the plane, with the appropriate conditions on edges to ensure compatibility between the composed 2-cells. Dawson and Pare (1993) have shown that if there are two ways to interpret such a tiling as a tree of horizontal and vertical compositions, then the resulting 2-cells will be equal. However, there exist tilings which satisfy the local compatibility conditions but do not arise from the horizontal and vertical compositions. The minimal example of this is known as the *pinwheel* and is shown in Figure 4.25.

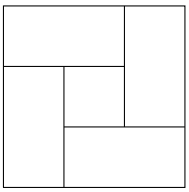


Figure 4.25: A pinwheel diagram, which cannot be expressed as a binary composite.

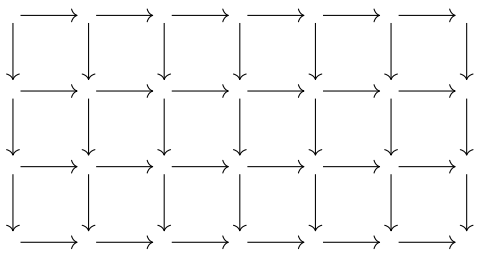
It was then shown by Dawson (1995) that this is essentially the only obstacle to composition of diagrams in double categories.

4.3.2 Free double categories

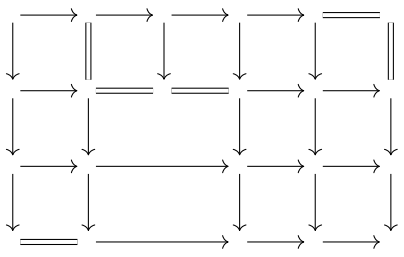
Double categories are rich objects and defining them therefore requires some care. Given horizontal and vertical categories with the same objects, and a set of generating tiles whose boundaries are chosen from the horizontal and vertical categories, we want to generate the free double category on these tiles.

One simple approach to generate such a double category would be to use its definition as internal category object in **Cat**, and simply internalize the definition of a free category on a graph. A graph object in **Cat** is called a **double graph** and

is essentially a double category without identities and compositions. Interpreted in **Cat**, the construction which defines a free category object from a graph object does give a double category, but as pointed out by Dawson and Paré (2002) this imposes important restrictions on the boundaries of the generating tiles: they must be generating morphisms of the resulting vertical and horizontal categories. Therefore, all generated composites have a grid-like shape:



Dawson and Paré (2002) propose a more general construction which allows identities as cell boundaries. To do so they use the notion of reflexive graph: it is a directed graph with designated loops on each vertex. One can define the free category generated by a reflexive graph, where the loops are interpreted as identities. Internalized in **Cat**, this gives rise to the notion of **double reflexive graph** which generates a double category. This makes it possible to use generators which have identities as boundaries:



As this is still not as general as it could be, Fiore et al. (2008) introduces the notion of *double derivation scheme*. A double derivation scheme is a double graph whose horizontal and vertical objects form categories. Therefore, generating a double category from a double derivation scheme makes it possible to use arbitrary boundaries for the generating cells. The main difference with the previous approaches is that the notion of double derivation scheme does not arise by internalizing in

Cat a notion formulated in the internal language of categories. Moreover a double derivation scheme can also introduce algebraic equations between expressions, quotienting the generated structure accordingly. In our case, no such equations are used, so we give a simpler description of the construction.

Definition 4.87. *A* ***double signature*** $S = (A, H, V, C)$ *is given by:*

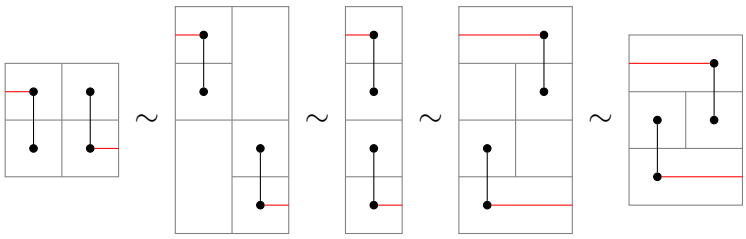
- *a set of objects* A*;*
- *a set of generating horizontal morphisms* H*;*
- *a set of generating vertical morphisms* V*;*
- *a set of generating 2-cells* C*.*

Furthermore each $h \in H$ *is associated with* $\operatorname{dom} h, \operatorname{cod} h \in A$ *and similarly for* V*. This defines free categories* H^* *and* V^**. Each* $\alpha \in C$ *is associated with compatible vertical and horizontal domains and codomains* $\operatorname{domh} \alpha, \operatorname{codh} \alpha \in H^*$ *and* $\operatorname{domv} \alpha, \operatorname{codv} \alpha \in V^*$*. The required compatibility is* $\operatorname{dom} \operatorname{domh} \alpha = \operatorname{dom} \operatorname{domv} \alpha$ *and three other similar equations.*

The set of 2-cells of the double category generated by this data is generated inductively from the generators in C, vertical and horizontal identities. From these generators we take the closure by vertical and horizontal composition of compatible cells: this gives us the set of 2-cell expressions on the signature. To obtain the set of 2-cells, we quotient by unitality and associativity of the vertical and horizontal compositions and by the exchange law. Furthermore, horizontal and vertical identities on identity morphisms (depicted as empty 2-cells) are equated. These are precisely the laws of double categories, hence this defines the free double category S_d on the given data.

Expressions in double categories can be drawn as string diagrams (Myers, 2016), and in the sequel we will use the terms “expression” and “diagram” interchangeably.

Here is an example of a series of equivalences between expressions of 2-cells, drawn as string diagrams:



We draw horizontal wires in red, this will help us to to distinguish them from vertical wires in the next section. We also omit region colors as they are irrelevant for equivalences and do not play any role in the word problem.

Our goal in this work is to propose an alternate representation for 2-cells, making it possible to decide whether two expressions of 2-cells are equivalent under these axioms.

4.3.3 Translation to 2-categories

Double categories can be seen as a generalization of 2-categories, as a 2-category is a double category where all vertical morphisms are identities. Given the inherent duality in double categories, a 2-category can also be seen as a double category with identity horizontal morphisms.

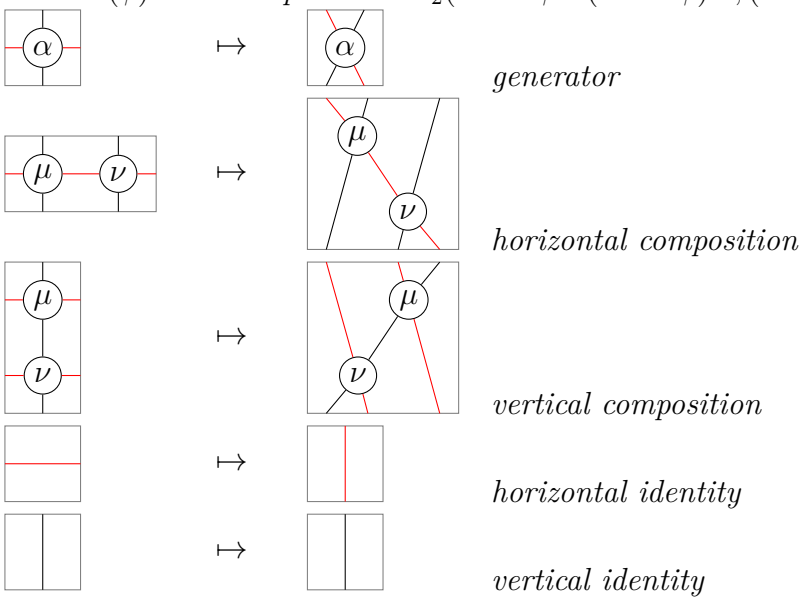
In this section, we show how a free double category can conversely give rise to a free 2-category. Our goal is to reuse the algorithms for the word problem in 2-categories developed in the previous chapter.

Definition 4.88. *Given a double signature $S = (A, H, V, C)$, we define the 2-category S_2 as the free 2-category generated by:*

- *objects $a \in A$;*
- *1-morphisms $h : \operatorname{dom} h \to \operatorname{cod} h$ for $h \in H$ and $v^{op} : \operatorname{cod} v \to \operatorname{dom} v$ for $v \in H$;*
- *2-morphisms $\alpha : \operatorname{domh} \alpha \circ (\operatorname{domv} \alpha)^{op} \to (\operatorname{codv} \alpha)^{op} \circ \operatorname{codh} \alpha$*

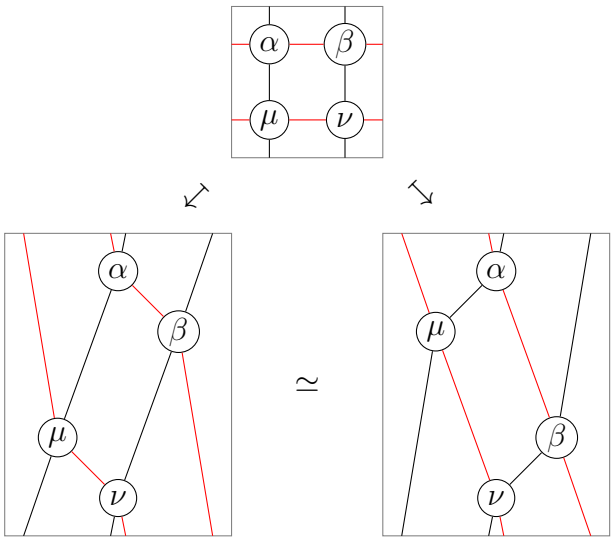
Note that the vertical generators are reversed in the 2-category, making it possible to compose the horizontal and vertical domains together, and similarly for the codomain.

Definition 4.89. *Let ϕ be a 2-cell expression in S_d. We inductively define its translation $t(\phi)$ as a morphism in $S_2(\mathrm{domh}\,\phi \circ (\mathrm{domv}\,\phi)^{op}, (\mathrm{codv}\,\phi)^{op} \circ \mathrm{codh}\,\phi)$:*



Lemma 4.90. *The translation t respects the axioms of double categories, i.e. it extends to a map from 2-cells in S_d to 2-cells in S_2.*

Proof. One can check that unitality and associativity are respected. The exchange law in double categories translates to the exchange law in 2-categories:



□

Note that our translation is a simple function, as we could not find a way to frame it as a suitable sort of functor. We argue that this is constrained on us by the problem at hand, which is precisely about relating different algebraic structures.

Our goal is to show the converse: if the translations of two expressions in S_d are equivalent as morphisms in S_2, then so are their antecedents in S_d. To do so, we need to construct a reverse translation, from diagrams in the free 2-category to diagrams in the free double category.

4.3.4 Partial tilings

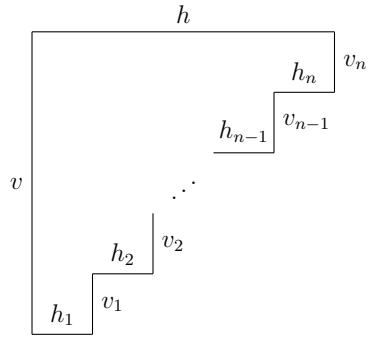
To provide an inverse to the translation t, let us first introduce a necessary condition on a diagram in S_2 to be in the image of t.

Definition 4.91. *A diagram $\phi \in S_2$ is **admissible** if its domain is of the form $v^{op}; h$ and its codomain is of the form $h'; v'^{op}$.*

For all $\psi \in S_d$, $t(\psi)$ is admissible. Conversely, for all admissible $\phi \in S_2$, we want to construct a corresponding tiling. To do so, we introduce the notion of partial tiling as an incomplete diagram in the free double category.

Definition 4.92. *Let $n \geq 1$, and $h, h_1, \ldots, h_n \in H^*$ and $v, v_1, \ldots, v_n \in V^*$. Assume that h_i is not an identity for $i > 1$ and v_i is not an identity for $i < n$.*

*A **partial tiling** of type $h, v \to h_1, v_1, \ldots, h_n, v_n$ is a subdivision of the following shape into rectangles:*



Each of the rectangles in the subdivision is attributed a generator $\alpha \in C$ or a vertical or horizontal identity, such that the horizontal and vertical domains and codomains match on each edge.

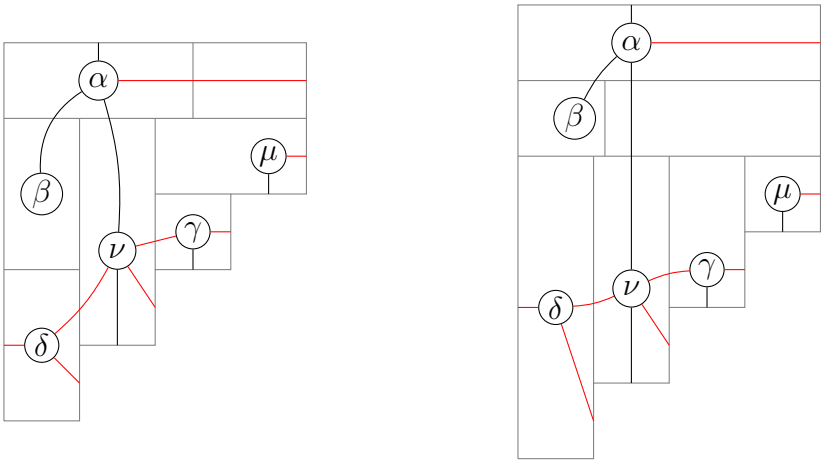
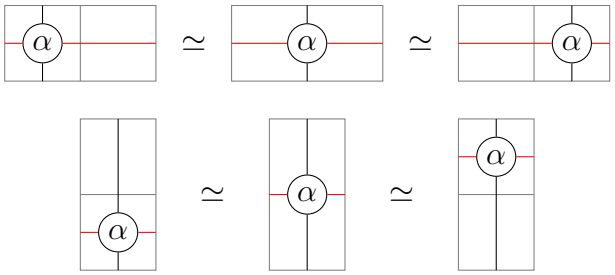


Figure 4.26: Examples of partial tilings.

The definition above is purposefully left at a rather intuitive level as a precise topological definition would rather obscure the purpose of the concept. We think of a partial tiling as some upper-left corner of a 2-cell in a double category. We will therefore draw partial tilings just like string diagrams for double categories, as in Figure 4.26.

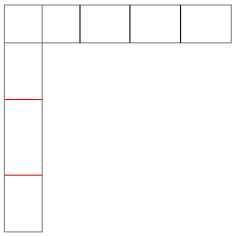
Definition 4.93. *Two partial tilings are* ***equivalent*** *when they can be related by a series of applications of these rules (where α can be an identity itself):*



as well as continuous translations of horizontal or vertical boundaries in the subdivision. We denote this equivalence by $\simeq$.

For instance, the two partial tilings in Figure 4.26 are equivalent.

In the special case where $n = 2$, h_1 and v_2 are identities and $h = h_2$, $v = v_2$, and assuming $\operatorname{dom} h = \operatorname{dom} v$, there is an **empty partial tiling** of type $h, v \to 1_{\operatorname{cod} v}, v, h, 1_{\operatorname{cod} h}$:



Another special case are partial tilings of type $h, v \to h_1, v_1$ which have a rectangular shape. In this case, we can interpret them as 2-cells, but only if they are binary composable.

Lemma 4.94. *A partial tiling of type $h, v \to h_1, v_1$ is **binary composable** if it can be obtained by repeated application of the horizontal and vertical composition from generators. In this case, it represents a 2-cell in S_d, and its meaning is invariant under equivalence of partial tilings.*

Proof. If a diagram is binary composable then by the general associativity result of Dawson and Pare (1993), it can be interpreted as a 2-cell in S_d which does not depend on the order of composition chosen. Then, equivalences of partial tilings correspond to unitality of identities when interpreted in a binary composable diagram, so the 2-cell is invariant under these equivalences. □

Definition 4.95. *Let h be an horizontal morphism in S_d. It can be uniquely decomposed as a composition of generators $h = h_1 \circ \cdots \circ h_k$. We define the **length** of h as $|h| = k$. Let $0 \leq i < k$ and $1 \leq j < k$. We say that $h' = h_{i+1} \circ \cdots \circ h_j$ is a **factor at index** i of h. Similar notions are defined for vertical morphisms.*

For instance, the factors at index 0 of a morphism h are its prefixes.

Definition 4.96. *Let m be a partial tiling of type $h, v \to h_1, v_1, \ldots, h_n, v_n$ and let $\alpha : h', v' \to h'', v''$ be a generator. A **gluing position** of α on m is one of the following:*

- *if h' is a prefix of h_1, then $(0, 0, 0)$ is a gluing position;*

- *if h' is a factor of h_1 at index $i > 0$ and v' is an identity, then $(0, i, 0)$ is a gluing position;*
- *if v' is a prefix of v_n, then $(n, 0, 0)$ is a gluing position;*
- *if v' is a factor of v_n at index $i > 0$ and h' is an identity, then $(n, 0, i)$ is a gluing position;*

Furthermore, for all $1 \leq k < n$:

- *if v' is a prefix of v_k and h' is a prefix of h_{k+1}, then $(k, 0, 0)$ is a gluing position;*
- *if v' is a factor of v_k at index i and h' is an identity, then $(k, 0, i)$ is a gluing position;*
- *if h' is a factor of h_{k+1} at index i and v' is an identity, then $(k, i, 0)$ is a gluing position.*

*For each gluing position p we define the **gluing** of α to m, denoted by $m \star_p \alpha$, by the partial tiling obtained by adjoining α at the designated position, and adding any necessary identity to satisfy the condition of Definition 4.92.*

Figure 4.27 shows all the possible gluing positions. Figure 4.28 shows how a generator can be glued at multiple positions on a partial tiling and how identities can be used to ensure that the resulting arrangement has non-identity inner boundaries.

Definition 4.97. *Let ϕ be a diagram in the free 2-category S_2. We assume that the domain of ϕ is of the form $v^{op}; h$ with v and h paths of generators from S.*

Let $l \in \mathbb{N}$ be a level in ϕ and $h_1; v_1^{op}; \ldots; h_n; v_n^{op}$ be the type of the diagram at this height, where v_1 and h_n can possibly be identities unlike the others elements of the sequence.

We associate to this data a partial tiling $p_k(\phi) : h, v \to h_1, v_1, \ldots, h_n, v_n$, by induction on k. If $k = 0$, $p_k(\phi)$ is the empty partial tiling of type $v, h \to 1, v, h, 1$. Otherwise, let α be the generator between levels $k-1$ and k. We define $p_k(\phi)$ as

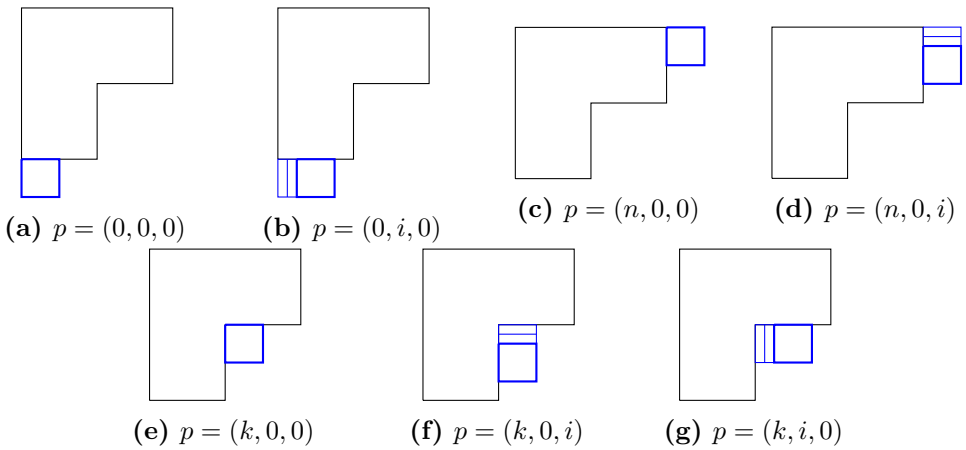


Figure 4.27: Possible gluing positions. When the second or third component of the position is not null, an identity cell is added.

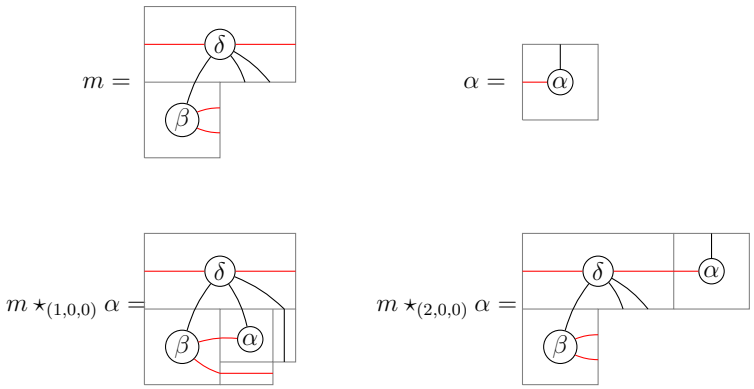


Figure 4.28: Example of gluings of a generator on a partial tiling.

the gluing of α on $p_{k-1}(\phi)$ at the position indicated by the connection of α to the level $k-1$ of ϕ.

Finally we define $p(\phi)$ as $p_f(\phi)$ for f the final level of ϕ.

The construction relies on the following two lemmas:

Lemma 4.98. *Let α be the generator between slices k and $k+1$ in ϕ, a diagram in S_2. If α has at least one input wire, this determines a unique gluing position g of α on $p_k(\phi)$.*

Proof. Each wire crossing level k in ϕ corresponds to an open wire on the boundary of $p_k(\phi)$, either in a vertical or horizontal boundary depending on the colour of the wire.

Let $v^{\mathrm{op}}; h$ be the domain of α.

By assumption, at least one of v, h is not an identity. Assume first that v is not an identity. As no horizontal inner boundaries of $p_k(\phi)$ are identities, as required by Definition 4.92, any contiguous sequence of red wires in ϕ corresponds to a contiguous sequence of wires on some vertical boundary v_i of $p_k(\phi)$.

Let j be such that v is a factor at index j in v_i. One can then check that $(i, 0, j)$ is a valid gluing position for α on $p_k(\phi)$.

Similarly, if h is not an identity, then the corresponding wires in ϕ determine a unique occurrence of h in a vertical boundary h_i of $p_k(\phi)$, and by denoting by j the index of h in h_i, this determines the gluing position $(i-1, j, 0)$. □

Lemma 4.99. *Let again α be the generator between slices h and $h+1$ in ϕ, a diagram in S_2. If α has no input wire, meaning that its domain is the identity, then this determines either one or two gluing positions of α on $p_h(\phi)$. If there are two such positions l, l' then $p_h(\phi) \star_l \alpha \simeq p_h(\phi) \star_{l'} \alpha$.*

Proof. Let $h_1; v_1^{\mathrm{op}}; \ldots; h_n; v_n^{\mathrm{op}}$ be the type of the diagram at height h. Again, each wire in this sequence corresponds to an open wire on the boundary of $p_k(\phi)$. The wires passing to the left of α in ϕ determine a position in this sequence where the generator α is inserted. The gluing positions depend on this position.

If α is bordered by two horizontal wires on each side (respectively two vertical wires), this determines a unique gluing position $(k, i, 0)$ (respectively $(k, 0, i)$) as in the previous lemma. Similarly, if α neighbours a vertical wire on its left and a horizontal wire on its right, this determines a unique gluing position $(k, 0, 0)$ as in the previous lemma.

The remaining cases are when α neighbours a horizontal wire on its left and a vertical wire on its right, when α does not have any wire on its left and a vertical one on its right, when it has a horizontal wire on its left and none on its right, or when there are no wires neither on the left or the right of α. In this case this determines two gluing positions $l = (k, i, 0)$ and $l' = (k+1, 0, j)$, and Figure 4.29 shows how $p_h(\phi) \star_l \alpha \simeq p_h(\phi) \star_{l'} \alpha$ in this case. □

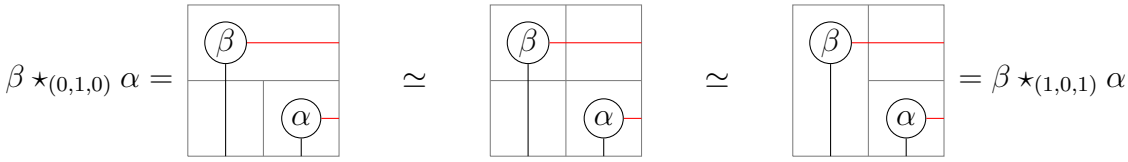


Figure 4.29: Two equivalent gluing positions in Lemma 4.99.

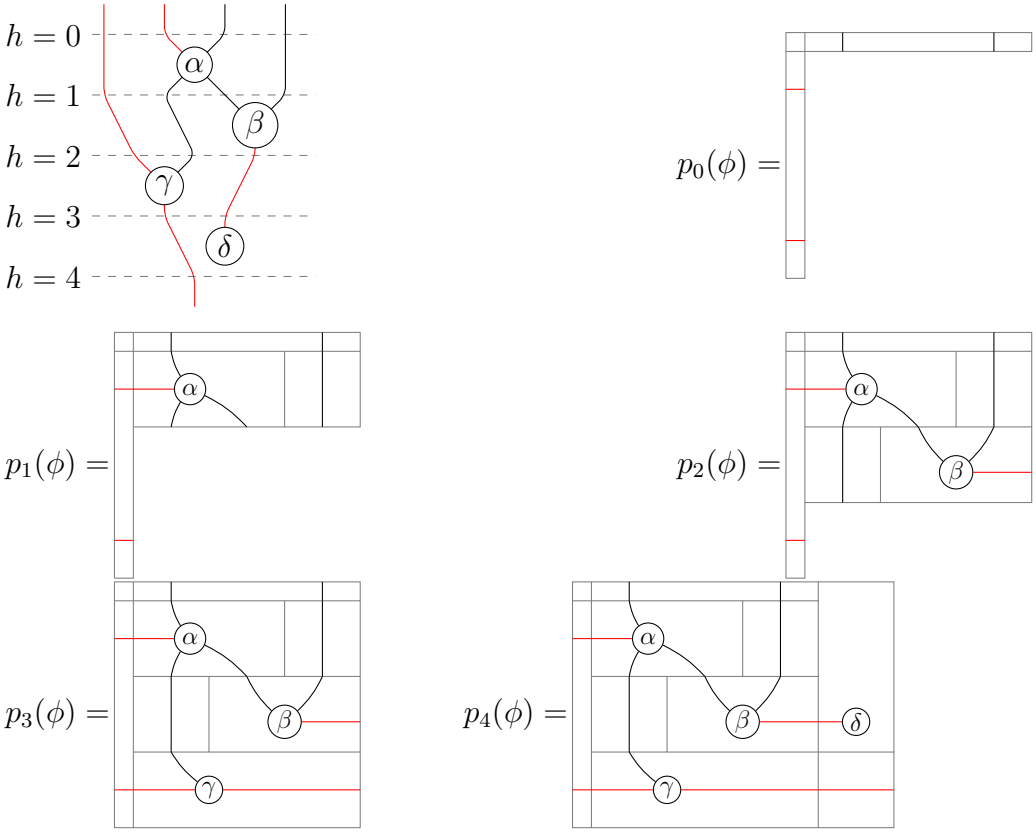
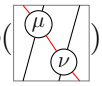
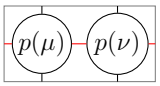
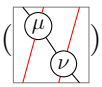
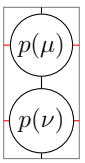
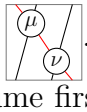
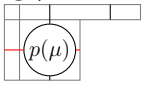


Figure 4.30: Inductive construction of $p(\phi)$.

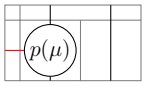
Lemma 4.100. *For all 2-cell diagrams $\mu, \nu \in S_2$ such that*  *is defined,* $p($  $) \simeq$  $.$

Similarly, if  *is defined, then* $p($  $) \simeq$  $.$

Proof. By duality let us prove the result for the first case, horizontal composition. Let $\phi =$  . Let h be the level between μ and ν in ϕ.

Assume first that the red edge connecting μ and ν is not empty (it is not an identity vertical morphism). Then $p_h(\phi) =$  and $p(\phi)$ is obtained from

$p_h(\phi)$ by gluing on it the generators in ν. Since the vertical codomain of μ passes to the left of ν, these generators are glued on positions (k, i, j) with $k > 0$. Performing these gluings on an empty diagram gives $p(\nu)$, so $p(\phi)$ is equivalent to the required double diagram.

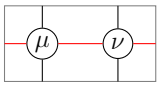
If there is no red edge connecting μ to ν then $p_h(\phi) =$ 

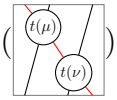
and $p(\phi)$ is obtained from $p_h(\phi)$ by gluing the generators in ν on the second part of its vertical codomain, so it can again be rewritten into the required form by unitality. □

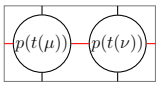
Lemma 4.101. *For any 2-cell diagram $\phi \in S_d$, $p(t(\phi)) \simeq \phi$.*

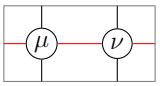
Proof. By induction on ϕ. If ϕ is a generator or the identity, the result holds.

If ϕ is a horizontal or vertical composition, then we use Lemma 4.100 and the induction hypothesis of the composed diagrams:

$p(t($ 

$)) = p($ 

$) \simeq$ 

$\simeq$ 

□

Lemma 4.102. *Let ϕ, ϕ' be admissible diagrams in S_2 with $\phi \sim \phi'$. Then $p(\phi) \simeq p(\phi')$.*

Proof. By induction we can assume that ϕ and ϕ' are related by a single exchange, swapping the generators between levels $h-1$, h and $h+1$. Let α be the generator between levels $h-1$ and h and β the one between h and $h+1$. It suffices to check that $p_{h-1}(\phi) \star_l \alpha \star_{l'} \beta \simeq p_{h-1}(\phi) \star_m \beta \star_{m'} \alpha$ where l, l' are the gluing positions for the generators in ϕ and m, m' are their counterparts in ϕ'. By a tedious case analysis one can check that because the generators at these slices can be exchanged, this ensures that the induced gluing positions are disjoint, such that the equivalence above either holds trivially (the partial tilings being syntactically equal) or via equivalences analogous to those of Figure 4.29. □

Theorem 4.103. *The translation t respects the axioms of double categories, and conversely, if any two diagrams in its image are equivalent, then so are their antecedents.*

Proof. The forward direction is established by Lemma 4.90 and the backward one by Lemma 4.102. □

4.3.5 Word problem

We can use the translation defined in the previous section to solve the word problem for double categories:

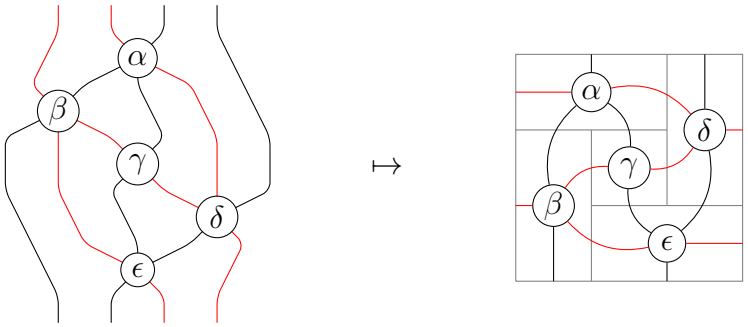
Theorem 4.104. *Let S be a double signature. The word problem for 2-cells in the free double category S_d can be solved in $O(ve)$, where v is the number of generators in the expressions and e the number of connecting edges between them.*

Proof. Given two diagrams ϕ, ϕ' in S_d, we can compute their translation $t(\phi), t(\phi')$ in linear time. Then, we can check if these diagrams are equivalent as 2-cells in S_2 using the algorithm of Delpeuch and Vicary (2018), in $O(ve)$. As p is faithful (Lemma 4.102), this determines if ϕ and ϕ' are equivalent in S_d. □

4.3.6 Conclusion

We have solved the word problem for free double categories by reducing it to the word problem for free 2-categories.

Interestingly, our translation p from the free 2-category to the free double category works for all admissible diagrams, and admissibility is a simple condition on the domain and codomain. We are not requiring any global condition such as binary composability on the 2-cell. As a consequence, this translation p can produce tilings which are not binary composable.



It is therefore tempting to extend the forward translation t to double category diagrams which are not binary composable. By the characterization of Dawson (1995) of non-composable diagrams, it is sufficient to translate the two pinwheels: by induction, all diagrams could then be interpreted. However, there could potentially be multiple ways to decompose a diagram as a tree of binary and pinwheel composites, so to define t properly we would need an equivalent of the general associativity result of Dawson and Pare (1993) with pinwheel composition. That would only be possible given an appropriate notion of equivalence, which would amount to developing a notion of "double category with pinwheels". This does not strike us as a particularly useful notion as it would be rather complicated, with four different composition operators and many axioms to relate their applications, only to represent planar systems.

What this really means is that free 2-categories already provide the appropriate notion of "free double category with pinwheel composites", in the sense that they capture the desired combinatorics with a much simpler axiomatization.

This fact has been observed at an intuitive level by Reutter and Vicary (2019) who modeled biunitary connections in a 2-category rather than a double category, by using the same rotation. They noticed that biunitaries forming a pinwheel pattern could be composed into a new biunitary. As double categories are not equipped with such a composition, a 2-categorical model provides a more useful representation. Modelling biunitaries in double categories would artificially forbid pinwheel composites which are actually allowed physically. We suspect that other uses of double categories, for instance in computer science (Bruni et al., 2002), could be recast in 2-categories without loss of expressivity, as they do not rely on the exclusion of pinwheel composites.

4.4 Braided monoidal categories

This section is taken from the article *The word problem for braided monoidal categories is unknot-hard* (Delpeuch and Vicary, 2021), presented in the *Applied Category Theory 2021* conference.

In this section, we turn our attention to the word problem for braided monoidal categories. Unlike the previous sections in this chapter, we do not propose an algorithm deciding equality, but instead show that this word problem seems to be a difficult one. More precisely we show that it is at least as hard as the unknotting problem.

The unknotting problem consists in determining whether a knot can be untied and was first formulated by Dehn (1910). The decidability of this problem remained open until Haken (1961) gave the first algorithm for it. As of today, no polynomial time algorithm is known for it.

One reason why we are interested in braided monoidal categories is that they are a particularly natural sort of categories, in the sense that they arise as doubly degenerate Gray categories (Gurski and Cheng, 2011), as explained in Section 2.3.5. Studying the word problem for them is therefore a first step towards understanding the word problem for weak higher categories, for which little is known to date.

4.4.1 Background

As explained in Section 2.3.3, braided monoidal categories generalize the braid groups, in the sense that the category of braids can be obtained as the free braided monoidal category on a single object and no generating morphism. The word problem for the braid group itself is well understood:

Theorem 4.105. *The word problem for the braid group B_n can be solved in polynomial time: given two strings of generators and generator inverses, one can determine if they represent the same braid in quadratic time in the length of the strings l (for a fixed n).*

See Dehornoy (2007) for a review of the various techniques which can be used to achieve this complexity. It seems hard to generalize any of these to the case of braided monoidal categories.

Intuitively, the word problem for braided monoidal categories is harder than the one for the braid group because of the existence of other morphisms which

can block the interaction between braids. Because of these additional morphisms, string diagrams in braided monoidal categories can look *knotted* and the equivalence problem for them intuitively becomes harder. In this section we make this intuition more precise, by showing that the equivalence problem for braided monoidal categories is at least as hard as the unknotting problem.

Braided pivotal categories

Definition 4.106. *In a monoidal category $\mathcal{C}$, an object $A \in \mathcal{C}$ has a **left adjoint** $B \in \mathcal{C}$ (or equivalently, A is the **right adjoint** of B) when there are morphisms*  *and*  *such that the **yanking equations** (or **zigzag equations**) are satisfied:*

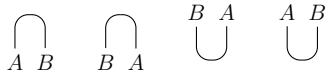
Definition 4.107. *A monoidal category $\mathcal{C}$ is **left autonomous** if every object $A \in \mathcal{C}$ has a left adjoint ${}^*A$. A monoidal category $\mathcal{C}$ is **right autonomous** if every object $A \in \mathcal{C}$ has a right adjoint A^*. A category that is both left and right autonomous is simply called **autonomous**.*

Lemma 4.108. *Any braided monoidal category that is left autonomous is also right autonomous (and therefore autonomous).*

Proof. See Section 4.4.5 (Braided Autonomous Categories) of Selinger (2010). □

Definition 4.109. *A **strict pivotal category** is a monoidal category where every object A has identical left and right adjoints.*

In a braided pivotal category, for each object A there is an object B with the following morphisms:



such that all four yanking equations are satisfied.

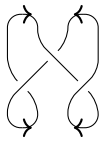
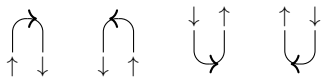


Figure 4.31: An oriented knot as a morphism in $\mathbb{ROT}\mathtt{ang}$.

Definition 4.110 (Freyd and Yetter, 1989)**.** $\mathbb{ROT}\mathtt{ang}$ *is the free braided pivotal category generated by an object represented by the symbol "↑". We denote by "↓" its adjoint.*

As the notations suggest, the wires of string diagrams in $\mathbb{ROT}\mathtt{ang}$ can be associated with an upwards or downwards orientation. We adopt the following representation for the morphisms arising from the pivotal structure:



With this convention, we can represent any oriented knot, i.e. any embedding of an oriented loop in $\mathbb{R}^3$, as a morphism in $\mathbb{ROT}\mathtt{ang}$, as in Figure 4.31. In fact, this representation is more than a convention, as the following theorem shows:

Definition 4.111 (Definition 2.4 in Freyd and Yetter (1989))**.** *A* ***tangle*** *is a rectangular portion of a knot diagram. We say that two tangles are equal if there is a regular isotopy carrying one to the other in such a way that corresponding edges of the diagram are preserved set-wise.*

Theorem 4.112 (Theorem 3.5 in Freyd and Yetter (1989))**.** $\mathbb{ROT}\mathtt{ang}$ *is the category of oriented tangles up to regular isotopy.*

This means that two morphisms $f, g \in \mathbb{ROT}\mathtt{ang}$ are equal if and only if their string diagrams, considered as oriented tangles in three-dimensional space as defined above, are regularly isotopic. Regular isotopy is a more restrictive sort of isotopy than the notion generally used in knot theory, as the following morphisms are distinct in $\mathbb{ROT}\mathtt{ang}$:

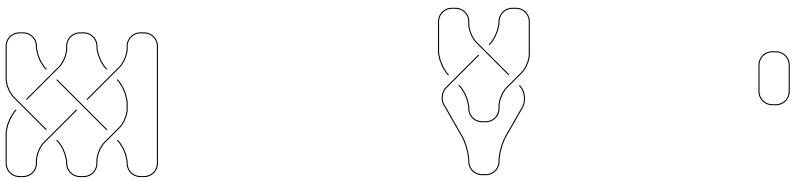


Figure 4.32: Some knot diagrams.

The move that equates them is called the Reidemeister type I move, which is therefore not admissible for the string diagrams of $\mathbb{ROT}\mathtt{ang}$.

Definition 4.113. *A morphism $f \in \mathbb{ROT}\mathtt{ang}$ is an **(oriented) knot** if its string diagram has a single connected component.*

The unknotting problem

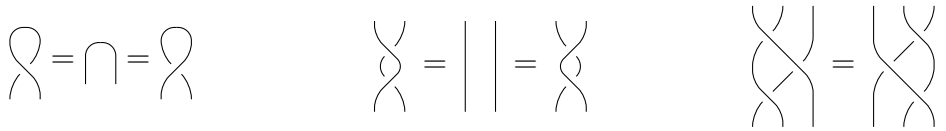
In this section we give a brief overview of the unknotting problem and some complexity results about it.

Definition 4.114. *A **topological knot** is the embedding of a loop in $\mathbb{R}^3$. Two knots K_1, K_2 are in knot isotopy if there is an orientation-preserving homeomorphism h of $\mathbb{R}^3$ such that $h(K_1) = h(K_2)$. A **knot diagram** is the projection of a knot on a plane, such that no two crossings happen at the same place. Additionally, the diagram records the relative position of the strands at each crossing.*

All knots considered here will be required to be tame, i.e. knot isotopic to a polygonal knot. This gets rid of some pathological cases.

Some example knot diagrams are given in Figure 4.32. The Reidemeister moves are local transformations of knot diagrams which are divided in three categories, as shown in Figure 4.33. Note that in addition to these moves, all planar isotopies are implicitly allowed, without restricting the direction of strands in any way (unlike the recumbent isotopies of string diagrams).

Theorem 4.115 (Reidemeister)**.** *Two knot diagrams represent knot-isotopic knots if and only if they are related by a sequence of Reidemeister moves.*



(a) Reidemeister type I **(b)** Reidemeister type II **(c)** Reidemeister type III

Figure 4.33: Reidemeister moves.

Knot diagrams can be encoded in various ways, for instance as four-valent planar maps where vertices are crossings and edges are parts of strands. This makes it possible to formulate decision problems about knots and study their complexity.

Definition 4.116. *The unknotting problem* UNKNOT *is the decision problem to determine if a knot is knot-isotopic to the unknot. In other words, it consists in determining whether there exists a series of Reidemeister moves which eliminates all crossings in a given knot diagram.*

This problem was first formulated by Dehn (1910) and its decidability remained open until Haken (1961) found an algorithm for it. The problem has since attracted a lot of attention, and we give a summary of the latest results about it.

Theorem 4.117 (Lackenby, 2015)**.** *There exists a polynomial $P(c)$ such that for all knot diagram K of the unknot with c crossings, there is a sequence of Reidemeister moves unknotting it, whose length is bounded by $P(c)$.*

Corollary 4.118. *UNKNOT lies in NP.*

Theorem 4.119 (Lackenby, 2019)**.** *UNKNOT lies in co-NP.*

Recently, Lackenby announced a quasi-polynomial time solution to UNKNOT, but the corresponding article has not been made public to date. No polynomial time algorithm for this problem is known so far.

4.4.2 Reducing the unknotting problem to the braided pivotal word problem

Despite the discrepancy between knot isotopy, used in the unknotting problem, and the regular isotopy used in $\mathbb{ROTang}$, we will show that the unknotting problem

can be reduced to the word problem in $\mathbb{ROT}\mathtt{ang}$. This will show that the word problem for $\mathbb{ROT}\mathtt{ang}$ is at least as hard as the unknotting problem. This section is dedicated to this result.

Writhe and turning number

The main differences between the unknotting problem and the word problem for $\mathbb{ROT}\mathtt{ang}$ is that in the latter, knots are oriented and the Reidemeister type I move is not allowed. Because of this, we will see in this section that we can associate a quantity called *writhe* to diagrams in $\mathbb{ROT}\mathtt{ang}$, which is preserved by all the axioms of this category.

Definition 4.120. *The **writhe** (or **framing number**) $W(f)$ of a diagram $f \in \mathbb{ROT}\mathtt{ang}$ is the sum of the valuations $W(b)$ for each braiding b which appears in f:*

$W($  $) = +1$ $\qquad$ $W($  $) = -1$

Definition 4.121. *The **turning number** (or **winding number**) $T(f)$ of a morphism $f \in \mathbb{ROT}\mathtt{ang}$ is the sum of the local turning numbers which appear in f:*

$T($  $) = +1$ $\quad$ $T($  $) = -1$ $\quad$ $T($  $) = -1$ $\quad$ $T($  $) = +1$

The turning number is well-defined because the axioms of $\mathbb{ROT}\mathtt{ang}$ respect the turning number, making it independent of the particular diagram considered.

Theorem 4.122 (Trace, 1983)**.** *Let $f, g \in \mathbb{ROT}\mathtt{ang}$ be two knots. Then there is a regular isotopy between f and g if and only if f and g are knot-isotopic, $W(f) = W(g)$ and $T(f) = T(g)$.*

This means that to reduce the unknot problem to the word problem for $\mathbb{ROT}\mathtt{ang}$, we simply need to be able to tweak diagrams to adjust their writhe and turning number without changing their knot-isotopy class. This is what the following section establishes.

Unknotting in braided pivotal categories

Lemma 4.123. *Given a writhe w and a turning number t such that $2w + t$ is a multiple of 4, we can construct a morphism $f \in \mathbb{ROT}\mathtt{ang}(\uparrow, \uparrow)$ such that $W(f) = w$ and $T(f) = t$, and f is knot-isotopic to the identity.*

Proof. We first define the following morphisms in $\mathbb{ROT}\mathtt{ang}(\uparrow, \uparrow)$:

$a =$  $b =$  $c =$  $d =$ 

They have the following invariants:

$$W(a) = +1 \qquad W(b) = -1 \qquad W(c) = +1 \qquad W(d) = -1$$
$$T(a) = +2 \qquad T(b) = -2 \qquad T(c) = -2 \qquad T(d) = +2$$

Let $w, t \in \mathbb{Z}$ such that $2w + t$ is a multiple of 4. We construct the required morphism $f \in \mathbb{ROT}\mathtt{ang}(\uparrow, \uparrow)$ by composition of a, b, c and d using the fact that $W(g \circ h) = W(g) + W(h)$ and $T(g \circ h) = T(g) + T(h)$ for all $g, h \in \mathbb{ROT}\mathtt{ang}(\uparrow, \uparrow)$. Let $p = \frac{2w+t}{4}$. If p is positive, we start by p copies of a, otherwise $-p$ copies of b. Then, let $q = \frac{2w-t}{4}$. If q is positive, we continue with q copies of c, otherwise $-q$ copies of d. One can check that the composite has the required writhe and turning number. □

Corollary 4.124. *The knot isotopy problem can be reduced to the word problem for $\mathbb{ROT}\mathtt{ang}$.*

Proof. Given two knots k, l represented as crossing diagrams in the plane, pick an orientation for them and turn them into morphisms $f, g \in \mathbb{ROT}\mathtt{ang}$. We can compute the writhe and turning number of f and g in polynomial time.

As noted by Trace (1983), for any oriented knot f, $\frac{2W(f)+T(f)}{2}$ is odd. In other words there are $p, q \in \mathbb{Z}$ such that $2W(f)+T(f) = 4p+2$ and $2W(g)+T(g) = 4q+2$. Therefore, $2(W(f) - W(g)) + (T(f) - T(g)) = 4(p - q)$. By Lemma 4.123, we can

therefore construct a morphism $h \in \mathbb{ROTang}(\uparrow, \uparrow)$ such that $W(h) = W(f) - W(g)$ and $T(h) = T(f) - T(g)$, and such that h is knot-isotopic to a straight wire.

Therefore, we can insert h on any strand of g, obtaining a morphism g' which represents the same knot as g, such that $W(g) = W(f)$ and $T(g) = T(f)$. By by Theorem 4.122, f and g' are knot-isotopic if and only if they are equal as morphisms of $\mathbb{ROTang}$. This completes the proof. □

4.4.3 Reducing the unknotting problem to the braided monoidal word problem

So far, Corollary 4.124 only reduces the unknot problem to the word problem for $\mathbb{ROTang}$ while our goal is to reduce it to the word problem for braided monoidal categories. The category $\mathbb{ROTang}$ can be presented as a free braided monoidal category, but that requires additional equations between the generators representing the caps and cups. In this section, we show how these equations can be eliminated too. We call *unknot diagram* any knot diagram which is knot-isotopic to the unknot.

Definition 4.125. *The category $\mathbb{CC}$ is the free braided monoidal category generated by objects $\{\uparrow, \downarrow\}$ and morphisms* {  ,  ,  ,  }.

It is important to note that no equations are imposed between the morphism generators, unlike in $\mathbb{ROTang}$. Therefore, there exists a functor from $\mathbb{CC}$ to $\mathbb{ROTang}$, mapping the generators of $\mathbb{CC}$ to the corresponding units and counits in $\mathbb{ROTang}$, but the reverse mapping would not be functorial.

Cap-cup cycles

In this section we introduce a more precise invariant than the turning number: the sequence of caps and cups encountered while following the strand of a knot.

Definition 4.126. *A **cap-cup cycle** is a finite sequence of elements in* { ,  ,  ,  } *considered up to cyclic permutation, such that caps and cups alternate. The turning number of a cap-cup cycle is the sum of the turning numbers of its elements, defined as in Definition 4.121.*

The cap-cup cycle is intended to replace the turning number in a context where eliminating caps and cups using the adjunction equations is not allowed.

Definition 4.127. *Given a knot $f \in \mathbb{CC}$, its cap-cup cycle* $\mathrm{ccc}(f)$ *can be obtained by starting from any strand in f, following it in the direction indicated by its type and recording all the caps and cups encountered until one travels back to the starting point. This cycle is invariant under all axioms of a braided monoidal category.*

For instance, the knot of Figure 4.31 has cap-cup cycle (↰ , ↳ , ↱ , ↲).

Lemma 4.128. *For all knot diagrams $f \in \mathbb{CC}$,* $\mathrm{ccc}(f)$ *is of even length, and* $T(\mathrm{ccc}(f)) = T(f)$.

Lemma 4.129. *For all cap-cup cycles c such that $T(c) = \pm 2$, one can construct a knot diagram $f \in \mathbb{CC}$ without any crossings, such that* $\mathrm{ccc}(f) = c$.

Proof. By induction on the length of the cycle c. If $|c| = 2$, then $c = ($ ↱ , ↲ $)$ or $c = ($ ↰ , ↳ $)$, both of which can be realized by the composite of both elements. If $|c| > 2$, then there is an element $x \in c$ such that $T(x) = +1$ and another element $y \in c$ with $T(y) = -1$. One can also assume that they are adjacent in c. Indeed, if all elements $x \in c$ with $T(x) = +1$ are such that the elements on their left and right also have a positive turning number, then by propagating this, all elements in the cycle have a positive turning number, which is a contradiction.

Consider the cycle c' obtained by removing x and y from c. By induction, construct a knot diagram $f' \in \mathbb{CC}$ such that the cap-cup cycle of f' is c'. Now, at the point where we removed x and y, we can insert in f' a zigzag corresponding to x and y (in the order they appeared in c), which gives us the required knot. □

To generalize this lemma to knot diagrams with crossings, we introduce a new notion of cap-cup cycle where each cap or cup can carry its own writhe.

Definition 4.130. *The set of* ***twisted cap-cups*** *is* $\mathbb{T} \coloneqq \{$ ↰ , ↱ , ↲ , ↳ $\} \times \mathbb{Z}$.

(a) $(\ \cap\ , 2)$ **(b)** $(\ \cup\ , -1)$ **(c)** $(\ \cup\ , 0)$ **(d)** $(\ \cap\ , -2)$

Figure 4.34: Examples of twisted cap-cups

We think of a pair $(c, w) \in \mathbb{T}$ as a cap-cup c composed with braids such that the writhe of the resulting morphism is w. Figure 4.34 gives a few examples of twisted cap-cups.

Definition 4.131. *The **turning number** of a twisted cap-cup $(c, w) \in \mathbb{T}$ is defined as $T((c, w)) = (-1)^{|w|}t(c)$. The **writhe** of a twisted cap-cup is $W((c, w)) = w$. The **signature** of a twisted cap-cup is $S((c, w)) = c$ if w is even, and c with a flipped wire orientation if w is odd.*

The signature of a twisted cap-cup is essentially obtained by applying Reidemeister type I moves to the twisted cap-cup until no crossing remains. Therefore, this preserves the domain and codomain of the morphism.

Definition 4.132. *A **twisted cap-cup cycle** is a finite sequence of twisted cap-cups up to cyclic permutation, such that caps and cups alternate. The turning number of a cap-cup cycle is the sum of the turning numbers of its elements, and similarly for its writhe.*

Definition 4.133. *Given a twisted cap-cup cycle c we define a cap-cup cycle $U(c)$ obtained by forgetting the writhe component in each twisted cap-cup. We also define a cap-cup cycle $S(c)$ obtained by taking the signature of each twisted cap-cup in the cycle.*

Lemma 4.134. *Let c be a twisted cap-cup cycle such that $T(c) = \pm 2$. One can construct an unknot diagram $R(c) \in \mathbb{CC}$ such that $W(R(c)) = W(c)$ and $\mathrm{ccc}(R(c)) = U(c)$.*

Proof. First, notice that for all twisted cap-cup cycle c, $T(S(c)) = T(c)$. So if $T(c) = \pm 2$ then $T(S(c)) = \pm 2$ and we can apply Lemma 4.129 to $S(c)$, obtaining a morphism f such that $\mathrm{ccc}(f) = S(c)$. Now we obtain another knot diagram $R(c)$ by replacing each cap and cup of f by the twisted cap-cup in c it was generated from. This is possible because taking the signature of a twisted cap-cup preserves the domain and codomain of the corresponding morphism. We therefore obtain $W(R(c)) = W(c)$ and $\mathrm{ccc}(R(c)) = U(c)$ as required. □

Lemma 4.135. *Let c be a cap-cup cycle and $w \in \mathbb{Z}$ such that $w + \frac{T(c)}{2}$ is odd. Then we can construct an unknot diagram $f(c, w) \in \mathbb{CC}$ such that $W(f(c, w)) = w$ and $\mathrm{ccc}(f(c, w)) = c$.*

Proof. We can view c as a twisted cap-cup cycle where all the writhe components are null. We will transform c to incorporate the writhe w in the writhe components of the cycle.

First, consider the case where $T(c) = \pm 2$. By assumption, w is therefore even. We can pick any element (a, b) of c and replace it by $(a, b + w)$, giving us a new twisted cap-cup cycle c'. We have $T(c') = T(c) = \pm 2$, so we can apply Lemma 4.134, giving the required morphism $R(c') =: f(c, w)$.

Second, if $T(c) = 0$. By assumption, w is odd. Again, take any element (a, b) in c and replace it by $(a, b + w)$. This changes the turning number of that element, negating its sign. Therefore, the turning number of the new twisted cap-cup cycle is ± 2, and we are back to the previous case.

Third, if $|T(c)| > 2$. By symmetry let us assume $T(c) > 2$. We work by induction on $T(c)$. There are at least two elements of c with turning number $+1$, let them be (a, b) and (a', b'). We replace them by $(a, b+1)$ and $(a', b'-1)$ respectively. We have $t((a, b + 1)) = t((a', b' - 1)) = -1$ so this reduces the turning number by 4, keeps the writhe unchanged and keeps $U(c)$ unchanged. So we can obtain the required morphism by induction. □

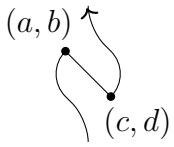
The following lemma establishes that the way the writhe is spread on the elements of a twisted cap-cup cycle does not actually matter. The writhe can be transferred between any two elements without resorting to Reidemeister 1 or zigzag elimination.

Lemma 4.136. *Let c, c' be twisted cap-cup cycles such that $W(c) = W(c')$, $T(c) = T(c') = \pm 2$ and $U(c) = U(c')$. Then $R(c)$ is isotopic to $R(c')$ via the axioms of braided monoidal categories.*

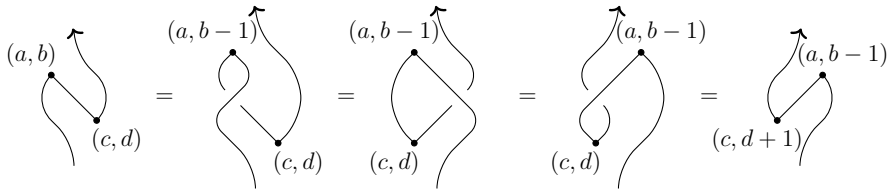
Proof. We define a relation $\diamond$ on twisted cap-cup cycles: $c \diamond c'$ when c' can be obtained from c by replacing two consecutive elements $(a, b), (c, d)$ by $(a, b-1), (c, d+1)$.

Let c, c' be twisted cap-cup cycles as in the lemma. We first show that if $c \diamond c'$ then $R(c)$ is isotopic to $R(c')$ as a braided monoidal morphism.

If $T((a, b)) = -T((c, d))$ then the sequence $(a, b), (c, d)$ is realized in $R(c)$ as follows, up to vertical and horizontal symmetries:

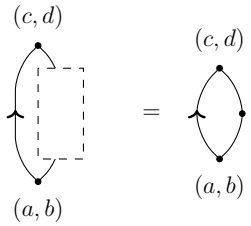


where the morphisms are composed of a single cap or cup, followed by braids to obtain the desired writhe. We have the following isotopy:

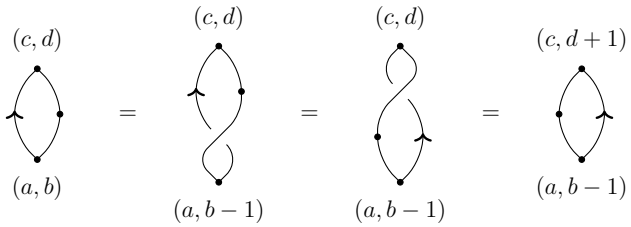


Note that the first and last equalities are not Reidemeister I moves: they can simply be expressed as unboxing the composite morphisms (a, b) and $(c, d+1)$, possibly with the help of Reidemeister II moves to create braids when required. This shows that $R(c)$ is isotopic to $R(c')$ as braided monoidal morphism.

If $T((a,b)) = T((c,d))$ then the sequence $(a,b),(c,d)$ is realized in $R(c)$ as follows, again up to vertical and horizontal symmetries:



The dashed area in the left-hand side represents the rest of the knot. Because by construction we know that it does not cross the wire passing on its left, nor is it connected with anything else, we can abstract it away as a simple morphism taking one wire as input and one wire as output, as in the right-hand side. Then:



So again $R(c)$ is isotopic to $R(c')$.

So the $\diamond$ relation respects braided monoidal isotopy. But now, by assumption $W(c) = W(c')$ and $U(c) = U(c')$. By a sequence of $\diamond$ steps one can transfer the writhe of any element of c to any other element. So c and c' are related by a series of $\diamond$ steps, so they are equal as braided monoidal morphisms. □

Bridge isotopy

In this section, we introduce a notion of knot isotopy which forbids the elimination of caps and cups, but still allows Reidemeister I moves.

Definition 4.137. *A knot diagram $k \in \mathbb{CC}$ is in **bridge position** if all caps appear above of all cups in its string diagram. The number of caps (or equivalently cups) is called the **bridge number** of the diagram.*

(a) A knot diagram not in bridge position

(b) An equivalent diagram in bridge position

Figure 4.35: Bridge position

For instance, all knot diagrams of Figure 4.32 are in bridge position. Figure 4.35 shows a knot diagram that is not in bridge position and an equivalent diagram in bridge position. The following lemma shows that any knot diagram can be put in bridge position without cancelling any zigzag, as illustrated by Figure 4.35.

Lemma 4.138. *Any knot diagram $k \in \mathbb{CC}$ can be expressed in bridge position via the axioms of braided monoidal categories.*

Proof. While there is a cap or cup that is not on the first or last slice of the diagram, pull the cup down or pull the cap up using the pull-through move (naturality of the braid). This move can be executed regardless of the surroundings of the cap or cup. □

Note that bridge positions are not unique and there are generally multiple pull-through moves available to pull a cap or cup towards the boundary of the diagram.

Definition 4.139 (Otal, 1982; Jang et al., 2019)**.** *A **bridge isotopy** between two knot diagrams in bridge position is a sequence of moves (including Reidemeister I) such that at each step the diagram is in bridge position.*

Note that because cups and caps are required to stay apart throughout the isotopy, the bridge number of the diagram is preserved by bridge isotopy.

Theorem 4.140 (Otal, 1982)**.** *Let K, K' be two diagrams of the unknot in bridge position, with equal bridge number. Then they are in bridge isotopy.*

Unknotting with braided monoidal categories

We can now combine the results above to establish a polynomial time reduction between the unknotting problem and the word problem for braided monoidal categories.

Lemma 4.141. *Let k be a diagram of the unknot. Then it is braided monoidal isotopic to $f(\mathrm{ccc}(k), W(k))$.*

Proof. Recall that $f(\mathrm{ccc}(k), W(k))$ is the diagram of the unknot constructed in Lemma 4.135 so that its cap-cup cycle and writhe match that of k.

First, the bridge number of k and $f(\mathrm{ccc}(k), W(k))$ are equal since $\mathrm{ccc}(f(\mathrm{ccc}(k), W(k))) = \mathrm{ccc}(k)$. So by Theorem 4.140, the two diagrams are in bridge isotopy. This is not quite enough for us since this bridge isotopy might contain Reidemeister I moves, which are not allowed in braided monoidal isotopy.

To get rid of those Reidemeister I moves, we follow the same approach as Theorem 4.122. First, we view all caps and cups present at all stages of the isotopy as twisted cap or cups with a null writhe component. Then, scanning the knot-isotopy from start to end, we replace Reidemeister I moves by identities (when the Reidemeister I move cancels a braiding) or by Reidemeister II moves (when the Reidemeister I move introduces a braiding). Doing so, we bundle up the leftover braid with the cap or cup in the writhe component of the twisted cap-cup.

Since the isotopy is a bridge isotopy, caps and cups never get cancelled so adding this writhe component does not prevent any further step of the isotopy.

After this transformation, the target of the isotopy might have some additional writhe components on some caps and cups. But the original target was $f(\mathrm{ccc}(k), W(k))$, which was defined as $R(c)$, the realization of a twisted cap-cup cycle c. So the new target can also be seen as the realization of another twisted cap-cup cycle c', which has identical writhe and cap-cup cycle, because it is in braided monoidal isotopy with the source. Therefore, we can apply Lemma 4.136

and obtain a braided monoidal isotopy between the new target of our isotopy and $f(\mathrm{ccc}(k), W(k))$, completing the proof. □

Theorem 4.142. *The unknotting problem can be polynomially reduced to the word problem for braided monoidal categories.*

Proof. Given a knot diagram k, we convert it to a braided monoidal word problem as follows. First, we orient it in an arbitrary way, obtaining a morphism $k' \in \mathbb{ROTang}$. We compute its writhe $W(k')$ and cap-cup cycle $\mathrm{ccc}(k')$. Then we compute $f(\mathrm{ccc}(k'), W(k'))$. All these steps can be done in polynomial time. A summary of those steps on a concrete example is given in Figure 4.36. Finally, the corresponding word problem is to determine whether k and $f(\mathrm{ccc}(k'), W(k'))$ are in braided monoidal isotopy. If they are, then k is the unknot. If they are not then by Lemma 4.141, k is knotted. □

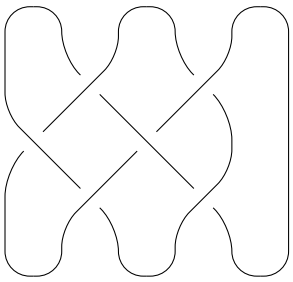
Corollary 4.143. *The word problem for the 3-cells of free Gray categories is at least as hard as the unknotting problem.*

Proof. Implied by the characterization of doubly degenerate Gray categories as braided monoidal categories (Gurski and Cheng, 2011). □

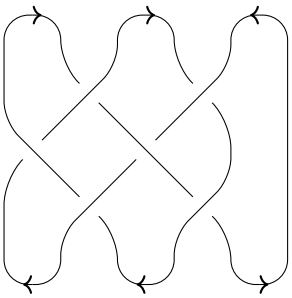
4.4.4 Conclusion

We have established a connection between two areas. On one side, word problems arising naturally in category theory, which have not been studied much from a computational perspective so far. On the other side, the unknotting problem, which has been studied by knot theorists for more than a century.

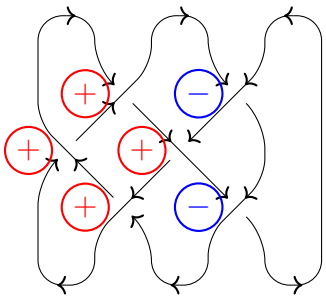
Our hope with this connection is to make it evident that much more work is required on word problems in category theory, especially if we hope to develop practical proof assistants for higher categories. To our knowledge, no algorithm for the braided monoidal word problem is known to date. Although we do not see an obvious way to get one, it seems intuitively clear that given two equivalent braided monoidal diagrams, there is no point introducing large quantities of new



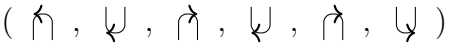
(a) Initial knot diagram



(b) Oriented knot as a morphism in $\mathbb{ROT}\mathtt{ang}$



(c) Computing the writhe: $+2$ in this case

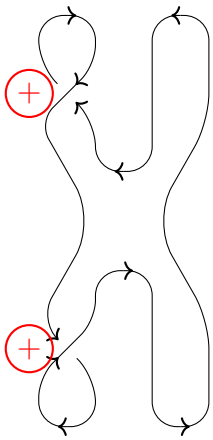


(d) The cap-cup cycle

$((\ \cap\ , 0), (\ \cup\ , 0), (\ \cap\ , 1),$

$(\ \cup\ , 1), (\ \cap\ , 0), (\ \cup\ , 0))$

(e) The twisted cap-cup cycle of Lemma 4.135



(f) The realization of Lemma 4.134

Figure 4.36: Entire process of detecting knotedness using a solution to the braided monoidal word problem

crossings to prove that they are equivalent. There should be a way to bound the number of steps of equivalence proofs, and similar results exist for knots (Coward and Lackenby, 2014).

Conjecture 4.144. *The word problem for 3-cells of Gray categories (and hence cells of braided monoidal categories) is decidable.*

Again, the word problem we mean here is deciding the equality of morphisms up to the axioms of Gray categories and nothing else. Note that the naive algorithm consisting in exploring all expressions reachable from a given expression does not terminate, since the Reidemeister II move can be applied indefinitely. This makes approaches such as that of Makkai (2005) inapplicable. Furthermore, it is possible that the word problem becomes undecidable at a higher level, perhaps for similar reasons that the isotopy of four-dimensional manifolds is undecidable (Markov, 1958; Boone et al., 1968).

Another natural question arising from our work is whether the problem of knot equivalence could be reduced to the word problem for braided monoidal categories. Knot equivalence is the problem of determining if two knot diagrams represent the same knot. In this context, it seems more difficult to suppress the need for the yanking equations, so our results do not seem to adapt easily to this more general case.

Appendices

A

Coherence axioms of bimonoidal categories

The following coherence axioms are taken from Laplaza (1972). Their axioms (II) and (XV) were removed as they only apply to symmetric bimonoidal categories. In the following, the monoidal structure $(\mathcal{C}, \cdot, I)$ has coherence isomorphisms

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

$$\lambda_A : IA \to A$$

$$\rho_A : AI \to A$$

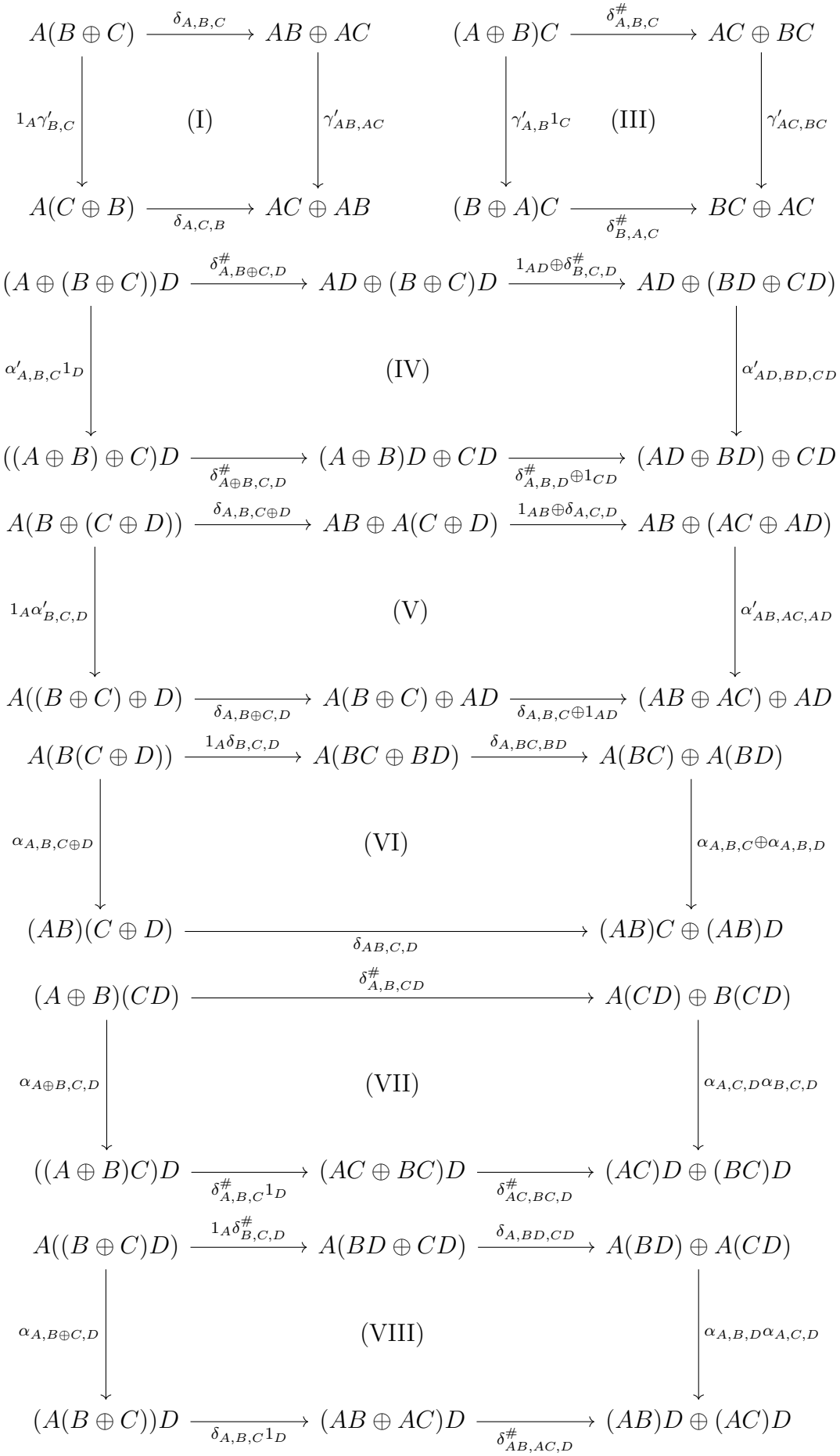
and the monoidal structure $(\mathcal{C}, \oplus, O)$ has coherence isomorphisms

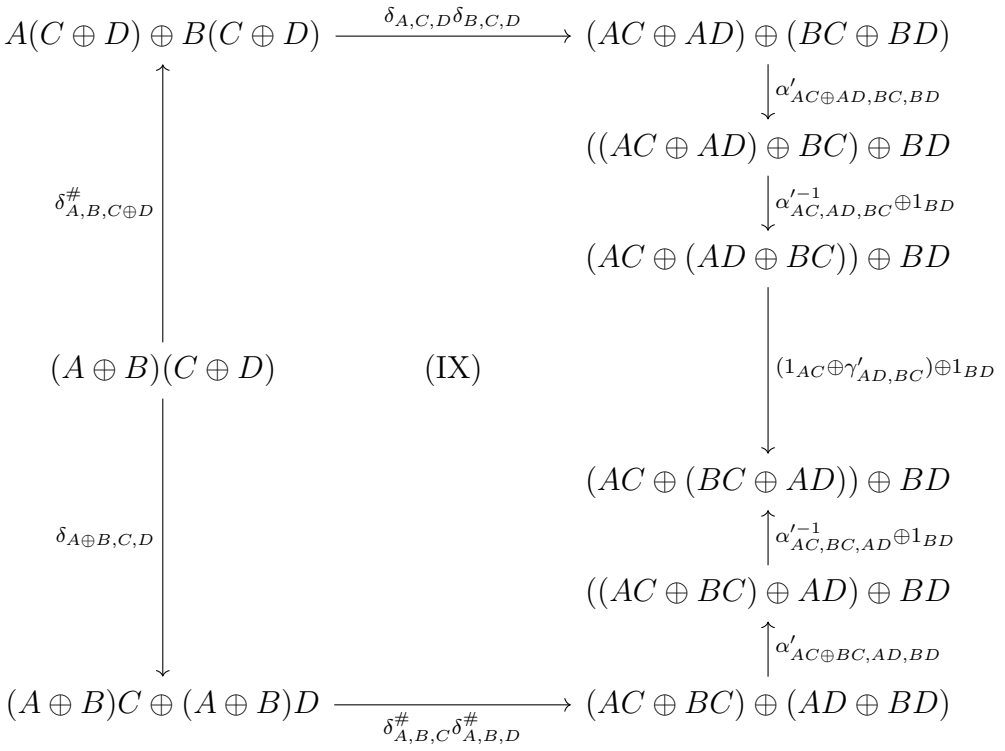
$$\alpha'_{A,B,C} : A \oplus (B \oplus C) \to (A \oplus B) \oplus C$$

$$\lambda'_A : O \oplus A \to A$$

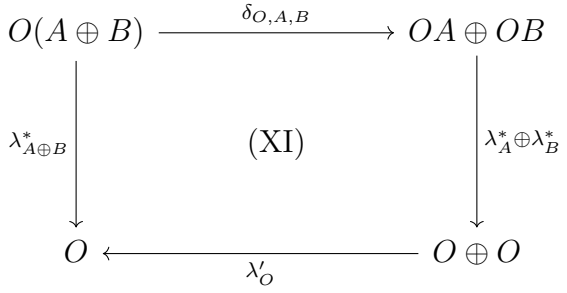
$$\rho'_A : A \oplus O \to A$$

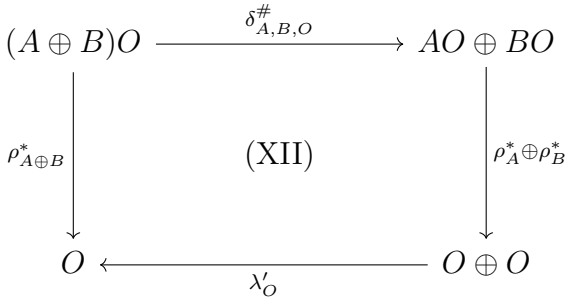
$$\gamma'_{A,B} : A \oplus B \to B \oplus A$$


$A(B \oplus C)$
$\delta_{A,B,C}$
$AB \oplus AC$
$(A \oplus B)C$
$\delta^{\#}_{A,B,C}$
$AC \oplus BC$
$1_A\gamma'_{B,C}$
(I)
$\gamma'_{AB,AC}$
$\gamma'_{A,B}1_C$
(III)
$\gamma'_{AC,BC}$
$A(C \oplus B)$
$\delta_{A,C,B}$
$AC \oplus AB$
$(B \oplus A)C$
$\delta^{\#}_{B,A,C}$
$BC \oplus AC$
$(A \oplus (B \oplus C))D$
$\delta^{\#}_{A,B\oplus C,D}$
$AD \oplus (B \oplus C)D$
$1_{AD} \oplus \delta^{\#}_{B,C,D}$
$AD \oplus (BD \oplus CD)$
$\alpha'_{A,B,C}1_D$
(IV)
$\alpha'_{AD,BD,CD}$
$((A \oplus B) \oplus C)D$
$\delta^{\#}_{A\oplus B,C,D}$
$(A \oplus B)D \oplus CD$
$\delta^{\#}_{A,B,D} \oplus 1_{CD}$
$(AD \oplus BD) \oplus CD$
$A(B \oplus (C \oplus D))$
$\delta_{A,B,C\oplus D}$
$AB \oplus A(C \oplus D)$
$1_{AB} \oplus \delta_{A,C,D}$
$AB \oplus (AC \oplus AD)$
$1_A\alpha'_{B,C,D}$
(V)
$\alpha'_{AB,AC,AD}$
$A((B \oplus C) \oplus D)$
$\delta_{A,B\oplus C,D}$
$A(B \oplus C) \oplus AD$
$\delta_{A,B,C} \oplus 1_{AD}$
$(AB \oplus AC) \oplus AD$
$A(B(C \oplus D))$
$1_A\delta_{B,C,D}$
$A(BC \oplus BD)$
$\delta_{A,BC,BD}$
$A(BC) \oplus A(BD)$
$\alpha_{A,B,C\oplus D}$
(VI)
$\alpha_{A,B,C} \oplus \alpha_{A,B,D}$
$(AB)(C \oplus D)$
$\delta_{AB,C,D}$
$(AB)C \oplus (AB)D$
$(A \oplus B)(CD)$
$\delta^{\#}_{A,B,CD}$
$A(CD) \oplus B(CD)$
$\alpha_{A\oplus B,C,D}$
(VII)
$\alpha_{A,C,D}\alpha_{B,C,D}$
$((A \oplus B)C)D$
$\delta^{\#}_{A,B,C}1_D$
$(AC \oplus BC)D$
$\delta^{\#}_{AC,BC,D}$
$(AC)D \oplus (BC)D$
$A((B \oplus C)D)$
$1_A\delta^{\#}_{B,C,D}$
$A(BD \oplus CD)$
$\delta_{A,BD,CD}$
$A(BD) \oplus A(CD)$
$\alpha_{A,B\oplus C,D}$
(VIII)
$\alpha_{A,B,D}\alpha_{A,C,D}$
$(A(B \oplus C))D$
$\delta_{A,B,C}1_D$
$(AB \oplus AC)D$
$\delta^{\#}_{AB,AC,D}$
$(AB)D \oplus (AC)D$



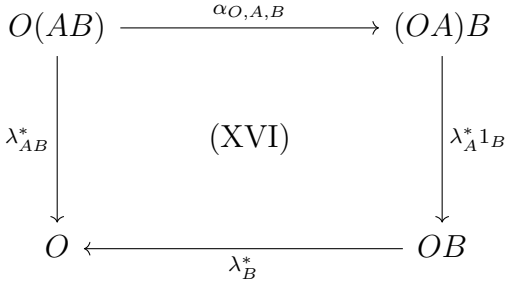
(X): $\lambda^*_O = \rho^*_O : O \otimes O \to O$





(XIII): $\rho_O = \lambda^*_O : OI \to O$

(XIV): $\lambda_O = \rho^*_O : IO \to O$



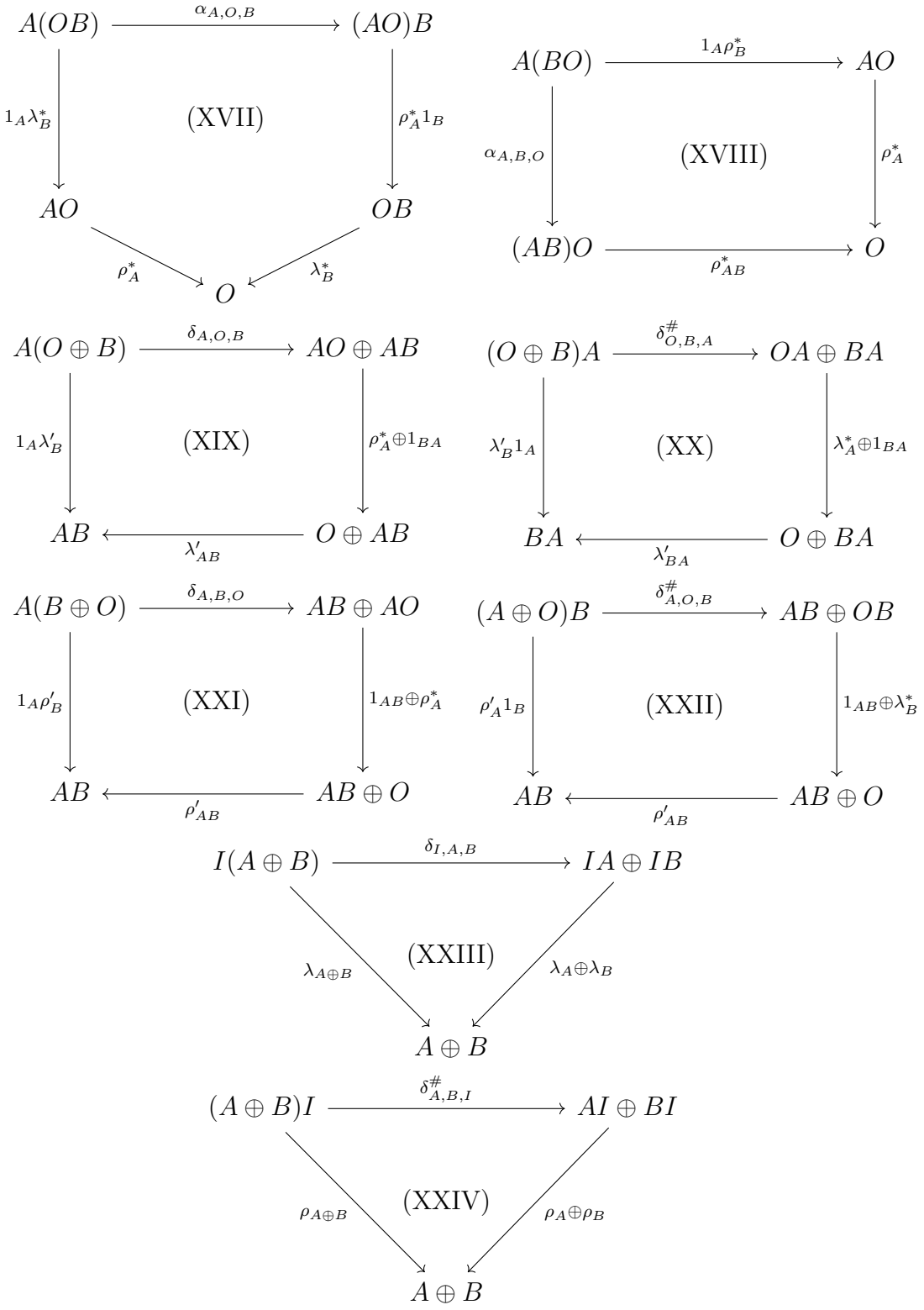


Figure A.1: Coherence axioms

A.1 Axioms of a bimonoidal functor

Definition A.1. *(Elgueta, 2020, Def. 2.5.1) A* (strong) bimonoidal functor *between bimonoidal categories* $(\mathcal{C}, \cdot^{\mathcal{C}}, I^{\mathcal{C}}, \oplus^{\mathcal{C}}, O^{\mathcal{C}})$ *and* $(\mathcal{D}, \cdot^{\mathcal{D}}, I^{\mathcal{D}}, \oplus^{\mathcal{D}}, O^{\mathcal{D}})$ *is a functor* $F : \mathcal{C} \to \mathcal{D}$*, isomorphisms*

$$\varepsilon^{\cdot} : I^{\mathcal{D}} \to F(I^{\mathcal{C}}) \qquad \varepsilon^{\oplus} : O^{\mathcal{D}} \to F(O^{\mathcal{C}})$$

and natural isomorphisms

$$\mu^{\cdot}_{A,B} : F(A) \cdot^{\mathcal{D}} F(B) \to F(A \cdot^{\mathcal{C}} B) \qquad \mu^{\oplus}_{A,B} : F(A) \oplus^{\mathcal{D}} F(B) \to F(A \oplus^{\mathcal{C}} B)$$

So that $(F, \eta^{\cdot}, \mu^{\cdot})$ *is a strong monoidal functor from* $(\mathcal{C}, \cdot^{\mathcal{C}}, I^{\mathcal{C}})$ *to* $(\mathcal{D}, \cdot^{\mathcal{D}}, I^{\mathcal{D}})$*,* $(F, \eta^{\oplus}, \mu^{\oplus})$ *is a strong monoidal functor from* $(\mathcal{C}, \oplus^{\mathcal{C}}, O^{\mathcal{C}})$ *to* $(\mathcal{D}, \oplus^{\mathcal{D}}, O^{\mathcal{D}})$*, where the following diagrams also commute:*

$$\begin{array}{ccccc}
FA \cdot^{\mathcal{D}} F(B \oplus^{\mathcal{C}} C) & \xrightarrow{1_{FA} \cdot^{\mathcal{D}} \mu^{\oplus}_{B,C}} & FA \cdot^{\mathcal{D}} (FB \oplus^{\mathcal{C}} FC) & \xrightarrow{\delta^{\mathcal{D}}_{FA,FB,FC}} & FA \cdot^{\mathcal{D}} FB \oplus^{\mathcal{D}} FA \cdot^{\mathcal{D}} FC \\
\downarrow{\scriptstyle \mu^{\cdot}_{A,B \oplus^{\mathcal{C}} C}} & & & & \downarrow{\scriptstyle \mu^{\cdot}_{A,B} \oplus^{\mathcal{D}} \mu^{\cdot}_{A,C}} \\
F(A \cdot^{\mathcal{C}} (B \oplus^{\mathcal{C}} D)) & \xrightarrow[F(\delta^{\mathcal{C}}_{A,B,C})]{} & F(A \cdot^{\mathcal{C}} B \oplus^{\mathcal{C}} A \cdot^{\mathcal{C}} D) & \xrightarrow[\mu^{\oplus}_{A \cdot^{\mathcal{C}} B, A \cdot^{\mathcal{D}} C}]{} & F(A \cdot^{\mathcal{C}} B) \oplus^{\mathcal{D}} F(A \cdot^{\mathcal{C}} C)
\end{array}$$

$$\begin{array}{ccccc}
FA \cdot^{\mathcal{D}} FO^{\mathcal{C}} & \xrightarrow{1_{FA} \cdot^{\mathcal{D}} \varepsilon^{\oplus}} & FA \cdot^{\mathcal{D}} O^{\mathcal{D}} & \xrightarrow{\rho^{*,\mathcal{D}}_{O}} & O^{\mathcal{D}} \\
\downarrow{\scriptstyle \mu^{\cdot}_{A,O^{\mathcal{C}}}} & & & & \| \\
F(A \cdot^{\mathcal{C}} O^{\mathcal{C}}) & \xrightarrow[\rho^{*,\mathcal{C}}_{O}]{} & FO^{\mathcal{C}} & \xrightarrow[\varepsilon^{\oplus}]{} & O^{\mathcal{D}}
\end{array}$$

$$\begin{array}{ccccc}
F(A \oplus^{\mathcal{C}} B) \cdot^{\mathcal{D}} FC & \xrightarrow{\mu^{\oplus}_{A,B} \cdot^{\mathcal{D}} 1_{FC}} & (FA \oplus^{\mathcal{C}} FB) \cdot^{\mathcal{D}} FC & \xrightarrow{\delta^{\#,\mathcal{D}}_{FA,FB,FC}} & FA \cdot^{\mathcal{D}} FC \oplus^{\mathcal{D}} FB \cdot^{\mathcal{D}} FC \\
\downarrow{\scriptstyle \mu^{\cdot}_{A \oplus^{\mathcal{C}} B, C}} & & & & \downarrow{\scriptstyle \mu^{\cdot}_{A,C} \oplus^{\mathcal{D}} \mu^{\cdot}_{B,C}} \\
F((A \oplus^{\mathcal{C}} B) \cdot^{\mathcal{C}} C)) & \xrightarrow[F(\delta^{\#,\mathcal{C}}_{A,B,C})]{} & F(A \cdot^{\mathcal{C}} C \oplus^{\mathcal{C}} B \cdot^{\mathcal{C}} C) & \xrightarrow[\mu^{\oplus}_{A \cdot^{\mathcal{C}} C, B \cdot^{\mathcal{D}} C}]{} & F(A \cdot^{\mathcal{C}} C) \oplus^{\mathcal{D}} F(B \cdot^{\mathcal{C}} C)
\end{array}$$

$$\begin{array}{ccccc}
FO^{\mathcal{C}} \cdot^{\mathcal{D}} FA & \xrightarrow{\varepsilon^{\oplus} \cdot^{\mathcal{D}} 1_{FA}} & O^{\mathcal{D}} \cdot^{\mathcal{D}} FA & \xrightarrow{\lambda^{*,\mathcal{D}}_{O}} & O^{\mathcal{D}} \\
\downarrow{\scriptstyle \mu^{\cdot}_{O^{\mathcal{C}},A}} & & & & \| \\
F(O^{\mathcal{C}} \cdot^{\mathcal{C}} A) & \xrightarrow[\lambda^{*,\mathcal{C}}_{O}]{} & FO^{\mathcal{C}} & \xrightarrow[\varepsilon^{\oplus}]{} & O^{\mathcal{D}}
\end{array}$$

Bibliography

Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *LICS Proceedings*, pages 415–425. IEEE Computer Society, 2004. doi:10.1109/lics.2004.1319636. arXiv:quant-ph/0402130.

Emil Artin. Theory of braids. *Ann. of Math*, 48(2):101–126, 1947. doi:10.2307/1969218.

Krzysztof Bar, Aleks Kissinger, and Jamie Vicary. Globular: An online proof assistant for higher-dimensional rewriting. *LMCS*, December 2016. doi:10.23638/LMCS-14(1:8)2018. arXiv:1612.01093.

W. W. Boone, W. Haken, and V. Poénaru. On Recursively Unsolvable Problems in Topology and Their Classification. In H. Arnold Schmidt, K. Schütte, and H. J. Thiele, editors, *Studies in Logic and the Foundations of Mathematics*, volume 50 of *Contributions to Mathematical Logic*, pages 37–74. Elsevier, 1968. doi:10.1016/S0049-237X(08)70518-4.

Francis Borceux. *Handbook of Categorical Algebra*, volume Volume 1 of *Encyclopedia of Mathematics and Its Applications 50-51, 53 [i.e. 52]*. Cambridge University Press, 1994. ISBN 0-521-44178-1 978-0-521-44178-0 0-521-44179-X 0-521-44180-3.

Roberto Bruni, José Meseguer, and Ugo Montanari. Symmetric monoidal and cartesian double categories as a semantic framework for tile logic. *Mathematical Structures in Computer Science*, 12(1):53–90, February 2002. ISSN 1469-8072, 0960-1295. doi:10.1017/S0960129501003462.

Albert Burroni. Higher-dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, July 1993. ISSN 0304-3975. doi:10.1016/0304-3975(93)90054-W.

Kenta Cho, Bart Jacobs, Bas Westerbaan, and Abraham Westerbaan. An Introduction to Effectus Theory. *arXiv:1512.05813 [quant-ph]*, December 2015. arXiv:1512.05813.

Stephen Clark, Bob Coecke, and Mehrnoosh Sadrzadeh. A compositional distributional model of meaning. In *Proceedings of the Second Quantum Interaction Symposium (QI-2008)*, pages 133–140, 2008.

Bob Coecke. Quantum Picturalism. *Contemporary Physics*, 51(1):59–83, January 2010. ISSN 0010-7514, 1366-5812. doi:10.1080/00107510903257624. arXiv:0908.1787.

Cole Comfort, Antonin Delpeuch, and Jules Hedges. Sheet diagrams for bimonoidal categories. *arXiv:2010.13361 [math]*, December 2020. arXiv:2010.13361.

Alexander Coward and Marc Lackenby. An upper bound on Reidemeister moves. *American Journal of Mathematics*, 136(4):1023–1066, 2014. ISSN 1080-6377. doi:10.1353/ajm.2014.0027.

R. Dawson and R. Paré. Characterizing tileorders. *Order*, 10(2):111–128, June 1993. ISSN 1572-9273. doi:10.1007/BF01111295.

R. J. M. Dawson, R. Paré, and D. A. Pronk. Free extensions of double categories. *Cahiers de topologie et géométrie différentielle catégoriques*, 45(1):35–80, 2004.

Robert Dawson. A forbidden-suborder characterization of binarily-composable diagrams in double categories. *Theory and Applications of Categories [electronic only]*, 1: 146–155, 1995. ISSN 1201-561X.

Robert Dawson and Robert Pare. General associativity and general composition for double categories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 34 (1):57–79, 1993.

Robert Dawson and Robert Paré. What is a free double category like? *Journal of Pure and Applied Algebra*, 168(1):19–34, March 2002. ISSN 0022-4049. doi:10.1016/S0022-4049(01)00049-4.

Giovanni de Felice, Alexis Toumi, and Bob Coecke. Discopy: Monoidal categories in Python. *arXiv preprint arXiv:2005.02975*, 2020. doi:10.4204/EPTCS.333.13. arXiv:2005.02975.

M. Dehn. über die topologie des dreidimensionalen raumes. *Mathematische Annalen*, 69: 137–168, 1910. doi:10.1007/BF01455155.

Patrick Dehornoy. Efficient solutions to the braid isotopy problem. *arXiv:math/0703666*, March 2007. doi:10.1016/j.dam.2007.12.009. arXiv:math/0703666.

Antonin Delpeuch. A complete language for faceted dataflow programs. In *arXiv:1906.05937 [Cs, Math]*, Electronic Proceedings in Theoretical Computer Science, Oxford, June 2019. Open Publishing Association. doi:10.4204/eptcs.323.1. arXiv:1906.05937.

Antonin Delpeuch. The word problem for double categories. *Theory and Applications of Categories*, 35:1–18, 2020. ISSN 1201-561X. arXiv:1907.09927.

Antonin Delpeuch and Jamie Vicary. Normalization for planar string diagrams and a quadratic equivalence algorithm. *to appear in Logical Methods in Computer Science*, April 2018. arXiv:1804.07832.

Antonin Delpeuch and Jamie Vicary. The word problem for braided monoidal categories is unknot-hard. *arXiv:2105.04237 [math]*, May 2021. arXiv:2105.04237.

Lucas Dixon, Ross Duncan, and Aleks Kissinger. Open Graphs and Computational Reasoning. *Electronic Proceedings in Theoretical Computer Science*, 26:169–180, June 2010. ISSN 2075-2180. doi:10.4204/EPTCS.26.16. arXiv:1007.3794.

Ross Duncan. Generalized Proof-Nets for Compact Categories with Biproducts. In Simon Gay and Ian Mackie, editors, *Semantic Techniques in Quantum Computation*, pages 70–134. Cambridge University Press, Cambridge, 2009. ISBN 978-1-139-19331-3. doi:10.1017/CBO9781139193313.004. arXiv:0903.5154.

Charles Ehresmann. Catégories structurées. *Annales scientifiques de l'École Normale Supérieure*, 80(4):349–426, 1963.

Josep Elgueta. The groupoid of finite sets is biinitial in the 2-category of rig categories. *arXiv:2004.08684 [math]*, April 2020. arXiv:2004.08684.

Thomas M Fiore, Simona Paoli, and Dorette Pronk. Model structures on the category of small double categories. *Algebraic & Geometric Topology*, 8(4):1855–1959, October 2008. ISSN 1472-2739, 1472-2747. doi:10.2140/agt.2008.8.1855.

Simon Forest. *Computational Descriptions of Higher Categories*. Theses, Institut Polytechnique de Paris, January 2021.

Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7): 665–667, January 1976. ISSN 0092-7872. doi:10.1080/00927877608822127.

Peter J Freyd and David N Yetter. Braided compact closed categories with applications to low dimensional topology. *Advances in Mathematics*, 77(2):156–182, October 1989. ISSN 00018708. doi:10.1016/0001-8708(89)90018-2.

Tobias Fritz and Paolo Perrone. Bimonoidal Structure of Probability Monads. *Electronic Notes in Theoretical Computer Science*, 341:121–149, December 2018. ISSN 15710661. doi:10.1016/j.entcs.2018.11.007. arXiv:1804.03527.

Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. *arXiv:1603.04641 [cs]*, March 2016. arXiv:1603.04641.

Jose Manuel Gomez. From fibered symmetric bimonoidal categories to symmetric spectra. *arXiv:0905.3156 [math]*, May 2009. arXiv:0905.3156.

Bertrand Guillou. Strictification of categories weakly enriched in symmetric monoidal categories. *arXiv:0909.5270 [math]*, September 2009. arXiv:0909.5270.

Nick Gurski and Eugenia Cheng. The periodic table of n-categories II: Degenerate tricategories. *Cahiers de Topologie et Géométrie Différentielle Catégoriques*, 52(2):45, 2011.

W. Haken. Theorie der Normalflächen. *Acta Math.*, 105:245–375, 1961. doi:10.1007/BF02559591.

N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proceedings of the IEEE*, 79(9):1305–1320, 1991. ISSN 00189219. doi:10.1109/5.97300.

Jules Hedges. Morphisms of Open Games. *Electronic Notes in Theoretical Computer Science*, 341:151–177, December 2018. ISSN 1571-0661. doi:10.1016/j.entcs.2018.11.008.

Lukas Heidemann, Nick Hu, and Jamie Vicary. Homotopy.io. 2019. doi:10.5281/zenodo.2540764.

Chris Heunen and Jamie Vicary. Lectures on categorical quantum mechanics. *Computer Science Department. Oxford University*, 2012.

Thomas T. Hildebrandt, Prakash Panangaden, and Glynn Winskel. A relational model of non-deterministic dataflow. *Mathematical Structures in Computer Science*, 14(5): 613–649, October 2004. ISSN 0960-1295, 1469-8072. doi:10.1017/S0960129504004293.

J. E. Hopcroft and J. K. Wong. Linear Time Algorithm for Isomorphism of Planar Graphs (Preliminary Report). In *Proceedings of the Sixth Annual ACM Symposium on Theory of Computing*, STOC '74, pages 172–184, New York, NY, USA, 1974. ACM. doi:10.1145/800119.803896.

Guenter Hotz. Eine Algebraisierung des Syntheseproblems von Schaltkreisen I. *Elektronische Informationsverarbeitung und Kybernetik*, 1(3):185–205, 1965.

Benjamin Taylor Hummon. *Surface Diagrams for Gray-Categories*. PhD thesis, UC San Diego, 2012.

David Huynh, Tom Morris, Stefano Mazzocchi, Iain Sproat, Martin Magdinier, Thad Guidry, Jesus M. Castagnetto, James Home, Cora Johnson-Roberson, Will Moffat, Pablo Moyano, David Leoni, Peilonghui, Rudy Alvarez, Vishal Talwar, Scott Wiedemann, Mateja Verlic, Antonin Delpeuch, Shixiong Zhu, Charles Pritchard, Ankit Sardesai, Gideon Thomas, Daniel Berthereau, and Andreas Kohn. OpenRefine. 2019. doi:10.5281/zenodo.595996.

Roshan P. James and Amr Sabry. Information effects. *ACM SIGPLAN Notices*, 47(1): 73–84, January 2012. ISSN 0362-1340. doi:10.1145/2103621.2103667.

Yeonhee Jang, Tsuyoshi Kobayashi, Makoto Ozawa, and Kazuto Takao. Stabilization of bridge decompositions of knots and bridge positions of knot types (The theory of transformation groups and its applications). *RIMS Kokyuroku*, 2135:23–28, 2019.

Gareth A. Jones and David Singerman. Theory of Maps on Orientable Surfaces. *Proceedings of the London Mathematical Society*, s3-37(2):273–307, September 1978. ISSN 00246115. doi:10.1112/plms/s3-37.2.273.

André Joyal and Ross Street. Braided monoidal categories. *Mathematics Reports*, 86008, 1986.

André Joyal and Ross Street. Planar diagrams and tensor algebra. September 1988.

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

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

André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, April 1996. ISSN 0305-0041, 1469-8064. doi:10.1017/S0305004100074338.

Gilles Kahn. The semantics of a simple language for parallel programming. *Information processing*, 74:471–475, 1974.

K. M. Kavi, B. P. Buckles, and U. N. Bhat. Isomorphisms Between Petri Nets and Dataflow Graphs. *IEEE Transactions on Software Engineering*, SE-13(10):1127–1134, October 1987. ISSN 0098-5589. doi:10.1109/TSE.1987.232854.

G. M. Kelly. On MacLane's conditions for coherence of natural associativities, commutativities, etc. 1964. doi:10.1016/0021-8693(64)90018-3.

Marc Lackenby. A polynomial upper bound on Reidemeister moves. *Annals of Mathematics*, pages 491–564, 2015. doi:10.4007/annals.2015.182.2.3.

Marc Lackenby. The efficient certification of knottedness and Thurston norm. *arXiv:1604.00290 [math]*, July 2019. arXiv:1604.00290.

Miguel L. Laplaza. Coherence for distributivity. In G. M. Kelly, M. Laplaza, G. Lewis, and Saunders Mac Lane, editors, *Coherence in Categories*, volume 281, pages 29–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 1972. ISBN 978-3-540-05963-9 978-3-540-37958-4. doi:10.1007/BFb0059555.

Saunders Mac Lane. Natural associativity and commutativity. *Rice Univ. Studies*, 49(4): 28–46, 1963.

M. Makkai. The word problem for computads. May 2005.

A. Markov. The insolubility of the problem of homeomorphy. *Doklady Akademii Nauk SSSR*, 121:218–220, 1958. ISSN 0002-3264.

J Peter May. *The Geometry of Iterated Loop Spaces*, volume 271. Springer, 1972. ISBN 978-3-540-37603-3.

José Meseguer, Ugo Montanari, and Vladimiro Sassone. On the semantics of Petri Nets. In W.R. Cleaveland, editor, *CONCUR '92*, volume 630, pages 286–301. Springer Berlin Heidelberg, 1992. ISBN 978-3-540-55822-4. doi:10.1007/BFb0084798.

Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*, volume 10. JHU Press, 2001. ISBN 0-8018-6689-8.

David Jaz Myers. String Diagrams For Double Categories and Equipments. *arXiv:1612.02762 [math]*, December 2016. arXiv:1612.02762.

Jean-Pierre Otal. Présentations en ponts du nœud trivial. *C. R. Acad. Sci., Paris, Sér. I*, 294:553–556, 1982. ISSN 0764-4442.

Carl Adam Petri. Communication with automata. page 97, 1966.

Emil L. Post. Recursive Unsolvability of a problem of Thue. *The Journal of Symbolic Logic*, 12(1):1–11, March 1947. ISSN 0022-4812, 1943-5886. doi:10.2307/2267170.

Vaughn Pratt. Modeling concurrency with geometry. In *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '91*, pages 311–322, Orlando, Florida, United States, 1991. ACM Press. ISBN 978-0-89791-419-2. doi:10.1145/99583.99625.

David J. Reutter and Jamie Vicary. Biunitary constructions in quantum information. *Higher Structures*, 3(1):109–154, 2019. ISSN 2209-0606. arXiv:1609.07775.

P. Selinger. A Survey of Graphical Languages for Monoidal Categories. In Bob Coecke, editor, *New Structures for Physics*, number 813 in Lecture Notes in Physics, pages 289–355. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12820-2 978-3-642-12821-9. doi:10.1007/978-3-642-12821-9_4.

Mei Chee Shum. Tortile tensor categories. *Journal of Pure and Applied Algebra*, 93(1): 57–110, April 1994. ISSN 0022-4049. doi:10.1016/0022-4049(92)00039-T.

Eugene W. Stark. Dataflow networks are fibrations. In David H. Pitt, Pierre-Louis Curien, Samson Abramsky, Andrew M. Pitts, Axel Poigné, and David E. Rydeheard, editors, *Category Theory and Computer Science*, Lecture Notes in Computer Science, pages 261–281. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-38413-7. doi:10.1007/BFb0013470.

Sam Staton. Algebraic Effects, Linearity, and Quantum Programming Languages. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages - POPL '15*, pages 395–406, Mumbai, India, 2015. ACM Press. ISBN 978-1-4503-3300-9. doi:10.1145/2676726.2676999.

ME Szabo. Polycategories. *Communications in Algebra*, 3(8):663–689, 1975. doi:10.1080/00927877508822067.

Bruce Trace. On the Reidemeister moves of a classical knot. *Proceedings of the American Mathematical Society*, 89(4):722–724, 1983. ISSN 0002-9939, 1088-6826. doi:10.1090/S0002-9939-1983-0719004-4.

Edward Walker, Weijia Xu, and Vinoth Chandar. Composing and executing parallel data-flow graphs with shell pipes. In *Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science - WORKS '09*, pages 1–10, Portland, Oregon, 2009. ACM Press. ISBN 978-1-60558-717-2. doi:10.1145/1645164.1645175.