String Diagrams for Closed Symmetric Monoidal Categories

Callum Reader
Tallinn University of Technology, Estonia

Alessandro Di Giorgio
Tallinn University of Technology, Estonia

Abstract

We introduce a graphical language for closed symmetric monoidal categories based on an extension of string diagrams with special bracket wires representing internal homs. These bracket wires make the structure of the internal hom functor explicit, allowing standard morphism wires to interact with them through a well-defined set of graphical rules. We establish the soundness and completeness of the diagrammatic calculus, and illustrate its expressiveness through examples drawn from category theory, logic and programming language semantics.

2012 ACM Subject Classification Theory of computation → Logic; Theory of computation → Categorical semantics

Keywords and phrases diagrammatic languages, logic, lambda calculi

Digital Object Identifier 10.4230/LIPIcs.CSL.2026.12

Related Version *Full Version*: https://arxiv.org/abs/2512.06499 [46]

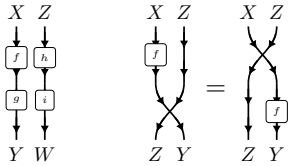
Funding *Callum Reader*: Supported by the Estonian Research Council under Grant PSG764.
Alessandro Di Giorgio: Supported by the European Union under Grant No. 101087529.

Acknowledgements We would like to acknowledge Matt Earnshaw, Alkis Ioannidis, Mario Roman, Pawel Sobocinski and Simon Willerton for several helpful discussions. We are also grateful to the anonymous reviewers of CSL '26 for their insightful comments and feedback.

1 Introduction

Diagrams have long played a central role in mathematics and computer science, serving both as intuitive aids and as rigorous tools for reasoning. From commutative diagrams in algebraic topology [19] to Penrose notation in physics [40], Peirce's existential graphs in mathematical logic [47], flowcharts in programming [39] and Petri nets in concurrency [41], the use of pictures to represent abstract structures predates the formalisation of many of the theories they illustrate. In particular, diagrammatic reasoning has been crucial in areas where morphisms, transformations, or processes are primary, allowing mathematicians and computer scientists to track complex compositions and dependencies visually.

String diagrams emerged from this tradition as a powerful formalism for reasoning in symmetric monoidal categories. Introduced in the context of category theory by Joyal and Street [31], string diagrams provide a graphical language where morphisms are depicted as boxes and wires, and categorical structures – such as tensor product, composition, and symmetry – are encoded geometrically.

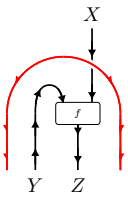


This approach makes algebraic reasoning not only easier to manage but often self-evident. For instance, the equation $(f;g) \otimes (h;i) = (f \otimes h);(g \otimes i)$, expressing functoriality of the tensor product, is represented by a single string diagram – both sides correspond to exactly the same picture, shown in the diagram on the left above. Other equations, such as the naturality of the symmetry, are captured by geometric transformations: for example, sliding a box past a crossing of wires, as illustrated by the pair of diagrams on the right.

As such, string diagrams have found applications across quantum computing [1, 16], database theory [12], programming language semantics [30, 8], control theory [13], logic [9] and more [24, 42, 7, 28].

With the traction gained by the growing number of applications, string diagrams were naturally extended to accommodate richer categorical settings. These extensions reflect the need to reason diagrammatically in settings where additional layers of structure – such as duals, traces, or multiple monoidal products – are present. For example, compact closed categories [23] enrich string diagrams with cups and caps to make duality explict as a bending of wires. Traced monoidal categories [32] introduce loops to represent fixed-point behaviour or feedback, crucial in the semantics of recursion and iteration. Similarly, diagrams [17, 10] for bimonoidal categories [34] have to deal with multiple tensor products, requiring new graphical conventions to keep the interactions between monoidal structures clear.

In this paper, we develop a string diagrammatic language for *closed* symmetric monoidal categories – a class of structured categories that plays a foundational role in logic and computer science. These categories are distinguished by the presence of an internal hom functor, allowing e.g. to represent morphisms from $X \otimes Y$ to Z as morphisms from X to $[Y, Z]$. In Set, the category of sets and functions, this operation corresponds to currying, with $[Y, Z]$ being the set of functions from Y to Z. In our language currying is represented as the diagram below.



In particular we employ red "bracket" strings to form the internal hom $[Y, Z]$; the direction on the wires indicate whether the objects are covariant or contravariant in the internal hom functor.

This structure captures the essence of higher-order computational models, underlying the categorical semantics of typed λ-calculi. In proof theory, the internal hom corresponds to implication, and its adjunction with the tensor product models the process of introducing and eliminating assumptions.

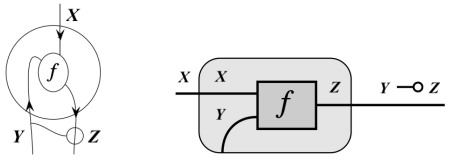
Despite the centrality of these categories, their string diagrammatic treatment has historically been more limited, due to the added complexity of representing internal homs in a geometric way. The aim of this paper is to fill this gap by introducing a formal diagrammatic language that is enough to support a wide range of applications, from logic to programming language semantics, as already suggested in this introduction.

Synopsis. In Section 2, we present our diagrammatic language via generators and equations, and we prove that it forms a closed symmetric monoidal category. Furthermore, we prove some derived equations that reinforce the visual intuition. In Section 3, we prove completeness

– showing that the string diagrams are the free closed symmetric monoidal category generated by a closed monoidal signature. To this end, we establish a normalisation and a decomposition result for our diagrams. In Appendix A, we provide several examples from category theory and computer science, including a diagrammatic encoding of the simply typed λ-calculus.

Related Work

There have been some attempts to develop a diagrammatic syntax for closed monoidal categories, most notably the *bubble-and-clasp* notation in [4] and the *hierarchical string diagrams* in [3, 27]. Higher-order mechanisms are represented in both of them as a bubble enclosing a diagram. For example, currying is depicted as the diagrams below.



They are close in spirit to our language, however there are some notable differences. The first diagram binds together the strings for Y and Z via a clasp to form the object $[Y, Z]$, while in the second diagram, the Y wire is curryed away and the output type is annotated with $Y \multimap Z$, denoting the object $[Y, Z]$.

For the case of [4], we are not aware of any completeness result with respect to closed symmetric monoidal categories. The language proposed in [3, 27] has been formalised using functorial boxes [37]. However, the approach is not fully diagrammatic, as it relies on a hybrid syntax combining graphical elements with algebraic notation, rather than a purely visual representation. The same holds true also for the diagrams for monoidal closed theories appearing in [26, 25].

In [49], it is shown that every closed monoidal category fully faithfully embeds into a $*$-autonomous category via a functor that preserves the closed structure. This result suggests that graphical calculi for $*$-autonomous categories [6, 15, 18] can, in principle, be used to reason about closed monoidal categories as well. However, such graphical languages are closer to proof-nets rather than string diagrams.

Recent, unpublished work by Willerton [51] provides a more geometric account of non-symmetric, closed monoidal categories. In this language, the internal hom is represented by a surface which surrounds the strings, acting as a boundary.

Finally, although they arise in different contexts, additional instances of string diagrams equipped with graphical mechanisms for managing different kinds of bracketing can be found in [2, 10, 9, 17, 50].

2 String Diagrams with Brackets

The string diagram language that we present here requires that certain manipulations of strings can only take place in certain "contexts" so to speak. These contexts represent the closed structure, typically denoted $[A, B]$ or $A \multimap B$. Other similar languages also have such contexts, in [27] this means using a functor box, while in [4] this means using a bubble.

The key difference here is that our contexts are not machinery that we impose on string diagrams. They are built from special types of strings – referred to as "bracketing strings" – and these bracketing strings are subject to manipulations and equations that we might

expect from any other string. In fact, the equations on bracketing strings correspond closely to symmetries as well as yankings, poppings and mergings – the same equations imposed on string diagrams for adjoint equivalences.

To see how we use these bracketing strings, consider our simplest example: diagram (a) in Figure 2, which represents the identity map on $[A, B]$. Here we read our diagrams from top to bottom. The upwards string – or "upstring" – indicates that A is contravariant in the internal hom functor, whilst the "downstring" B is covariant. Note that an upstring may only appear between a bracketing pair of strings, and that there is a left bracketing string, and a right bracketing string.

The reader might also notice that our bracketing strings are directional. This is so that we can nest our brackets and represent double dualisations. So diagram (b) in Figure 2 represents the identity on $[[A, B], C]$.

As in string diagrams for monoidal categories, we represent the tensor product by horizontal concatenation, so diagram (c) in Figure 2 represents the identity on $A \otimes [B \otimes C, D \otimes E]$. In this example note that all of the upstrings are on the left and all of the downstrings are on the right. This is a sort of "normal form" for our diagrams that we discuss in Subsection 3.1. Since our language is for closed *symmetric* monoidal categories, it is convenient to allow our upstrings and downstrings to appear in any order, and we add in a syntax for crossing these strings, subject to the usual symmetry equations[1]. For example, the identity on $A \otimes [B \otimes C, D \otimes E]$ can also be represented by diagram (d) in Figure 2.

Of course we include the usual boxes to represent morphisms. So, for example, if we have $g \colon A_0 \to A_1$ and $f \colon [A_0, B] \to C$ then the composite $[A_1, B] \xrightarrow{[g,B]} [A_0, B] \xrightarrow{f} C$ is represented by diagram (g) of Figure 2.

Boxes allow us to represent certain morphisms between internal homs. In order to represent all of the *canonical* morphisms found in a closed symmetric monoidal category we have a collection of manipulations that can be performed on diagrams. Locally, these manipulations look like the those of Figure 1. However, to prevent the generation of invalid diagrams, these manipulations are – informally – subject to the following restrictions.

1. Bracketing strings must occur in balanced pairs.
2. Upstrings can only occur between pairs of bracketing strings.
3. Downstrings can only leave a downward bracketing pairs of strings, if that bracketing pair of strings contains no upstrings.
4. Upstrings can only enter a downwards bracketing pair from another, if the outer bracketing pair has no other downstrings.
5. The corresponding restrictions on upwards bracketing pairs of strings.

For example, we can represent the canonical morphism $[A, B] \otimes C \to [A, B \otimes C]$ by diagram (f) in Figure 2. In, for example, Set this takes the pair (f, c) to the function f_c where $f_c(a) = (f(a), c)$.

In diagram (e) of Figure 2 we see the internal tensor-hom adjunction $[A, [B, C]] \to [A \otimes B, C]$. This is a use of syntactic sugar, where the crossing of the cupped bracket string represents two separate crossings, followed by a bracket cup. This syntactic sugar is unambiguous, since we later derive the equations in Figure 7c.

[1] Note that, this manipulation can be considered purely syntactically, but might also be seen as a kind of partially unbiased definition of closed symmetric monoidal in the vein of [35]. It also corresponds to the fact that a closed symmetric monoidal category is both left- and right-closed, but we do not explore that perspective here.

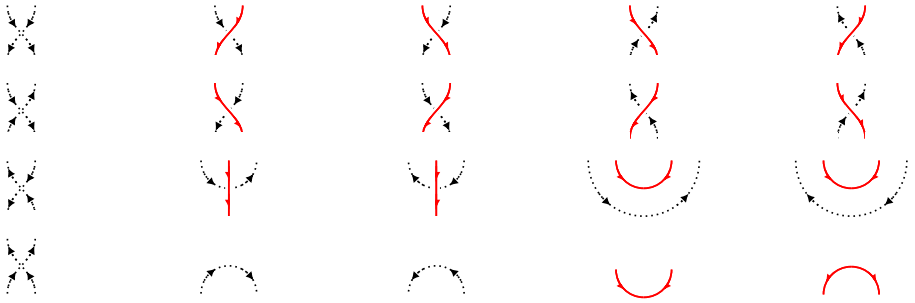


Figure 1 Local manipulations.

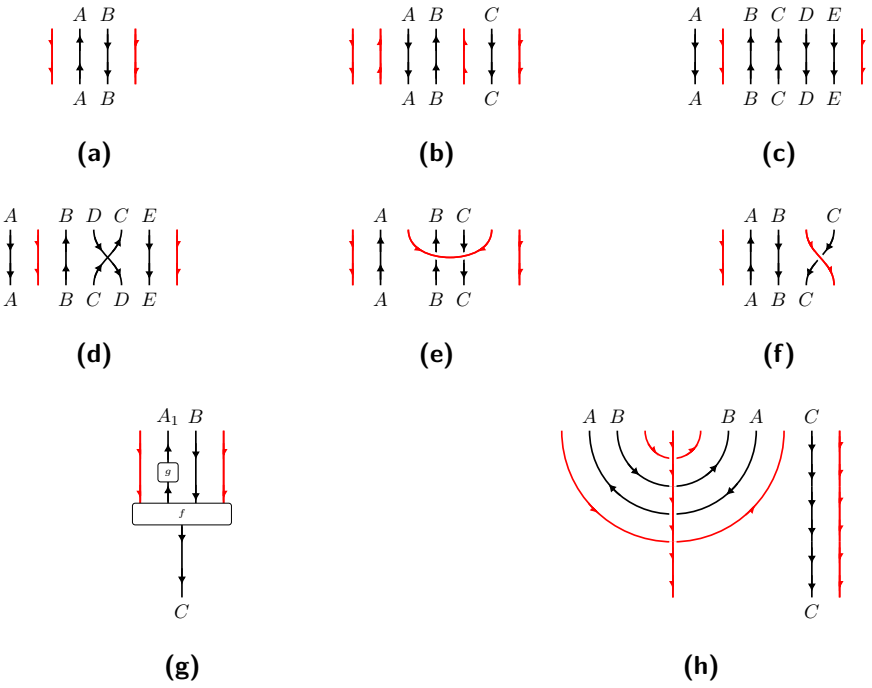


Figure 2 Examples of valid diagrams.

Figure 3 Examples of invalid diagrams.

Note that in diagrams (e) and (h) we cross an upstring with a bracketing string. We cannot, however, draw diagram (a) in Figure 3. This is because the presence of the downstring C *prevents* A from crossing the bracketing string. This should certainly be the case, since there is no canonical morphism $[A, [I, B] \otimes C] \to [I, [I, A \otimes B] \otimes C]$.

It is also worth noting that the cups in our language must always either cross a bracketing string or follow a bracketing string, and this prevents the occurrence of traces. For example, diagram (b) of Figure 3 is not a valid diagram.

Finally, note that the language is designed in such a way that the contents of bracketing pairs of strings can really be thought of as a string in itself – this is why we use half arrowheads to denote the left and right bracketing strings. So, for example, diagram (h) in Figure 2 represents the canonical evaluation $[A, B] \otimes [[A, B], C] \to C \xrightarrow{\sim} [I, C]$.

As is typical for string diagrams, we are interested in generating the language starting from a signature, that is a set of symbols $f \colon X \to Y$, each with their own name and type.

▶ **Definition 1.** *A* (unbiased) closed monoidal signature*, often simply denoted by* Σ*, is a quadruple* $(\mathcal{S}, \Sigma, ar, coar)$ *consisting of a set* $\mathcal{S}$ *of* sorts*, a set* Σ *of* generators*, and two functions* $ar, coar \colon \Sigma \to \mathcal{S}^\sharp$ *that assign to each generator its domain and codomain, respectively. Here,* $\mathcal{S}^\sharp$ *is the set of strings generated by the first layer of the grammar below*

$$\begin{array}{lcllll} X & \coloneqq & I & | & AX & | & [\mathbf{X}]X \\ X^* & \coloneqq & I & | & A^*X^* & | & [\mathbf{X}]^*X^* \\ \mathbf{X} & \coloneqq & I & | & X^*\mathbf{X} & | & X\mathbf{X} \end{array}$$

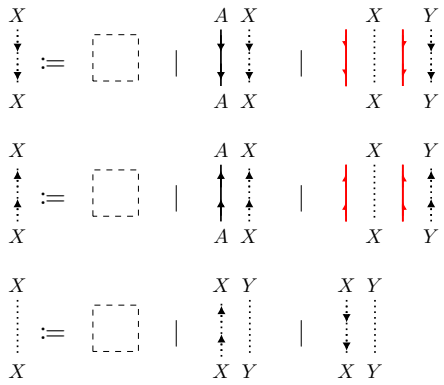
where I *is the empty string and* A *is a symbol in* $\mathcal{S}$*. Elements of* $\mathcal{S}^\sharp$ *are thus bracketed strings of sorts and their formal dual. Brackets* $[-]$ *and duals* $-^*$ *serve the same role of the brackets and upstrings in the diagrammatic language.*

As a convention, we use letters $A, B, C, \ldots$ *to denote elements of* $\mathcal{S}$*, and* $X, Y, Z, \ldots$ *and* $U, V, W, \ldots$ *to denote elements of* $\mathcal{S}^\sharp$*.*

Since our string types involve nested brackets, as well as upstrings and downstrings, in the first subsection we give an inductive definition of how we represent each of these types. In the second subsection we give an inductive definition of our terms. Rather than generating our diagrams from the manipulations in Figure 1, and discarding the invalid manipulations, we instead introduce a syntax that can *only* generate valid diagrams. In the third subsection we give a collection of equations that we impose on these diagrams such as symmetry, naturality, and yanking conditions. We then prove that additional, expected equations can be derived from these, before proving that the category they generate is, in fact, closed symmetric monoidal.

2.1 Objects

Since we now have "types" of string – upstrings, downstrings, upwards brackets, and downwards brackets – we must first generate all of the identity diagrams in a sensible way. We need to make sure that our bracketing strings are balanced, and that we can generate nested brackets, as well as strings in up- and downward directions. This is done via the following three layer grammar, that is a diagrammatic version of the one in Definition 1.



We use solid lines for the generating objects $A \in \mathcal{S}$ and dotted lines to indicate any object. Note that here undirected strings – those strings that appear inside bracketing strings – are really lists of up- and downstrings that can appear in any order. This does not represent the typical algebraic representation of a closed symmetric monoidal category, in which all of the contravariant terms appear on the left and all of the covariant terms appear on the right: for example, $[A \otimes B, C \otimes D \otimes E]$. In Definition 2 we define normalised terms and in Theorem 12 we prove that the existence of unordered terms is purely syntactic.

2.2 Morphisms

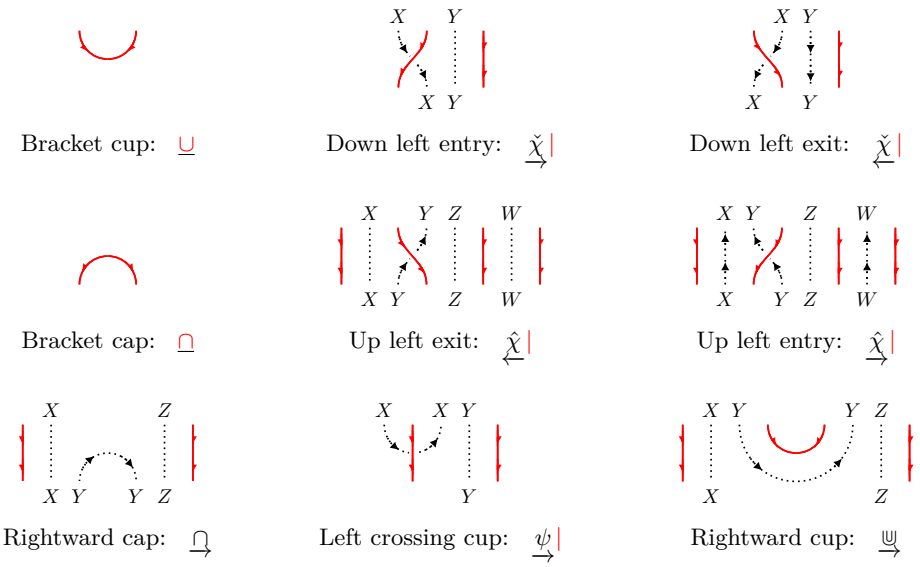


Figure 4 Generators for string diagrams with brackets.

At the beginning of the section we gave examples of our string diagrams, and the manipulations and restrictions that allow us to reason about closed symmetric monoidal categories. Rather than defining these string diagrams in terms of local manipulations, and then discarding invalid diagrams, we choose to restrict to generators that can *only* generate valid diagrams.

These are precisely those given in Figure 4, as well as their reflections in the vertical axis and the respective contravariant generators. In Lemma 14 we give an account of how – using the equations we impose in Figure 6 – these generators can be decomposed into the usual evaluation and coevaluation maps of a closed symmetric monoidal category. For the sake of intuition, here we give an account of what each of these diagrams represents in Set.

$\cup$, $\cap$ The bracket cup and cap are the isomorphisms $[1,1] \xrightarrow{\sim} 1$ and $1 \xrightarrow{\sim} [1,1]$. They simply encode the unique isomorphism $\{*\} \cong \{f \colon * \to *\}$.

$\check{\chi}_{\rightarrow}$ The down left entry represents a map $A \times [B,C] \to [B, A \times C]$. This takes a pair $(a, f \colon B \to C)$ and maps it to a function $f_a(b) = (a, f(b))$.

$\check{\chi}_{\leftarrow}$ The down left exit represents a map $[1, A \times B] \to A \times [1, B]$, where each f with $f(*) = (x, y)$ gets mapped to (x, g) with $g(*) = y$.

$\hat{\chi}_{\leftarrow}$ The up left exit allows us to represent a generalised form of internal uncurrying. This diagram represents a map $[A, [B,C] \times D] \to [A \times B, [1, C] \times D]$. This takes a function $f(a) = (f_1(a), f_2(a))$ and maps it to the function $g(a,b) = (* \mapsto f_1(a)(b), f_2(a))$.

$\hat{\chi}_{\rightarrow}$ The up left entry allows us to represent *internal* currying. This diagram represents a map $[A \times B, [C, D]] \to [A, [B \times C, D]]$. This takes a function $f(a,b) = g$ and maps it to the function $h(a)(b,c) = f(a,b)(c)$.

$\cap_{\rightarrow}$ The rightward cap allows the introduction of internal identities. This diagram represents a map $[A, C] \to [A \times B, B \times C]$. This takes a function $f(a)$ and maps it to the function $g(a,b) = (b, f(a))$.

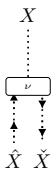
$\psi_{\rightarrow}$ The left crossing cup allows us to partially evaluate functions internally. This diagram represents a map $A \times [A \times B, C] \to [B, C]$. This takes a pair $(a, f \colon A \times B \to C)$ and maps it to the function $f(a, -) \colon B \to C$.

$\cup_{\rightarrow}$ The rightward cup allows us to partially compose functions. This diagram represents a map $[A, B \times C] \times [C \times D, E] \to [A \times D, B \times E]$. This map takes a pair of functions $((f \colon A \to B \times C), (g \colon C \times D, E))$ and maps them to the function $h(a,d) = (f_1(a), g(f_2(a), d))$.

The full diagrammatic syntax is given by the three-layer grammar in Figure 5. Note that, as for strings, the contravariant layer is a reflection of the covariant one. Moreover, note that the mixed terms in Figure 5 also include crossings between wires of different directions. These crossings are particularly useful for defining a normalisation procedure that sorts the components of undirected strings.

For the sake of readability we include syntactic sugar for the bracket cup and cap diagrams. In order to do this we first need a notion of normal form.

▶ **Definition 2.** *An undirected string is called* normalised*, or* in normal form*, if it is generated in such a way that all upstrings appear on the left, and all undirected strings that occur within brackets are also normalised. There is also a normaliser term, ν, pictured below. This is generated inductively from the crossing of upstrings and downstrings, as seen in the mixed layer of the grammar in Figure 5 (see [46, Definition 2.3]). Note that the output of the normaliser term is always in normal form.*



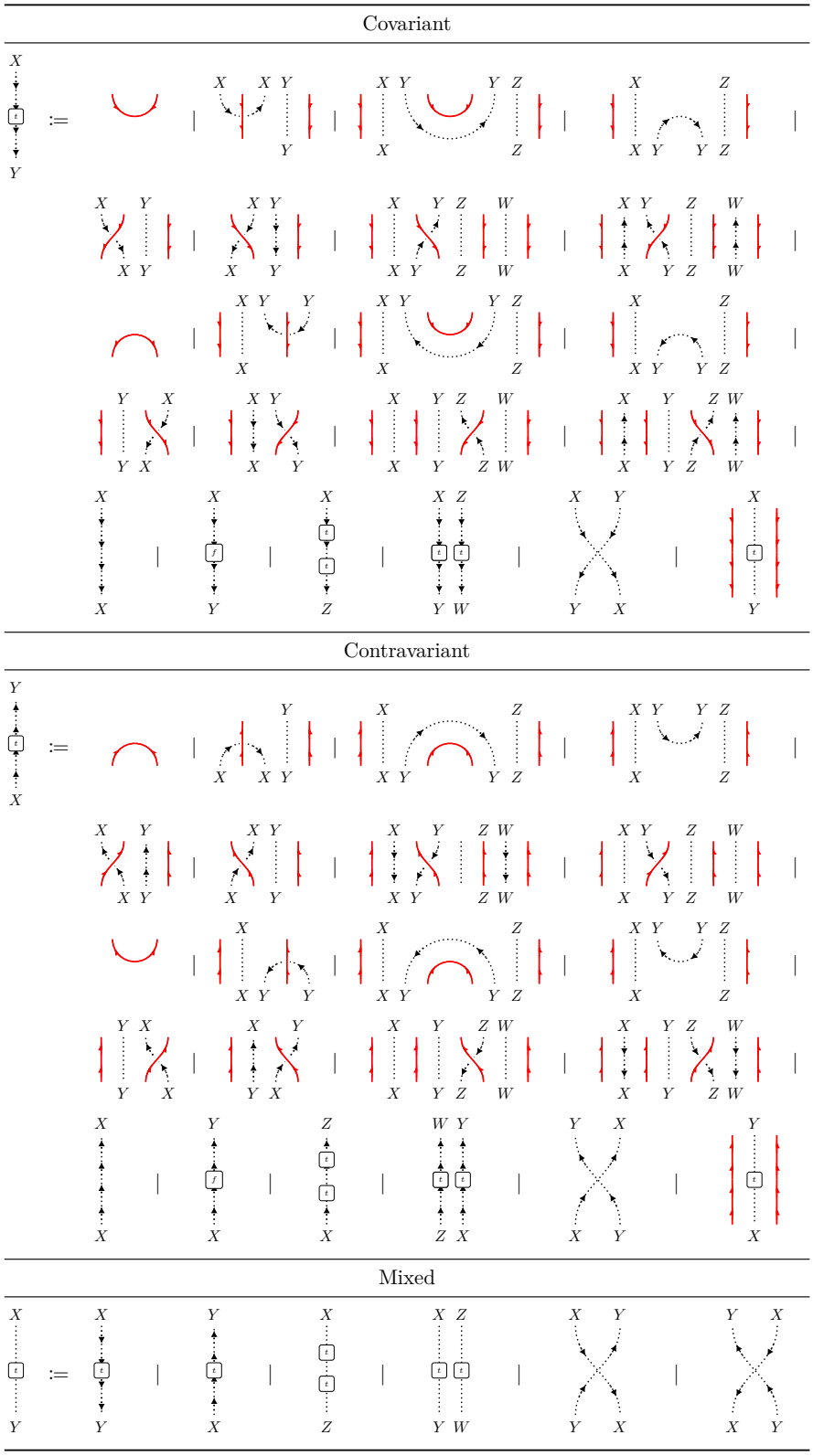
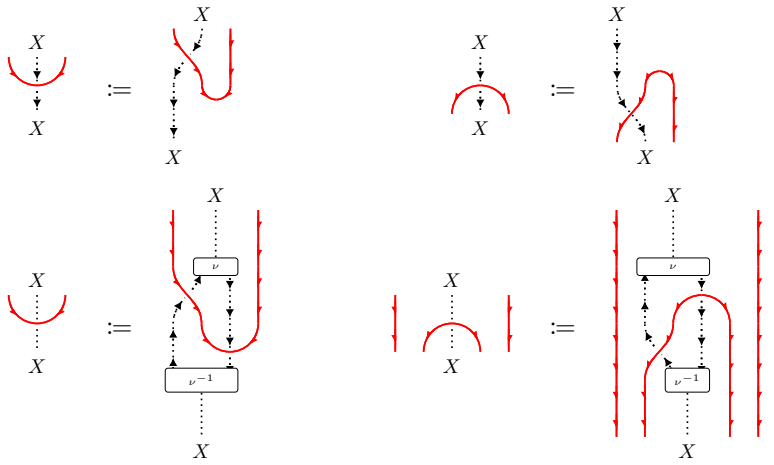


Figure 5 Full syntax of string diagrams with brackets.

▶ **Definition 3.** *For the sake of keeping our language geometrically intuitive, we introduce some syntactic sugar for strings that leave and enter cups and caps, respectively.*



Where here ν^{-1} is the reflection of ν in the horizontal axis. Note that by the equations that we present in Section 2.3, we can show that ν^{-1} is the inverse of ν.

Note also that by the additional naturality and symmetry equations derived in Figures 7a and 7c we can show that leaving the bracketing cup on the left is equal to leaving the bracketing cup on the right, and similarly for the bracketing cap. In other words, the syntactic sugar given above is unambiguous.

2.3 Equations

As highlighted above, reasoning in this diagrammatic language is context dependent: the manipulations that can be performed on a particular string may depend on the contents certain pairs of bracketing strings. However, for the sake of readability here we present the equations "locally". By this we mean that, if a certain local sequence *can* occur validly then we can replace it locally. Let us consider the following composite, built from the internal tensor-hom adjunction and written in the usual algebraic language.

$$[A \otimes B, [I, C]] \xrightarrow{\sim} [A, [B, C]] \xrightarrow{\sim} [A \otimes B, [I, C]]$$

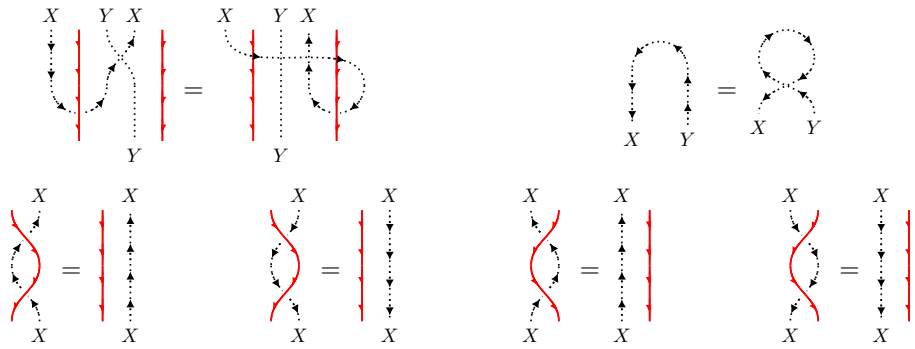
This composite is equal to the identity, and this holds regardless of what A, B and C are. We can really just see this as local rewrite of the substrings highlighted in black.

We understand that this composite of rewrites should be the identity, even though the substrings themselves aren't valid terms. The same is true of the manipulations in our string diagrams, which we treat as rewrites via a congruence. For the sake of brevity, we assume that for each given equation, the reflections in the horizontal and vertical axes also hold.

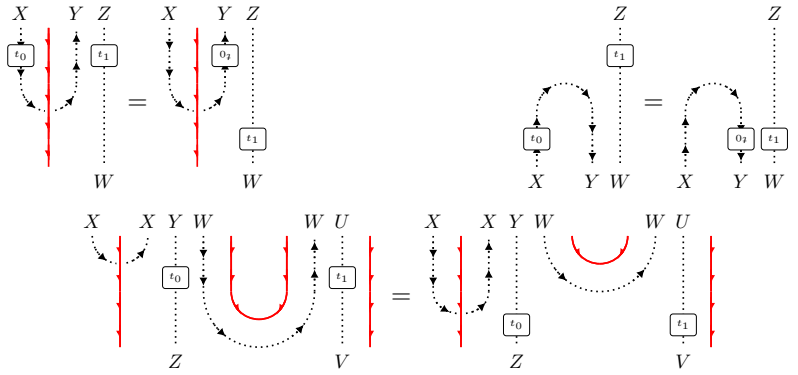
▶ **Definition 4.** *Given a (unbiased) closed monoidal signature $(\mathcal{S}, \Sigma)$, we denote as $\mathscr{C}_\Sigma$ the category whose objects are elements of $\mathcal{S}^\sharp$ and morphisms are* covariant terms *quotiented by the minimal congruence $\cong$ generated by the equations of SMCs and those in Figure 6.*

▶ Remark 5. For the details of how $\mathscr{C}_\Sigma$ and $\cong$ are constructed, see [46, Appendix A].

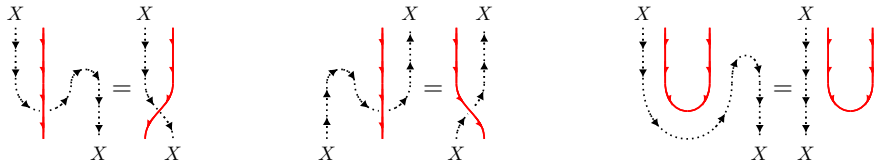
From the congruence $\cong$ we can derive a number of other equations. We do so for two reasons: firstly, to show that our geometric intuition is correct, and that almost all manipulations that we expect to be possible, are in fact possible; secondly, to provide the basis for the proofs in Section 3 where we show that our language is, in fact, the free closed symmetric monoidal category on a given closed monoidal signature.



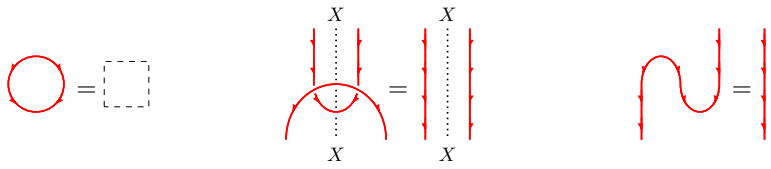
(a) Crossing equations.



(b) Naturality equations.

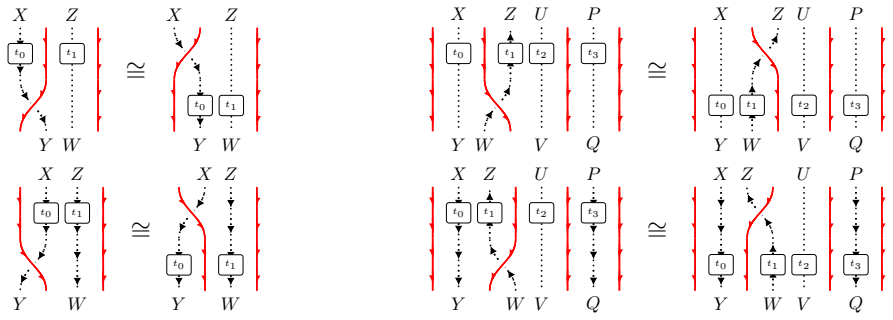


(c) Yanking equations.

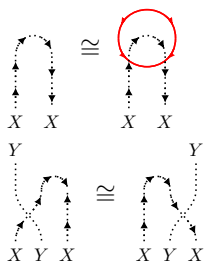


(d) Bracket pop, merge and yanking equations.

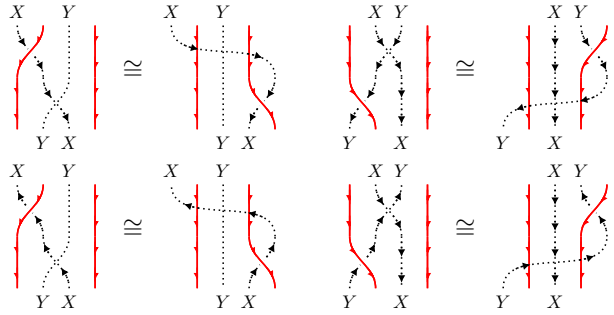
Figure 6 Equations for string diagrams with brackets.



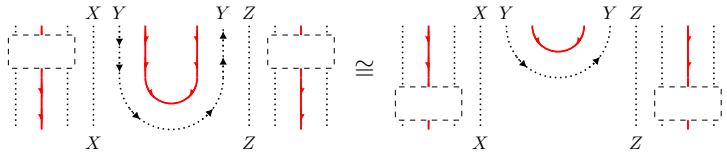
(a) Derived naturality equations.



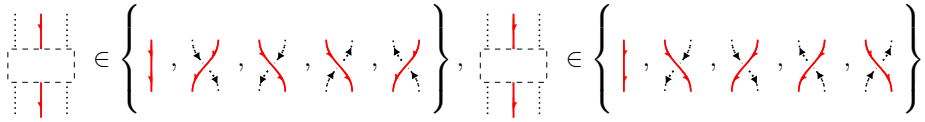
(b) Derived cap equations.



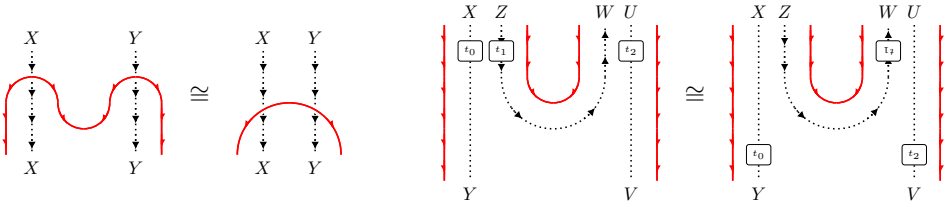
(c) Derived crossing equations.



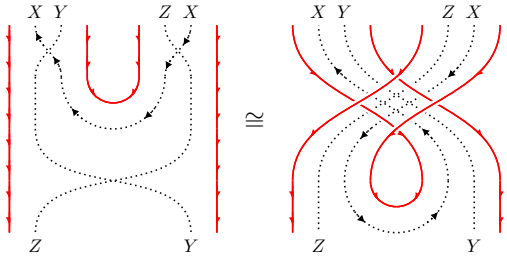
for all



(d) An interchange law for cups and crossings.



(e) Smoothing and naturality laws for the rightward cup.



(f) A symmetry law for the rightward cup.

Figure 7 Derived equations for string diagrams with brackets.

▶ **Theorem 6.** *The equations in Figure 7 hold.*

Proof. See [46, Theorem 2.6]. ◀

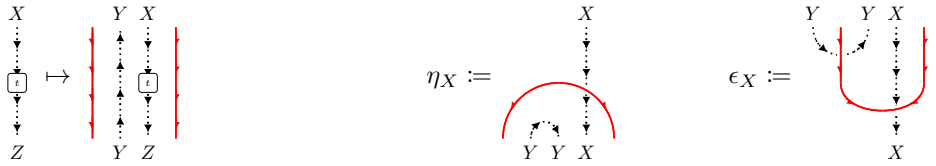
After presenting the syntax and equations of our diagrammatic calculus, we are now ready to prove that $\mathscr{C}_\Sigma$ is, in fact, a closed symmetric monoidal category.

▶ **Theorem 7.** *$\mathscr{C}_\Sigma$ is a closed symmetric monoidal category.*

Proof. First, observe that $\mathscr{C}_\Sigma$ is a symmetric monoidal category. The tensor product $\otimes$ is given by parallel composition of diagrams and symmetries by crossing of wires. Moreover, by definition of $\mathscr{C}_\Sigma$, diagrams are subject to the laws of SMCs.

To show that it is also closed, we define the right adjoint functor $[Y, -] \colon \mathscr{C}_\Sigma \to \mathscr{C}_\Sigma$ to the tensor product functor $Y \otimes - \colon \mathscr{C}_\Sigma \to \mathscr{C}_\Sigma$, as follows.

On objects, the functor maps X to $[Y^*X]$, the mapping on morphisms is given on the left below. To show that $Y \otimes - \dashv [Y, -]$, we define the unit η_X of the adjunction, or *coevaluation*, and the counit ϵ_X of the adjunction, or the *evaluation*, in the diagrams below on the right.



By the equations in Figure 7a we can slide diagrams across brackets, and thus naturalities of η_X and ϵ_X are easily verified.

The triangular identities arise as special cases of the currying-uncurrying bijection in Example 17, in the appendix, instantiated with $f, g = id_{X \otimes Y}$. ◀

3 Soundness and Completeness

In this section we prove the completeness result, namely that the category $\mathscr{C}_\Sigma$ of string diagrams with brackets is *the* free closed symmetric monoidal category generated by a closed monoidal signature. We begin by proving that the category of string diagrams is equivalent to one where diagrams are normalised in a suitable sense. We then prove that each of our generators can be decomposed into evaluation and coevaluation maps, before using this result to inductively define an interpretation functor.

3.1 Normalisation

As previously mentioned, the closed symmetric monoidal category generated by a signature, defined above, is in some sense, unbiased. This means that we will show it to be free in a 2-categorical sense – in other words we will show that it is free up to equivalence. In this subsection we define the *normalised* subcategory of the closed symmetric monoidal category on a signature – consisting only of normalised objects and terms. We prove that it is closed symmetric monoidal equivalent to the closed symmetric monoidal category on a signature. This ultimately means that, in the proof of completeness and soundness, we can ignore a lot of the purely syntactic generators, such as the crossings of upstrings with downstrings. We begin by proving that, although we include the reflections of the generating diagrams in Figure 4, this is purely for syntactical convenience.

▶ **Definition 8.** *The* unreflected *closed symmetric monoidal category $\mathscr{C}_\Sigma^{\rightarrow}$ is defined as the subcategory of $\mathscr{C}_\Sigma$ with the same objects but whose morphisms are generated:*

- *on downstrings by Figure 4, without reflection in the vertical axis;*
- *on upstrings by the generators of Figure 4 after a reflection in the horizontal axis.*

To see that this is closed symmetric monoidal note that it is closed under tensors, internal hom, and contains the evaluation and coevaluation diagrams.

▶ **Lemma 9.** *$\mathscr{C}_\Sigma$ is equal to $\mathscr{C}_\Sigma^{\rightarrow}$.*

Proof. By definition the two categories have the same objects, composition and closed symmetric monoidal structures. To see that they have the same morphisms, note that by the crossing equations in Figure 7c and Figure 7f, we know that every generator that is a reflection of those in Figure 4 can be written as a composition of symmetries and its corresponding generator in Figure 4. ◀

▶ **Definition 10.** *The* normalised *closed symmetric monoidal category, $\uparrow\mathscr{C}_\Sigma\downarrow$, on a signature Σ, is defined as the subcategory of $\mathscr{C}_\Sigma^{\rightarrow}$ whose objects are exactly the normalised ones, and whose morphisms are those generated by the diagrams of Figure 12, and, on upstrings, their reflections in the horizontal axis.*

To see that this is closed symmetric monoidal note that it is closed under tensors, internal hom, and contains the evaluation and coevaluation diagrams.

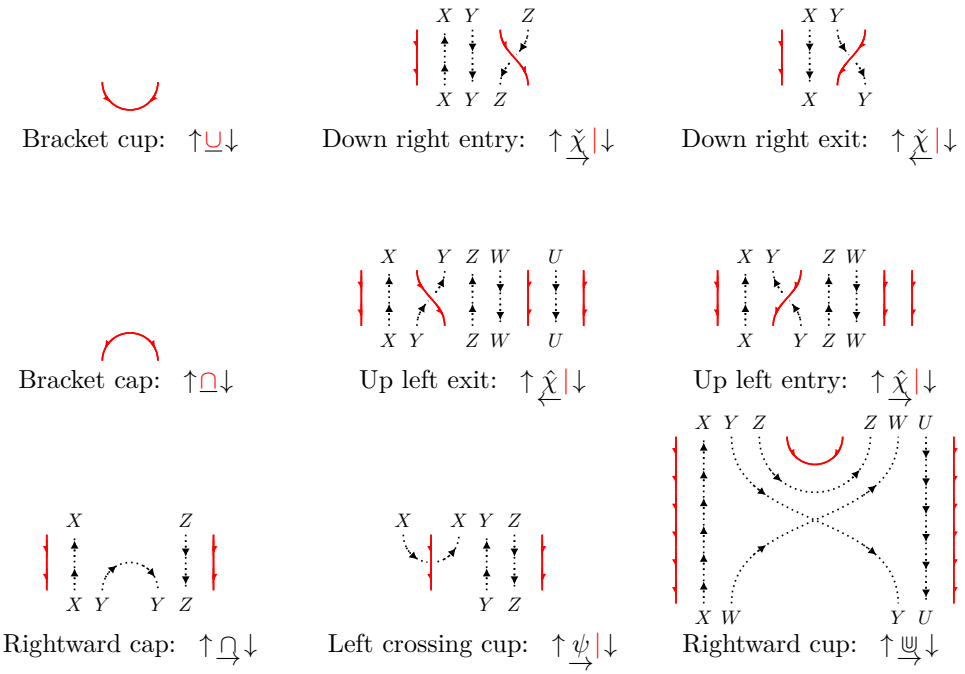


Figure 12 Normalised term generators.

Besides the obvious inclusion $\uparrow\mathscr{C}_\Sigma\downarrow \hookrightarrow \mathscr{C}_\Sigma^{\rightarrow}$, we have a functor in the opposite direction, defined using the normaliser ν.

▶ **Definition 11.** *The normalisation functor is a strong closed, strong symmetric monoidal functor* $\mathcal{N} : \mathscr{C}_\Sigma^{\rightarrow} \to \uparrow\mathscr{C}_\Sigma\downarrow$ *defined as follows*

- *On objects it sends every object to its normal form.*
- *On morphisms it is defined inductively as follows:*
  - $\alpha \mapsto (\uparrow\alpha\downarrow)$ *for all* α *in Figure 4;*
  - $f \mapsto \nu \circ f \circ \nu^{-1}$ *for all* $f \in \Sigma$*;*
  - *all other generators are mapped in the only possible way.*

The fact that this is indeed a functor, as well as being strong closed and strong symmetric monoidal, immediately follows from the inductive definition.

In essence, the functor above gives an equivalence between the "unbiased" and "biased" presentations of closed symmetric monoidal categories.

▶ **Theorem 12.** *There is a closed symmetric monoidal equivalence* $\mathscr{C}_\Sigma \simeq \uparrow\mathscr{C}_\Sigma\downarrow$.

Proof. Follows from the normaliser being a natural isomorphism (see [46, Theorem 4.6]). ◀

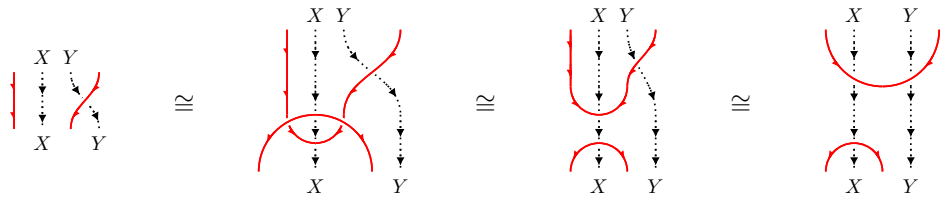
3.2 Decomposition

In this section we give yet another equivalent characterisation of the category of string diagrams with brackets. Specifically, we show that normalised diagrams can be expressed as composition of the co/evaluation diagrams shown in the proof of Theorem 7.

▶ **Definition 13.** *The closed symmetric monoidal category* $\mathscr{C}_\Sigma^\perp$ *is defined as the subcategory of* $\uparrow\mathscr{C}_\Sigma\downarrow$ *with the same objects but whose morphisms are generated by compositions of the co/evaluation diagrams.*

▶ **Lemma 14.** *There is an equality* $\uparrow\mathscr{C}_\Sigma\downarrow = \mathscr{C}_\Sigma^\perp$.

Proof. The equality is proved in [46, Lemma 4.9], by showing that each generator of $\uparrow\mathscr{C}_\Sigma\downarrow$ decomposes into composites, tensors and bracketings of co/evaluation diagrams. Here we show only the case of down right exit ($\uparrow \underset{\leftarrow}{\check{\chi}} | \downarrow$), where we first apply the merge equation, then naturality and, finally, syntactic sugar.



Hence down right exit decomposes as $[I, X \otimes Y] \xrightarrow{\epsilon^I_{X\otimes Y}} X \otimes Y \xrightarrow{\eta^I_X \otimes Y} [I, X] \otimes Y$. ◀

In order to prove the completeness theorem it is necessary to introduce the notion of *interpretation* of a closed monoidal signature into a closed symmetric monoidal category.

▶ **Definition 15.** *An interpretation* $\mathcal{I} = (\alpha_\mathcal{S}, \alpha_\Sigma)$ *of a closed monoidal signature* $(\mathcal{S}, \Sigma, ar, coar)$ *in a closed symmetric monoidal category* $\mathscr{C}$*, consists of a pair of functions* $\alpha_\mathcal{S} : \mathcal{S} \to Obj(\mathscr{C})$ *and* $\alpha_\Sigma : \Sigma \to Ar(\mathscr{C})$ *such that, for all* $f \in \Sigma$, $\alpha_\Sigma(f) : \alpha^\sharp_\mathcal{S}(ar(f)) \to \alpha^\sharp_\mathcal{S}(coar(f))$ *is a morphism of* $\mathscr{C}$*, where* $\alpha^\sharp_\mathcal{S} : \mathcal{S}^\sharp \to Obj(\mathscr{C})$ *is the inductive extension of* $\alpha_\mathcal{S}$.

A closed symmetric monoidal category $\mathscr{C}$ *is* freely generated *by* Σ *if, for all closed symmetric monoidal categories* $\mathscr{D}$ *and interpretations* $\mathcal{I}$ *of* Σ *in* $\mathscr{D}$*, there is a* unique *closed symmetric monoidal functor* $[\![-]\!]_\mathcal{I} : \mathscr{C} \to \mathscr{D}$ *extending* $\mathcal{I}$*, i.e.* $[\![f]\!]_\mathcal{I} = \alpha_\Sigma(f)$ *for every* $f \in \Sigma$.

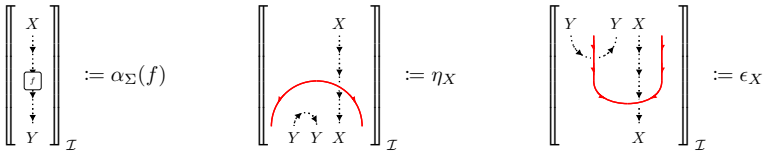
▶ **Theorem 16.** *$\mathscr{C}_\Sigma$ is – up to equivalence – the free closed symmetric monoidal category generated by the closed monoidal signature $(\mathcal{S}, \Sigma)$.*

Proof. By normalisation (Theorem 12) and decomposition (Lemma 14), we can safely work with the category $\mathscr{C}_\Sigma^\perp$ and define a closed symmetric monoidal functor $[\![-]\!]_\mathcal{I} : \mathscr{C}_\Sigma^\perp \to \mathscr{D}$, for any interpretation $\mathcal{I} = (\alpha_\mathcal{S}, \alpha_\Sigma)$ and any closed symmetric monoidal category $\mathscr{D}$.

Recall that objects of $\uparrow\mathscr{C}_\Sigma\downarrow$, and thus of $\mathscr{C}_\Sigma^\perp$, have a certain shape. In particular, now the dual of an object may only appear on the left part of a bracketing string. Thus, we define the interpretation functor on objects inductively as follows:

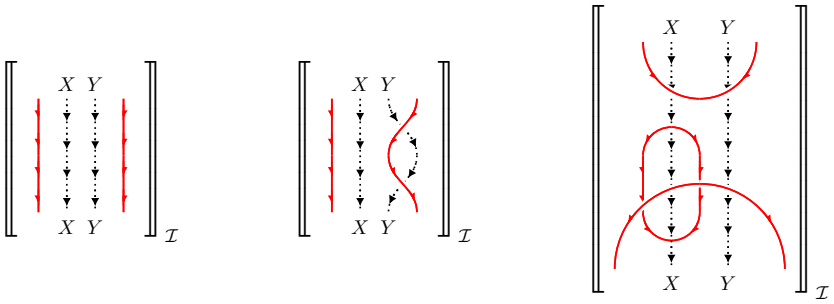
$$\begin{array}{rclrcl} [\![I]\!]_\mathcal{I} & \coloneqq & 1 & [\![A]\!]_\mathcal{I} & \coloneqq & \alpha_\mathcal{S}(A) \\ [\![XY]\!]_\mathcal{I} & \coloneqq & [\![X]\!]_\mathcal{I} \otimes [\![Y]\!]_\mathcal{I} & [\![[X^* Y]]\!]_\mathcal{I} & \coloneqq & [[\![X]\!]_\mathcal{I}, [\![Y]\!]_\mathcal{I}] \end{array}$$

To define $[\![-]\!]_\mathcal{I}$ on morphisms, it suffices to say where to map the generators in Σ and the co/evaluation diagrams. As expected, we map the generators according to the interpretation $\mathcal{I}$ and the co/evaluation diagrams to the co/unit pair of $\mathscr{D}$.



To establish soundness of the interpretation we need to ensure that $[\![-]\!]_\mathcal{I}$ is well-defined. Specifically we need to check that, the (normalised version of the) equations in Figure 6 hold after applying the interpretation assignment.

Take for example the crossing equality. We have that, by decomposition, the interpretation of the central diagram should be equal to the interpretation of the rightmost diagram.



Note that the interpretation of the rightmost diagram is given by the composite below, which, by naturality and the triangular equations, is equal to the identity, i.e. the interpretation of the leftmost diagram.

$$[1, X \otimes Y] \xrightarrow{\epsilon^1_{X\otimes Y}} X \otimes Y \xrightarrow{\eta^1_X \otimes Y} [1, X] \otimes Y \xrightarrow{\eta^1_{X\otimes Y}} [1, [1, X] \otimes Y] \xrightarrow{[1, \epsilon^1_X \otimes Y]} [1, X \otimes Y].$$

The other equalities follow similarly. To conclude, observe that $[\![-]\!]_\mathcal{I}$ is a functor of closed symmetric monoidal categories since, by definition, it preserves the tensor, symmetries and the co/evaluation morphisms. Moreover, observe that any other closed symmetric monoidal functor $F \colon \mathscr{C}_\Sigma^\perp \to \mathscr{D}$ extending $\mathcal{I}$ must also preserve the co/evaluation morphisms and hence $[\![-]\!]_\mathcal{I}$ is the only such functor. ◀

4 Conclusions and Future Work

We introduced a string diagram formalism which, by virtue of Theorem 16, serves as a universal language for closed symmetric monoidal categories. Compared to other approaches in the literature, our contribution offers (1) a completeness result and (2) a fully diagrammatic language that avoids mixed algebraic notation and requires no additional machinery beyond strings.

Given the ubiquity of closed symmetric monoidal categories and their cartesian variants, we believe our formalism constitutes a valuable tool for reasoning across a range of domains, from pure mathematics to logic and theoretical computer science.

As future work, we plan to investigate the hypergraph interpretation of our string diagrams, following the approach of [11]. This characterisation, beyond offering an alternative definition, also has computational significance: hypergraphs can serve as data structures for potential implementations in, for example, proof assistants or compiler optimisations. Some inspiration may also be drawn from the bigraphical approach taken in [25, 26].

For the non-symmetric setting, variants of these diagrams can be used in the study of biclosed bicategories, such as bicategories of relations, or profunctors. In particular, the recent trend of adopting bicategorical models as proof-relevant denotational semantics for programming languages [14, 21] and linear logic [20, 38] opens up new possibilities for applying our diagrams in this context.

In topology there are studies of fixed point theorems using traces in bicategories [43, 44]. If a compact closed bicategory is also biclosed compositionally, then the biclosed structure can be used to define a *co*trace [5, 45]. A wide variety of mathematical phenomena can be expressed as a cotrace – including ends and Hochschild cohomology – and we believe that results about these phenomena can now be expressed and derived purely graphically.

References

1 Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *19th IEEE Symposium on Logic in Computer Science (LICS 2004), 14-17 July 2004, Turku, Finland, Proceedings*, pages 415–425, 2004. doi:10.1109/LICS.2004.1319636.

2 Matteo Acclavio. Proof diagrams for multiplicative linear logic: Syntax and semantics. *Journal of Automated Reasoning*, 63(4):911–939, 2019. doi:10.1007/S10817-018-9466-4.

3 Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger, and Fabio Zanasi. Rewriting for monoidal closed categories. In Amy P. Felty, editor, *7th International Conference on Formal Structures for Computation and Deduction, FSCD 2022, August 2-5, 2022, Haifa, Israel*, volume 228 of *LIPIcs*, pages 29:1–29:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.FSCD.2022.29.

4 John C. Baez and Mike Stay. Physics, topology, logic and computation: A rosetta stone. *Lecture Notes in Physics*, 813:95–174, 2011.

5 Justin Barhite. Bicategorical traces and cotraces, 2024. arXiv:2307.13070.

6 Richard F Blute, J Robin B Cockett, Robert AG Seely, and Todd H Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996.

7 Guillaume Boisseau and Pawel Sobocinski. String diagrammatic electrical circuit theory. In Kohei Kishida, editor, *Proceedings of the Fourth International Conference on Applied Category Theory, ACT 2021, Cambridge, United Kingdom, 12-16th July 2021*, volume 372 of *EPTCS*, pages 178–191, 2021. doi:10.4204/EPTCS.372.13.

8 Filippo Bonchi, Alessandro Di Giorgio, and Elena Di Lavore. A diagrammatic algebra for program logics. In *International Conference on Foundations of Software Science and Computation Structures*, pages 308–330. Springer Nature Switzerland Cham, 2025. doi:10.1007/978-3-031-90897-2_15.

9 Filippo Bonchi, Alessandro Di Giorgio, Nathan Haydon, and Pawel Sobocinski. Diagrammatic algebra of first order logic. In *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 1–15, 2024. doi:10.1145/3661814.3662078.

10 Filippo Bonchi, Alessandro Di Giorgio, and Alessio Santamaria. Deconstructing the calculus of relations with tape diagrams. *Proceedings of the ACM on Programming Languages*, 7(POPL):1864–1894, 2023. doi:10.1145/3571257.

11 Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String diagram rewrite theory I: rewriting with frobenius structure. *J. ACM*, 69(2):14:1–14:58, 2022. doi:10.1145/3502719.

12 Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical Conjunctive Queries. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 13:1–13:23, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2018.13.

13 Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *Proceedings of the 25th International Conference on Concurrency Theory (CONCUR)*, pages 435–450. Springer, 2014.

14 Pierre Clairambault and Simon Forest. The cartesian closed bicategory of thin spans of groupoids. In *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2023. doi:10.1109/LICS56636.2023.10175754.

15 J Robin B Cockett and Robert AG Seely. Proof theory for full intuitionistic linear logic, bilinear logic, and mix categories. *Theory and Applications of categories*, 3(5):85–131, 1997.

16 Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. doi:10.1017/9781316219317.

17 Cole Comfort, Antonin Delpeuch, and Jules Hedges. Sheet diagrams for bimonoidal categories. *arXiv preprint*, 2020. arXiv:2010.13361.

18 Kosta Dosen and Zoran Petrić. *Proof-Net Categories*. Polimetrica sas, 2007.

19 Samuel Eilenberg and Norman Steenrod. *Foundations of algebraic topology*. Princeton University Press, 2015.

20 Marcelo Fiore, Zeinab Galal, and Hugo Paquet. Stabilized profunctors and stable species of structures. *Logical Methods in Computer Science*, 20, 2024. doi:10.46298/LMCS-20(1:17)2024.

21 Marcelo Fiore, Nicola Gambino, Martin Hyland, and Glynn Winskel. The cartesian closed bicategory of generalised species of structures. *Journal of the London Mathematical Society*, 77(1):203–220, 2008.

22 Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976. doi:10.1080/00927877608822127.

23 Peter J Freyd and David N Yetter. Braided compact closed categories with applications to low dimensional topology. *Advances in mathematics*, 77(2):156–182, 1989.

24 Ella Gale, Leo Lobski, and Fabio Zanasi. A categorical approach to synthetic chemistry. In *International Colloquium on Theoretical Aspects of Computing*, pages 276–294. Springer, 2023. doi:10.1007/978-3-031-47963-2_17.

25 Richard Garner, Tom Hirschowitz, and Aurélien Pardon. Graphical presentations of symmetric monoidal closed theories. *arXiv preprint arXiv:0810.4420*, 2008. arXiv:0810.4420.

26 Richard Garner, Tom Hirschowitz, and Aurélien Pardon. Variable binding, symmetric monoidal closed theories, and bigraphs. In *International Conference on Concurrency Theory*, pages 321–337. Springer, 2009. doi:10.1007/978-3-642-04081-8_22.

27 Dan Ghica and Fabio Zanasi. String diagrams for λ-calculi and functional computation. *arXiv preprint*, 2023. arXiv:2305.18945.

28 Dan R. Ghica and Achim Jung. Categorical semantics of digital circuits. In *2016 Formal Methods in Computer-Aided Design (FMCAD)*, pages 41–48, 2016. doi:10.1109/FMCAD.2016.7886659.

29 Dominic JD Hughes. Simple free star-autonomous categories and full coherence. *Journal of Pure and Applied Algebra*, 216(11):2386–2410, 2012.

30 Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint, Dec*, pages 80688–7, 1997.

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

32 André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. In *Mathematical proceedings of the cambridge philosophical society*, volume 119 (3), pages 447–468. Cambridge University Press, 1996.

33 G.M. Kelly and S. Maclane. Coherence in closed categories. *Journal of Pure and Applied Algebra*, 1(1):97–140, 1971. doi:10.1016/0022-4049(71)90013-2.

34 Miguel L. Laplaza. Coherence for distributivity. In G. M. Kelly, M. Laplaza, G. Lewis, and Saunders Mac Lane, editors, *Coherence in Categories*, Lecture Notes in Mathematics, pages 29–65, Berlin, Heidelberg, 1972. Springer. doi:10.1007/BFb0059555.

35 Tom Leinster. *Higher operads, higher categories*. London Mathematical Society Lecture Note Series (298). Cambridge University Press, 2004.

36 L. Méhats and S. Soloviev. Coherence in smccs and equivalences on derivations in imll with unit. *Annals of Pure and Applied Logic*, 147(3):127–179, 2007. doi:10.1016/j.apal.2007.03.005.

37 Paul-André Melliès. Functorial Boxes in String Diagrams. In Zoltán Ésik, editor, *Computer Science Logic*, Lecture Notes in Computer Science, pages 1–30, Berlin, Heidelberg, 2006. Springer. doi:10.1007/11874683_1.

38 Paul-André Melliès. Template games and differential linear logic. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019. doi:10.1109/LICS.2019.8785830.

39 Ike Nassi and Ben Shneiderman. Flowchart techniques for structured programming. *ACM Sigplan Notices*, 8(8):12–26, 1973. doi:10.1145/953349.953350.

40 R. Penrose. Applications of negative dimension tensors. In D. J. A. Welsh, editor, *Combinatorial Mathematics and its Applications*, pages 221–244. Academic Press, 1971.

41 James L Peterson. Petri nets. *ACM Computing Surveys (CSUR)*, 9(3):223–252, 1977. doi:10.1145/356698.356702.

42 Robin Piedeleu, Mateo Torres-Ruiz, Alexandra Silva, and Fabio Zanasi. A complete axiomatisation of equivalence for discrete probabilistic programming. In *European Symposium on Programming*, pages 202–229. Springer, 2025. doi:10.1007/978-3-031-91121-7_9.

43 Kate Ponto. Fixed point theory and trace for bicategories, 2008. arXiv:0807.1471.

44 Kate Ponto and Michael Shulman. Traces in symmetric monoidal categories. *Expositiones Mathematicae*, 32(3):248–273, 2014. doi:10.1016/j.exmath.2013.12.003.

45 Callum Reader. Scalar enrichment and cotraces in bicategories, 2024. arXiv:2403.14475.

46 Callum Reader and Alessandro Di Giorgio. String diagrams for closed symmetric monoidal categories, 2025. arXiv:2512.06499.

47 Don D. Roberts. *The Existential Graphs of Charles S. Peirce*. De Gruyter Mouton, 1973.

48 P. Selinger. *A Survey of Graphical Languages for Monoidal Categories*, volume 813 of *Lecture Notes in Physics*, pages 289–355. Springer, Berlin, Heidelberg, 2010. doi:10.1007/978-3-642-12821-9_4.

49 Michael Shulman. *-autonomous envelopes and conservativity. *arXiv preprint arXiv:2004.08487*, 2020.

50 Aleksei Tiurin, Chris Barrett, Dan R Ghica, and Nick Hu. Equivalence hypergraphs: Dpo rewriting for monoidal e-graphs. In *2025 40th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 209–222. IEEE, 2025. doi:10.1109/LICS65433.2025.00023.

51 Simon Willerton. Surface diagrams for closed monoidal categories. Private communication, 2025.

A Examples

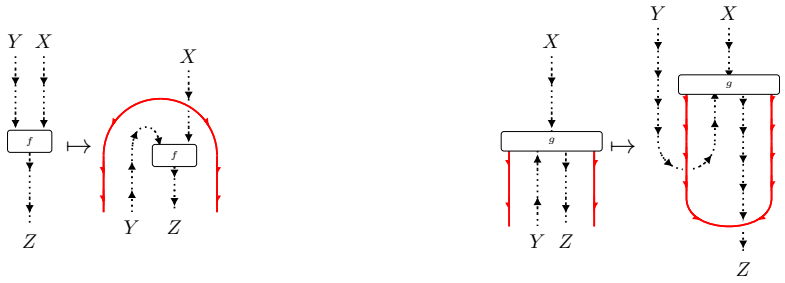
In this section we illustrate several motivating examples. We start with the archetypal case of currying and uncurrying in a closed symmetric monoidal category. This highlights the advantages of our diagrammatic language over other approaches in the literature.

Next, we extend the calculus with copy and delete morphisms to obtain a *cartesian* closed structure, and show how this allows us to reason diagrammatically about the λ-calculus.

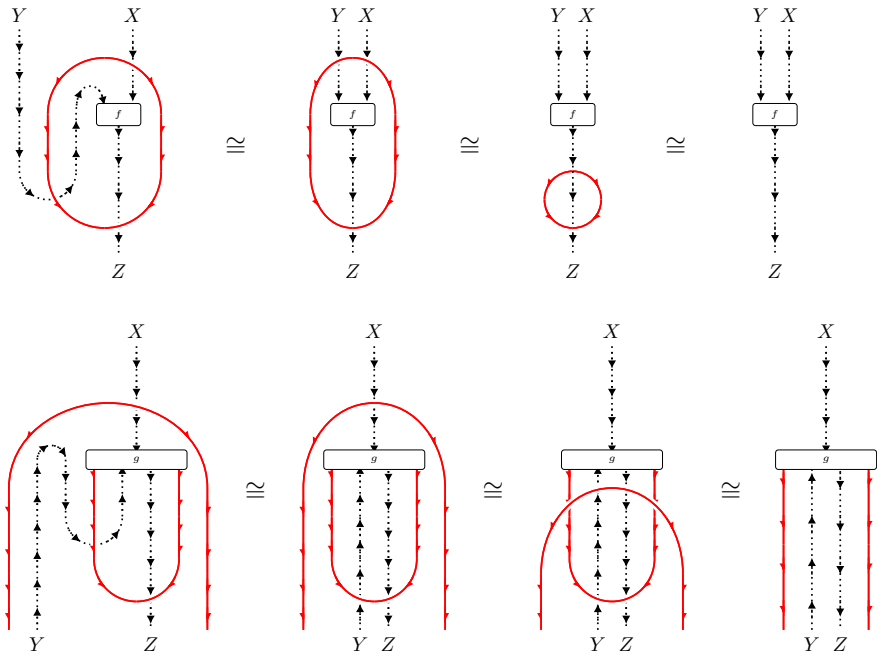
Our third example demonstrates how the self-enrichment of a closed symmetric monoidal category can be expressed graphically, and how the associativity and unitality of the enrichment follow naturally from the diagrams.

Finally, we use our calculus to illustrate a well-known coherence issue involving units in closed monoidal categories.

▶ **Example 17** (Currying). The archetypal example is that of *currying*. Given two morphisms $f\colon Y \otimes X \to Z$ and $g\colon X \to [Y, Z]$, we define the currying of f and the uncurrying of g as the diagrams on the left and on the right, respectively, below.



Note that in both cases, the operations correspond diagrammatically to bending the (un)curried wire. This visual intuition hints at an underlying bijection between the two operations, which we establish formally below.



Hence we have a bijection.

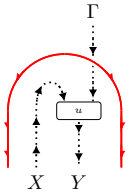
▶ **Example 18** (λ-calculus)**.** Expanding on the previous example, we give a diagrammatic encoding of the simply typed λ-calculus, that we recall now. Fixed a set of variables $\mathcal{X}$ and a set of basic types $\mathcal{S}$, we build types as $X \coloneqq A \mid X \to X$, where $A \in \mathcal{S}$. Terms are generated according to the following grammar, where $x \in \mathcal{X}$:

$$u \coloneqq x \mid \lambda x.u \mid u\ u$$

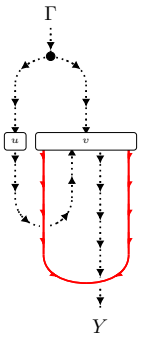
Amongst terms, we consider only those that can be typed according to type judgements of the form $\Gamma \vdash u\colon X$, where $\Gamma = x_1\colon X_1, \dots, x_n\colon X_n$ is a variable type assignment. Type judgements are defined according to the following rules:

$$\Gamma, x\colon X \vdash x\colon X \qquad \frac{\Gamma, x\colon X \vdash u\colon Y}{\Gamma \vdash \lambda x.u\colon X \to Y} \qquad \frac{\Gamma \vdash u\colon X \qquad \Gamma \vdash v\colon X \to Y}{\Gamma \vdash v\ u\colon Y}$$

We interpret typed terms in our category of string diagrams, additionally supplied with natural copy and discard operations that satisfy the axioms of cocommutative comonoids (see e.g. Table 8 in [48]). By Fox's Theorem [22], the additional structure makes the category cartesian closed. The diagrammatic encoding is then given as follows:



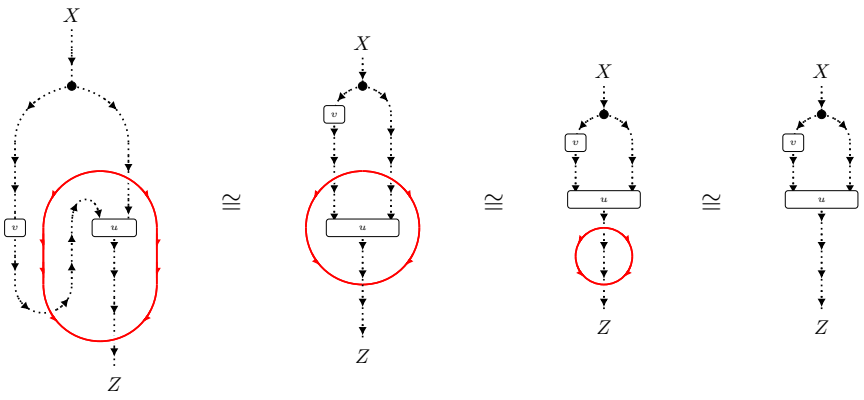
$\Gamma \vdash \lambda x.u\colon X \to Y$



$\Gamma \vdash v\ u\colon Y$

$\Gamma, x\colon X \vdash x\colon X$

We conclude by showing how β-reduction is performed in our language.

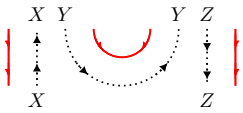


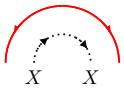
The proof above consists of yanking strings and sliding or popping bubbles, emphasising the geometric nature of string diagrams with brackets. We invite the reader to compare this with the proof of β-reduction in [27], which, instead, relies more heavily on algebraic reasoning.

▶ **Example 19** (Self enrichment)**.** One well-known and useful feature of closed symmetric monoidal categories is that they can be enriched over themselves using the internal hom. In other words, for any objects X, Y, and Z in the category there are canonical composition and identity morphisms:

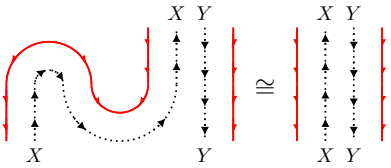
$$\overline{\circ}_{X,Y,Z}\colon [X,Y]\otimes[Y,Z]\to[X,Z] \quad \text{and} \quad \overline{\mathrm{id}}_X\colon I\to[X,X]$$

In our language these are represented by diagrams the left- and right-hand diagrams below. These, of course, adhere to associativity and unitality axioms.





The unitality axiom is shown below, using Figure 7d as well as the yanking equations for strings and bracketing strings. Associativity also follows from Figure 7d.



▶ **Example 20** (Triple unit diagram)**.** In their paper on coherence in closed symmetric monoidal categories, Kelly and Mac Lane [33] show that there is no full coherence in closed symmetric monoidal categories, in the sense that we can generate non-commuting diagrams from the structural morphisms.

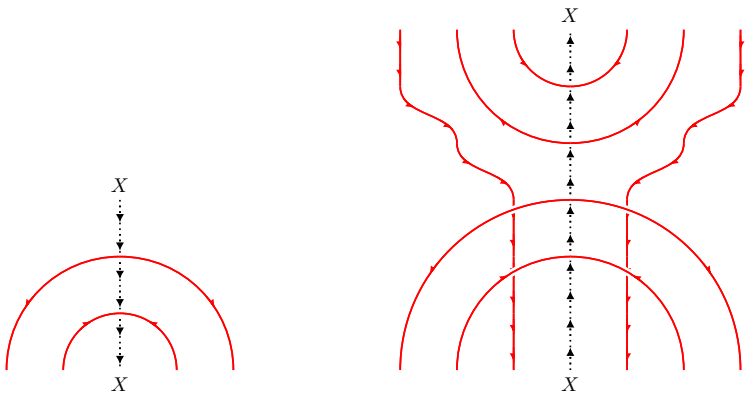
The typical example is the diagram below, that does not commute, for instance, in the closed monoidal category of pointed sets [36, 29]:

$$\begin{array}{ccc} [[[X,I],I],I] & = & [[[X,I],I],I] \\ & {\scriptstyle [v_X,I]}\searrow \quad \nearrow{\scriptstyle v_{[X,I]}} & \\ & [X,I] & \end{array}$$

where $v_X\colon X\to[[X,I],I]$ is given by the composite

$$X \xrightarrow{\eta^{X,I}} [[X,I],X\otimes[X,I]] \xrightarrow{\left[[X,I],\epsilon_I^X\right]} [[X,I],I]\,.$$

To see how this phenomenon appears in our diagrammatic language, note that v_X is depicted by the string diagram below on the left. Meanwhile, the composite in the pasting diagram is represented by the string diagram on the right.



We know by Theorem 16, that this diagram does not collapse to the identity diagram, but intuitively this follows from the fact that the cups and caps in the string diagram above cannot be merged.