

The Many-Worlds Calculus

Kostia Chardonnet ✉ 🏠

Université Paris-Saclay, CNRS, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France.
Université Paris Cité, CNRS, IRIF, F-75006, Paris, France

Marc de Visme ✉ 🏠

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, 91190, Gif-sur-Yvette, France

Benoît Valiron ✉ 🏠 

Université Paris-Saclay, CNRS, CentraleSupélec, ENS Paris-Saclay, LMF, 91190, Gif-sur-Yvette, France

Renaud Vilmart ✉ 🏠 

Université Paris-Saclay, CNRS, ENS Paris-Saclay, Inria, LMF, 91190, Gif-sur-Yvette, France

Abstract

In this paper, we explore the interaction between two monoidal structures: a multiplicative one, for the encoding of pairing, and an additive one, for the encoding of choice. We propose a PROP to model computation in this framework, where the choice is parametrized by an algebraic side effect: the model can support regular tests, probabilistic and non-deterministic branching, as well as quantum branching, i.e. superposition.

The graphical language comes equipped with a denotational semantics based on linear applications, and an equational theory. We prove the language to be universal, and the equational theory to be complete with respect to the semantics.

2012 ACM Subject Classification Theory of computation → Quantum computation theory; Theory of computation → Linear logic; Theory of computation → Equational logic and rewriting

Keywords and phrases Quantum Computation, Linear Logic, Graphical Languages, Geometry of Interaction

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The basic execution flow of a computation is arguably based on three notions: sequences, tuples and branches. Sequences form the building block of compositionality, tuples are what makes it possible to consider multiple pieces of information together, while branches allow the behavior to change depending on the inputs or on the state of the system.

Ranging from (sometimes informal) flow-chart languages [3] to sophisticated structures such as interaction or proof nets [25, 19], graphical languages are commonly used to represent the possible control flow of a computation. On a formal level, a graphical language is a PROP [24], that is, a symmetric, strict monoidal structure $(\mathcal{C}, \top, \boxtimes)$ whose objects are of the form $W \boxtimes \dots \boxtimes W$. The object W is a “wire”, and any object stands for a bunch of wires. The monoidal structure formalizes how the bunching of wires behaves.

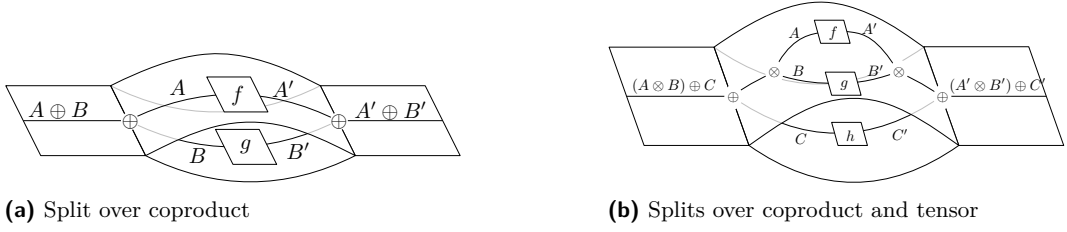
Two Canonical Monoidal Structures. The monoidal structure of a PROP is very versatile. On one hand, it can be considered in a *multiplicative* way, with $A \boxtimes B$ seen as the pairing of an element of type A and an element of type B . This approach is one followed in the design of MLL proof-nets for instance [14]. On the other hand, one can consider the monoidal structure in an *additive* way, with \boxtimes for instance being a co- or a bi-product. Standard examples are the category \mathbf{FinRel} of finite sets and relations, forming an additive PROP with \boxtimes being the disjoint union, or the category of finite dimensional vector spaces (or semi-modules) and linear maps, with \boxtimes being the cartesian product. From a computational



© Kostia Chardonnet and Marc de Visme and Benoît Valiron and Renaud Vilmart;
licensed under Creative Commons License CC-BY 4.0

Leibniz International Proceedings in Informatics

LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany



■ **Figure 1** Examples of branchings

perspective, an additive monoidal structure can be regarded as the possibility to *choose* a computational path upon the state of the input. Depending on the underlying system, this choice can be regarded as deterministic (if based on Set), non-deterministic (if based on Rel), probabilistic (if based on a suitable semi-module), *etc.*

To be able to handle both pairing and branching in a PROP, we cannot uniquely identify \boxtimes as being multiplicative additive. We instead need to *extend* the PROP with two additional monoidal structures, one for pairing (\otimes) and one for branching (\oplus).

In this paper, we focus on a framework where these two monoidal structures are available. Graphical languages for such a setting usually rely on a notion of *sheet*, or *worlds*, to handle general branching [16, 26]. Figure 1a shows for instance how to represent the construction of the morphism $f \oplus g : A \oplus A' \rightarrow B \oplus B'$ out of $f : A \rightarrow A'$ and $g : B \rightarrow B'$. The symbol “ \oplus ” stands for the “split” of worlds. Such a graphical language therefore comes with two distinct “splits”: one for the monoidal structure —leaving inside one specific world—, and one for the coproduct —splitting worlds—. They can be intertwined, as shown in Figure 1b. Another approach followed by [13] externalizes the two products (tensor product and coproduct) into the structure of the diagrams themselves, at the price of a less intuitive tensor product and a form of synchronization constraint.

However, in the state of the art this “splitting-world” understanding has only been carried for deterministic or probabilistic branching [14, 16, 30]. These existing approaches do not support more exotic branchings, such as *quantum superposition*.

Quantum Computation. Conventional wisdom has it that quantum computation is about *quantum data in superposition*. In the standard model, the memory holding quantum data is encapsulated inside a coprocessor accessed through a simple interface: The coprocessor holds individually addressable registers holding *quantum* bits, on which one can apply a fixed set of operations —*gates*— specified by the interface. If some of these gates can generate superposition of data, this is kept inside the coprocessor and opaque to the programmer. A typical interaction with the coprocessor is a purely classical sequence of elementary operations of the form “Apply gate X to register n ; apply gate Y to register m ; *etc.*”. Such a sequence of instructions is usually represented as a *quantum circuit*. In this model, a quantum program is then a conventional program building a quantum circuit and sending it as a batch-job to the coprocessor.

From a semantical perspective, the *state* of a quantum memory consisting of n quantum bits is a vector in a 2^n -dimensional Hilbert space. A (pure) quantum circuit is a linear, sequential description of elementary operations describing a *linear, unitary map* on the state space.

Coming all the way from Feynman’s diagrams [17], graphical languages are commonly used for representing quantum processes. Whether directly based on quantum circuits [20, 14, 27, 7] or stemming from categorical analysis such as the ZX-calculus [12], these formal languages

are still tied to the quantum coprocessor model in the sense that the only monoidal structure that can be applied to quantum information is the (multiplicative) Kronecker product. The only possible branching is based on the (probabilistic) measurement.

Quantum Control Flow. However, one peculiar feature of quantum computation is *non-causal execution paths*. Indeed, the Janus-faced quantum computational paradigm features two seemingly distinct notions of control structure. On the one hand, a quantum program follows *classical* control: it is hosted on the conventional computer governing the coprocessor, and can therefore only enjoy loops, tests and other regular causally ordered sequences of operations. On the other hand, the lab bench turns out to be more flexible than the rigid coprocessor model, permitting more elaborate *purely quantum* computational constructs than what quantum circuits or ZX-calculus allow.

The archetypal example of a quantum computational behavior hardly attainable within quantum circuits or ZX-calculus is the *Quantum Switch*. Consider two quantum bits x and y and two unitary operations U and V acting on y . The problem consists in generating the operation that performs UV on y if x is in state $|0\rangle$ and VU if it is in state $|1\rangle$. As x can be in superposition, in general the operation is then sending $(\alpha|0\rangle + \beta|1\rangle) \otimes |y\rangle$ to $\alpha|0\rangle \otimes (UV|y\rangle) + \beta|1\rangle \otimes (VU|y\rangle)$. It is a *purely quantum test*: not only can we have values in superpositions (here, x) but also *execution orders*. This is in sharp contrast with what happens within the standard quantum coprocessor model.

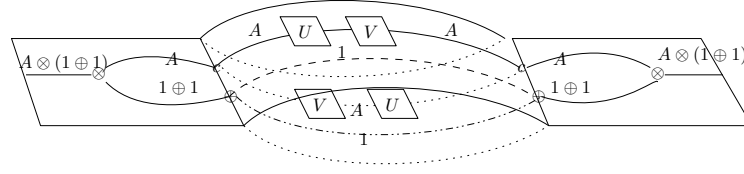
Computational models supporting superpositions of execution orders have been studied in the literature. One strand of research consists in proposing a suitable extension of quantum circuits [8, 28, 31, 32]. These approaches typically aim at discussing the notion of quantum channel from a quantum information theoretical standpoint.

Limitation of Current Approaches and Objective of the Paper. Although there is a finer and finer understanding of superposition of causal orders in the literature, none of the existing PROPs can support both the quantum switch on complex data built from tensors and coproducts. We claim in this paper that the same intuition underlying probabilistic branching can be followed for quantum (and more general) branching. In the conventional case, $1 \oplus 1$ is a regular boolean: either “left” (standing e.g. for True) or “right” (standing e.g. for False). In quantum computation, the sum-type $1 \oplus 1$ can however be understood as a *sum of vector spaces*, giving an alternative interpretation to $1 \oplus 1$: it can be regarded as the type of a quantum bit, superposition of True and False. One should note that this appealing standpoint should be taken cautiously: (Pure) quantum information imposes strong constraints on the structure of the data in superposition: orthogonality and unit-norm have to be preserved [1, 29].

The Quantum Switch can then be naturally understood in this framework. Consider for instance Figure 2, read from left to right: as input, a pair of an element of type A and a quantum bit. Based on the value of the qubit (True or False), the wire A goes in the upper or the lower sheet, and is fed with U then V or V then U . Then everything is merged back together.

Contributions. In this paper, we introduce a new graphical language for quantum computation, based on compact category with biproduct [23]. This language allows us to express any process with both pairing and a general notion of algebraic branching, encompassing deterministic, non-deterministic, probabilistic and quantum branching. We develop a denotational semantic and an equational theory, and prove its soundness and completeness with respect to the semantics. As a case-study, we show how the Quantum Switch can naturally be encoded in the language.

In the paper, the missing proofs can be found in Appendix.



■ Figure 2 Quantum Switch with Worlds

2 The Many-Worlds Calculus

Our calculus is parameterized by a commutative semiring $(R, +, 0, \times, 1)$. It can be instantiated by the complex numbers $(\mathbb{C}, +, 0, \times, 1)$ to represent pure quantum computations, the non-negative real numbers $(\mathbb{R}_{\geq 0}, +, 0, \times, 1)$ for probabilistic computations, or the booleans $(\{\text{False}, \text{True}\}, \text{OR}, \text{False}, \text{AND}, \text{True})$ for non-deterministic computations.

While the goal is to define a graphical language in which each wire can be enabled or disabled depending on the world in which the computation takes place, we first define the category $\mathbf{C_D}$ of diagrams without any "worlds", and will then add the world annotations.

2.1 A First Graphical Language

We define our graphical language within the paradigm of colored PROP [4, 21], meaning that a diagram will be composed of nodes, or *generators*, linked to each other through *colored wires*, that are allowed to cross each others. Additionally, we assume that our colored PROP is compact closed and auto-dual, i.e. we allow to bend wires to obtain a Cup ($\begin{smallmatrix} A \\ \cup \\ A \end{smallmatrix}$) or a Cap ($\begin{smallmatrix} A \\ \cap \\ A \end{smallmatrix}$).

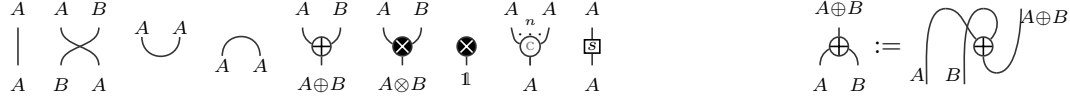
The generators of our language are described in Figure 3 and are respectively the Identity, the Swap, the Cup, the Cap, the Plus, the Tensor, the Unit, the n -ary Contraction for $n \geq 0$, and the Scalar for s ranging over the commutative semiring R . Mirrored versions of those generators are defined as syntactic sugar through the compact closure, as shown for the mirrored Plus on the right-hand-side of Figure 3. Diagrams are read top-to-bottom: the top-most wires are the *input* wires and the bottom-most wires are the *output* wires. The labels A, B , etc correspond to the colors of our PROP. There is one color for every type generated by the syntax¹ $A, B ::= \mathbb{1} \mid A \oplus B \mid A \otimes B$. As such, the Unit starts a wire of type $\mathbb{1}$, the Plus combines two wires of type A and B into a wire of type $A \oplus B$, and similarly the Tensor combines two wires of type A and B into a wire of type $A \otimes B$.

In a colored PROP, an object \mathfrak{A} is simply a list of colors $\mathfrak{A} = A_1 \square \dots \square A_n$ (or $\mathfrak{A} = \emptyset$ for the empty collection). For example, the Plus is a morphism from $A \square B$ to $A \oplus B$. The choice of the notation \square for wires in parallel is uncommon, we use it to put an emphasis on the fact that contrary to languages like the ZX-calculus, wires that are in parallel are not necessarily "in tensor with one another". In fact, $A \square B$ can be understood semantically as "either $A \otimes B$ or $A \oplus B$ ".

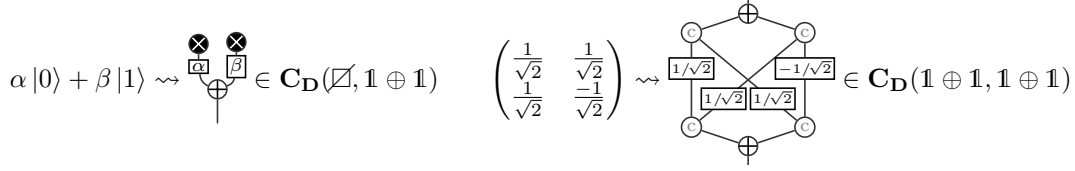
Diagrams are obtained from generators by composing them in parallel (written \square), or sequentially (written \circ). Sequential composition requires the type (and number) of wires to match. We write $\mathbf{C_D}$ for the category of diagrams we defined as such.

$$D_2 \circ D_1 := \begin{array}{c} \begin{array}{|c|} \hline \dots \\ \hline D_1 \\ \hline \dots \\ \hline D_2 \\ \hline \dots \\ \hline \end{array} \end{array} \quad D_1 \square D_2 := \begin{array}{cc} \begin{array}{|c|} \hline \dots \\ \hline D_1 \\ \hline \dots \\ \hline \end{array} & \begin{array}{|c|} \hline \dots \\ \hline D_2 \\ \hline \dots \\ \hline \end{array} \end{array}$$

¹ We do not assume any associativity, distributivity, etc, although types that are semantically equivalent will be made isomorphic through the equational theory.



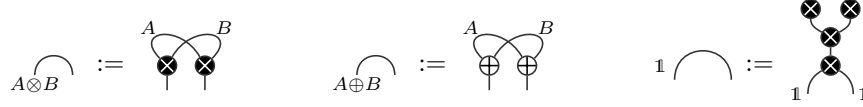
■ **Figure 3** Generators of our First Graphical Language ($n \geq 0, s \in R$)



■ **Figure 4** A Quantum Bit and the Hadamard Unitary

► **Example 1.** We consider $R = \mathbb{C}$. While \mathbf{C}_D lacks the worlds labeling², we can already illustrate our language by encoding some basic quantum primitive in it and show how they operate. In Figure 4 we show the encoding of a quantum bit $\alpha|0\rangle + \beta|1\rangle$ and the Hadamard unitary. In particular, the Plus allows to "build" a new quantum bit from two scalars in parallel or to "open" a quantum bit to recover its corresponding scalars, the left branch corresponding to $|0\rangle$ and the right branch to $|1\rangle$. The meaning of the Contraction is better seen when applying Hadamard to a quantum bit as we show in Figure 5: it allows us to duplicate and sum scalars. The rewriting sequence of Figure 5 is made using the equational theory defined in Section 4, however to correctly define our equational theory, the worlds labeling are required. So while this specific worlds-free rewriting sequence is sound, many other similar worlds-free rewriting sequences are unsound.

► **Remark 2.** Instead of having the Cup and the Cap as generators and defining the mirrored version of each generator through them, one could proceed the other way around by defining the Cap as follows, and the Cup in a mirrored way:



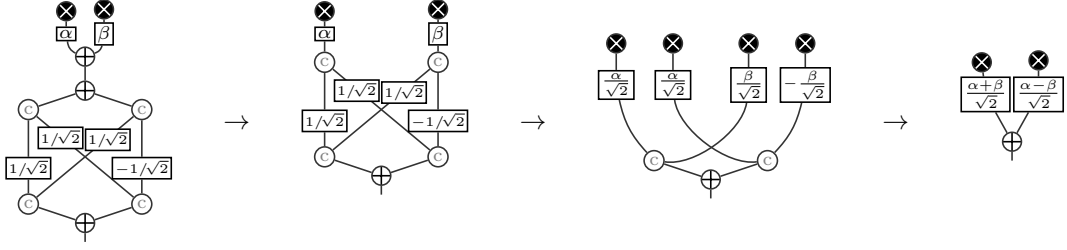
2.2 Adding Worlds Labeling

We now label wires of our diagram with sets of worlds $w \subseteq W$ from a given a world set W . For each world $a \in W$, wires labeled by a set containing a are said to be "enabled in a ", and the others are said "disabled in a ". This allows us to correlate the enabling of wires. Before making this formal, we illustrate this notion through the following example:

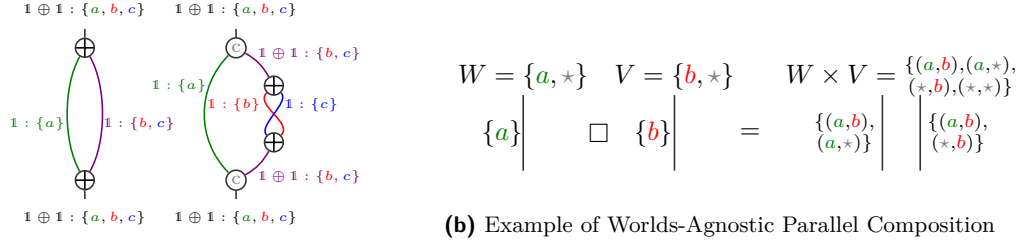
► **Example 3.** The "controlled not" on quantum bits can be represented by the Figure 6a. The figure is split into two parts: the control part on the left-hand-side, and the computational part on the right-hand-side. The idea is that the control part, that uses \oplus , will behave as an *if-then-else* and will bind the world a to the case where the control quantum bit is $|0\rangle$, and the worlds b and c to the case where the control quantum bit is $|1\rangle$. Lastly, the world \star appears nowhere in the labels, and corresponds to "we do not evaluate this circuit at all"³. On the computational part, we apply the identity within the world a , we negate $|0\rangle$ into $|1\rangle$

² We call worlds labeling the attribution of *multiple* worlds to a single wire, as defined in Section 2.2.

³ While not strictly necessary, it is often practical to have a world absent from every wire.



■ **Figure 5** Applying the Hadamard unitary to a quantum bit



(a) Controlled Not with world set $\{a, b, c, \star\}$

■ **Figure 6**

within b , and we negate $|1\rangle$ into $|0\rangle$ within c . The domain and codomain of this labelled diagram are $(1 \oplus 1 : \{a, b, c\}) \square (1 \oplus 1 : \{a, b, c\})$, which we will write $(\mathfrak{A}, \ell_{\mathfrak{A}})$ with an object $\mathfrak{A} = (1 \oplus 1) \square (1 \oplus 1)$ and a labeling function $\ell_{\mathfrak{A}} : 1 \mapsto \{a, b, c\} \quad 2 \mapsto \{a, b, c\}$. Similarly, the "controlled not" above can be seen as a diagram $\mathcal{D}_{\text{CNOT}}$ of $\mathbf{C}_D(\mathfrak{A}, \mathfrak{A})$ together with a labeling function $\ell_{\mathfrak{A}}$ which labels every wire with a set of worlds.

We now give a formal definition of the concept of worlds:

► **Definition 4.** Given a set of worlds W , we define the auto-dual compact closed colored PROP $(\mathbf{MW}_W, \square, \sqcap)$ of many-worlds calculus over W as follows:

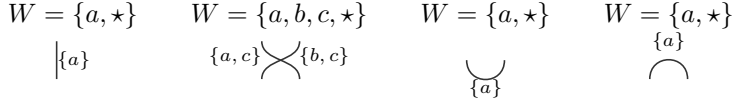
Its colors are the pairs $(A : w)$ of colors A of \mathbf{C}_D and subsets $w \subseteq W$. We write $(\mathfrak{A}, \ell_{\mathfrak{A}})$ for the objects, where \mathfrak{A} is an object of \mathbf{C}_D and $\ell_{\mathfrak{A}}$ is a labeling function from the colors composing \mathfrak{A} to the subsets of W .

Its morphisms $f \in \mathbf{MW}_W((\mathfrak{A}, \ell_{\mathfrak{A}}), (\mathfrak{B}, \ell_{\mathfrak{B}}))$ are pairs (\mathcal{D}_f, ℓ_f) of a morphism $\mathcal{D}_f \in \mathbf{C}_D(\mathfrak{A}, \mathfrak{B})$ and a labeling function ℓ_f from the wires of \mathcal{D}_f to the subsets of W , satisfying the following constraints: The label on an input or output wire of color $(A : w)$ must be equal to w , and

$$\begin{array}{c}
 |w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \begin{array}{c}
 w \\
 \text{---} \\
 w \text{---} \text{---} v \\
 \text{---} \\
 w
 \end{array}
 \quad
 \forall n \geq 0, \quad \begin{array}{c}
 w_1 \text{---} \text{---} w_n \\
 \text{---} \\
 w_1 \sqcup \dots \sqcup w_n
 \end{array}$$

where \sqcup denotes disjoint set-theoretic union. The constraints for the mirrored versions are similar. The sequential composition \circ and the parallel composition \square preserve the labels.

We write $[f]_W : \mathfrak{A} \rightarrow \mathfrak{B}$ for $f = (\mathcal{D}_f, \ell_f) \in \mathbf{MW}_W((\mathfrak{A}, \ell_{\mathfrak{A}}), (\mathfrak{B}, \ell_{\mathfrak{B}}))$ where $\ell_{\mathfrak{A}}$ and $\ell_{\mathfrak{B}}$ can be deduced from ℓ_f using the first restriction on labels. We note that when considering $[f]_W : \mathfrak{A} \rightarrow \mathfrak{B}$ and $[g]_W : \mathfrak{B} \rightarrow \mathfrak{C}$, there is no reason for the labels deduced on \mathfrak{B} to be the same, so there is no reason for $[g \circ f]_W : \mathfrak{A} \rightarrow \mathfrak{C}$ to be defined. More generally, when building two diagrams $[f]_W$ and $[g]_V$, there is no reason for the worlds sets W and V to be equal, but we might still want to be able to compose them with one another.



■ **Figure 7** Canonical labelings in \mathbf{MW}_v

► **Definition 5.** We define the auto-dual compact closed colored PROP $(\mathbf{MW}_v, \square, \boxtimes)$ of many-worlds calculus as follows:

Its colors and objects are the same as the ones of \mathbf{C}_D .

Its morphisms from \mathfrak{A} to \mathfrak{B} are simply morphisms $[f]_W : \mathfrak{A} \rightarrow \mathfrak{B}$ of \mathbf{MW}_W for any finite set W . See Figure 7 for the canonical labelings on the identity, swap, cup and cap. Morphisms are considered up to renaming of the worlds⁴.

The parallel composition is given by $[f]_W \square [g]_V := [f^{-\times V} \square g^{W \times -}]_{W \times V}$ where $f^{\sigma(-)}$ has the same diagram \mathcal{D}_f and has for labels $\ell_{f^{\sigma(-)}}(x) = \sigma(\ell_f(x))$.

The sequential composition is given by $[g]_V \square [f]_W := [g^{(W \times -) \cap Z} \circ f^{(- \times V) \cap Z}]_Z$ where $Z \subseteq W \times V$ is the greatest subset such that this composition is well-defined, as explained in the following example.

► **Example 6** (World Agnostic Composition). In Figure 6b, we show the result of the parallel composition of $[f]_{\{a, *\}} = \mathbf{id}_{A:\{a\}}$ and $[g]_{\{b, *\}} = \mathbf{id}_{A:\{b\}}$ for a color A : the world $(a, *)$ corresponds to the left wire being enabled and the right one disabled, the world $(*, b)$ is the opposite, the world (a, b) is both enabled and the world $(*, *)$ is both disabled.

Then, in Figure 8 we continue by composing with the Cup over $A : \{c, *\}$. To compute the composition, we proceed in two steps: first we handle the situation as if it were a parallel composition, leading to a diagram labelled over $W \times V \times U$, but with multiple contradictory labels on the wire. Then, we eliminate as few worlds as possible to make those labels compatible:

- We eliminate $(a, b, *)$ and $(a, *, *)$ which are on the *left* label but not on the *bottom* one.
- We eliminate $(*, b, c)$ and $(*, *, c)$ which are on the *bottom* label but not on the *left* one.
- Looking at the *right*, we eliminate $(*, b, *)$ and $(a, *, c)$ for similar reasons.

We eliminated six worlds, with the only remaining ones being $Z = \{(a, b, c), (*, *, *)\}$.

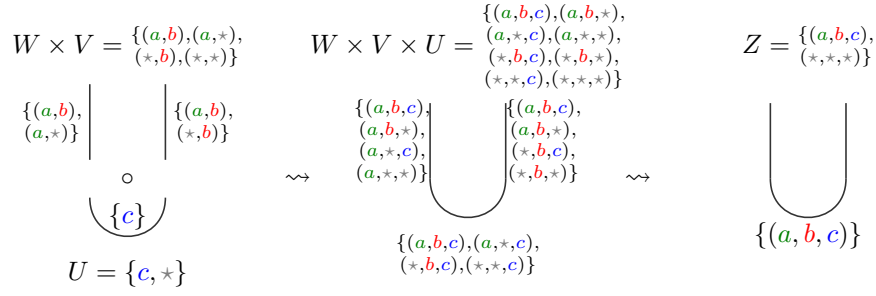
2.3 Representing Computation

As a motivational example, we may see how this Many-Worlds diagrams can be used to represent computations expressed in a language that explicitly uses the two compositions \otimes (through pairs), and \oplus (through pattern-matching), as in [29]. As this translation is not the focus of this paper, a lot of technicalities are glossed over.

Syntax of the Language

The language we present here is adapted from [29], where we consider the values and types together with the enrichment that is linear combinations of terms, but without abstraction

⁴ More precisely, $[(\mathcal{D}_f, \ell_f)]_W = [(\mathcal{D}_f, \sigma \circ \ell_f)]_{\sigma(W)}$ for any σ describing a bijection between W and $\sigma(W)$. Without this quotient, we would not have $[f]_W \circ \mathbf{id} = [f]_W$.



■ **Figure 8** Example of Worlds-Agnostic Sequential Composition

nor recursion. The syntax that is used in the language is given as follows, with scalars s ranging over the commutative semiring R :

$$\begin{aligned}
 (\text{Base types}) \quad A, B &::= \mathbb{1} \mid A \oplus B \mid A \otimes B \\
 (\text{Isos, first-order}) \quad \alpha &::= A \leftrightarrow B \\
 (\text{Values}) \quad v &::= \langle \rangle \mid x \mid \text{inj}_l v \mid \text{inj}_r v \mid \langle v_1, \dots, v_n \rangle \\
 (\text{Expressions}) \quad e &::= v \mid \text{let } \langle x_1, \dots, x_n \rangle = \omega \langle x_1, \dots, x_n \rangle \text{ in } e \mid e + e \mid \alpha e \\
 (\text{Isos}) \quad \omega &::= \{v_1 \leftrightarrow e_1 \mid \dots \mid v_n \leftrightarrow e_n\} \\
 (\text{Terms}) \quad t &::= \langle \rangle \mid x \mid \text{inj}_l t \mid \text{inj}_r t \mid \langle t_1, \dots, t_n \rangle \mid \\
 &\quad \alpha t \mid t + t \mid \omega t \mid \text{let } \langle x_1, \dots, x_n \rangle = t \text{ in } t
 \end{aligned}$$

The language in particular features branching through the inj_l and inj_r constructors, linear combinations of terms and expressions, and crucially *isos* that have type $A \leftrightarrow B$: they turn a term of type A to a term of type B using pattern-matching. The language comes with a predicate (not presented here, although it could be given a diagrammatic meaning), used in the typing rule of isos, to ensure exhaustivity and the non-overlapping character of the left-hand and right expressions of the clauses, allowing in particular to define unitaries (in the complex setting). Constraints on the linear combinations may also be used to enforce probabilistic constraints (i.e. that states are normalised in the quantum setting).

There are two different typing judgements, one for terms \vdash_t and one specific for isos \vdash_ω .

For a more complete description of the language, we refer the reader to [29]. In the following, we will use the shorthands $\mathbf{ff} := \text{inj}_l \langle \rangle$ and $\mathbf{tt} := \text{inj}_r \langle \rangle$.

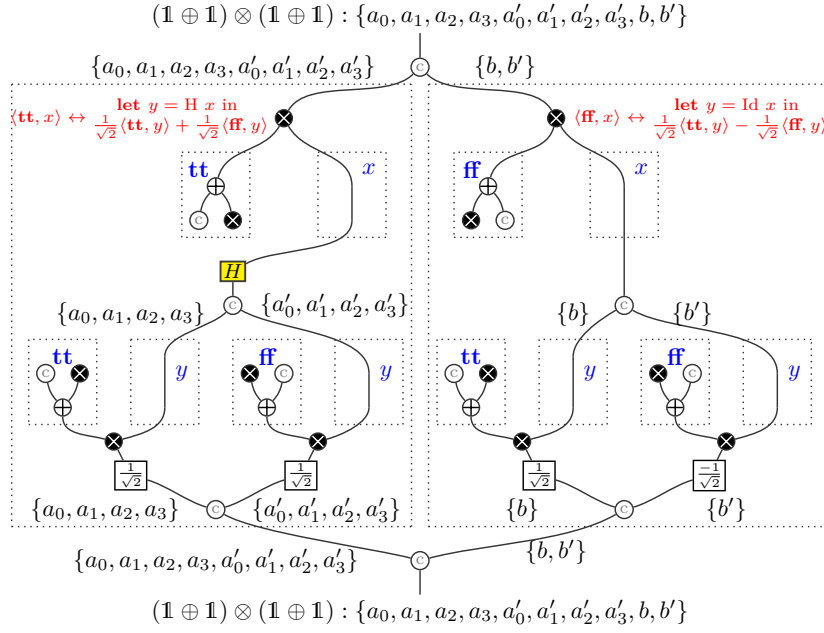
► **Example 7.** In the case where $R = \mathbb{C}$, one can encode the Hadamard gate by:

$$\left\{ \begin{array}{l} \mathbf{ff} \leftrightarrow \frac{1}{\sqrt{2}}(\mathbf{ff} + \mathbf{tt}) \\ \mathbf{tt} \leftrightarrow \frac{1}{\sqrt{2}}(\mathbf{ff} - \mathbf{tt}) \end{array} \right\} : \mathbb{1} \oplus \mathbb{1} \leftrightarrow \mathbb{1} \oplus \mathbb{1}$$

In this setting, any quantum circuit can be encoded by an iso.

Encoding into the Many-Worlds

One can encode any term of the language into a Many-Worlds diagram. Given some typing derivation ξ of a term $x_1 : A_1, \dots, x_n : A_n \vdash_t t : B$ we write $\langle \xi \rangle$ for the function that maps ξ to a diagram in the Many-Worlds Calculus with n input wires of type A_1, \dots, A_n and one output wire of type B . For the typing derivation ξ of an iso $\vdash_\omega \omega : A \leftrightarrow B$, $\langle \xi \rangle$ gives a diagram with one input wire of type A and one output wire of type B .



■ **Figure 9** Representation of the term t from Example 8.

(ξ) is defined inductively over \vdash_t and \vdash_ω as shown in appendix in Figure 14, where the worlds sets are handled by compositions of world-agnostic diagrams.

► **Example 8.** We can represent the term

$$t := \left\{ \begin{array}{l} \langle \mathbf{tt}, x \rangle \leftrightarrow \text{let } y = H x \text{ in } \frac{1}{\sqrt{2}} \langle \mathbf{tt}, y \rangle + \frac{1}{\sqrt{2}} \langle \mathbf{ff}, y \rangle \\ \langle \mathbf{ff}, x \rangle \leftrightarrow \text{let } y = \text{Id } x \text{ in } \frac{1}{\sqrt{2}} \langle \mathbf{tt}, y \rangle - \frac{1}{\sqrt{2}} \langle \mathbf{ff}, y \rangle \end{array} \right\} : (\mathbb{1} \oplus \mathbb{1})^{\otimes 2} \leftrightarrow (\mathbb{1} \oplus \mathbb{1})^{\otimes 2}$$

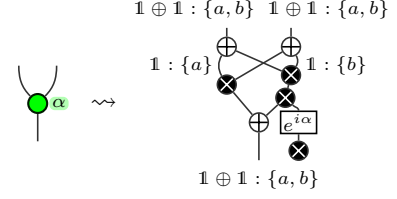
(already given in [29]) as shown in Figure 9. The yellow box stands for the Hadamard gate (which one can build following Example 7). Each line of this isomorphism corresponds to a column of the figure. Each columns start by matching the input as $\langle \mathbf{tt}, x \rangle$ or $\langle \mathbf{ff}, x \rangle$, then compute y from x , and then build the output by following the syntax. The world set is computed by composing each of the blocks of this term, using the world-agnostic composition of \mathbf{MW}_\forall . It can be seen as a subset of $\{a, b\} \times \{c, c'\} \times \{0, 1, 2, 3\}$ where $\{a, b\}$ corresponds to being on the first or second line of the matching, $\{c, c'\}$ being on the left or right of the sum, and $\{0, 1, 2, 3\}$ being the world set of the Hadamard gate.

2.4 Comparison with Other Graphical Languages

The distinctive feature of the Many-Worlds Calculus is that it graphically puts the tensor and the biproduct on an equal footing. By comparison, other graphical language for quantum computing are inherently centered around either one of them. The ZX-calculus [11] and cousin languages ZW- and ZH-Calculi [2, 22], as well as Duncan's Tensor-Sum Logic [16], use the tensor product as the default monoid, while more recent language – particularly for linear optics [9, 15, 10] – use the biproduct. We have a closer look at each of them in the following, and show how – at least part of – each language can be encoded naturally in the Many-Worlds Calculus. Most of them comes equipped with an equational theory. By completeness of our language (Theorem 11 to follow), all the equations expressible in the fragments we consider can be derived in our framework.

2.4.1 ZX-Calculus

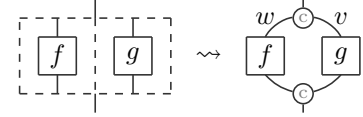
The first difference is the restrictions of the ZX-calculus to computations between qubits, in other words linear map from $\mathbb{C}^{2^n} \mapsto \mathbb{C}^{2^m}$, while our language can encode any linear map from $\mathbb{C}^n \mapsto \mathbb{C}^m$. The Tensor generator allowing the decomposition of \mathbb{C}^{2^n} into instances of \mathbb{C}^2 was already present in the *scalable* extension of the ZX-calculus [5], but the main difference comes from the Plus (and the Contraction).



Additionally, every ZX-diagram can be encoded in our graphical language. The identity, swap, cup and cap of the ZX calculus are encoded by the similar generators over the type $1 \oplus 1$, the Hadamard gate is encoded as in Figure 4, and the green spider is encoded as shown on the right. An encoding for the red spider can then be deduced from those. Diagrams are composed them together with the world-agnostic composition of \mathbf{MW}_\forall .

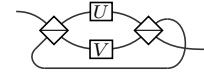
2.4.2 Tensor-Sum Logic

The core difference between their work and ours is the presence of the contraction in our graphical language. They instead rely on an enrichment of their category by a sum, which they represent graphically with boxes. We show on the right how the morphism $f + g$ would be encoded in both their and our language. More generally, their boxes correspond to uses of our contraction generator in a "well-bracketed" way. Another point of difference is their approach to quantum computation, as we do not assign the same semantics to those superpositions of morphisms. In their approach, the superposition is a classical construction and corresponds to the measurement and the classical control flow, while in our approach the superposition is a quantum construction and corresponds to the quantum control.

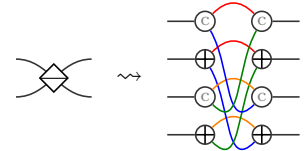


2.5 PBS-Calculus

The PBS Calculus [9] allow one to represent coherent quantum control by the use of *polarising beam splitters* (pbs): whenever a qbit enter a pbs node, depending on the polarity of the qbit it will either go through or be reflected. By making implicit the target system, controlled by the optical system represented by the diagram, the PBS-Calculus allows one to encode the Quantum Switch (depicted on the right). The pbs generator is related to the \oplus of the Many-Worlds.

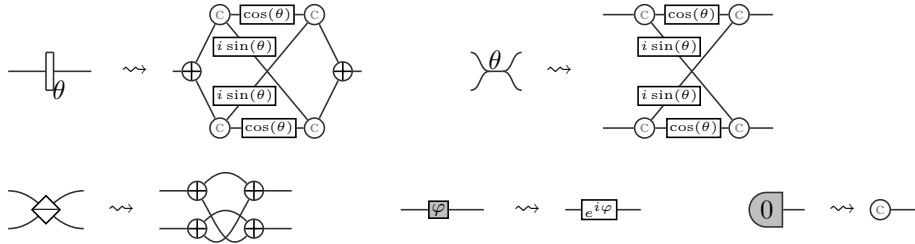


The first main difference with our language is that, since the generators of the PBS-Calculus represent physical components, any PBS-diagram is by construction physical, while our approach is more atomic and decomposes physical components into abstract smaller ones. The second main difference lie in the trace: while they can allow a particle to pass through a wire at most twice, in our system, each wire can be used at most once: more formally, their trace is based on the coproduct while ours is on the tensor product. If we are assured that each wire can only be used once during the computation, any PBS-diagram can be translated to the Many-Worlds calculus directly, with the transformation on the right, where we distinguish the control system (the part of the diagram connected to \oplus s) from the target system (connected to \odot s) which is implicit in the PBS-Calculus.



2.6 LOv-Calculus

In the PBS-Calculus, the qubit in the control system (the one explicitly represented) cannot be put in arbitrary superpositions of $|0\rangle$ and $|1\rangle$ *during* the computation. To allow this feature, we may add some linear optical components to the language's generators, and end up with the LOv-Calculus [10]. In this language, there is no trace and there is a unique photon traveling the circuit, which relieves us of the previous constraint. There is also no need for an implicit target system anymore. All wires at the interface between the generators are of type $1 \oplus 1$, and parallel wires have disjoint sets of worlds. Each generator can then be interpreted as follows:



2.7 Path-Calculus

The Path-Calculus is another recent graphical language for linear optical circuits [15]. Its generators correspond directly to a subset of the Many-Worlds' with $\circ \rightsquigarrow \odot$, $\triangleright \rightsquigarrow \odot$ and $\text{---}^r \rightsquigarrow \text{---}^{\square}$; where each wire has type 1 and where parallel wires are on disjoint sets of worlds. This language is then used as the core for a more expressive language called QPath, which this time cannot be directly encoded in our language, except when restricting the set of generators (specifically to $n = 1$), in which case $|1\rangle$

3 Semantics of the Many-Worlds Calculus

Our calculus represent linear operators between finite dimension R -modules (\mathbf{FdM}_R). In particular, in the case $R = \mathbb{C}$ we use linear operator between finite dimension Hilbert spaces, which correspond to pure quantum computations. More precisely, we will define two semantics, a world-dependent semantics $\llbracket - \rrbracket_a$ for every world $a \in W$, which will be a monoidal functor from \mathbf{MW}_W to \mathbf{FdM}_R , and a worlds-agnostic semantics $\llbracket - \rrbracket$ from \mathbf{MW}_{\forall} to \mathbf{FdM}_R .

We start by defining those semantics on the objects. For every object \mathfrak{A} of \mathbf{C}_D , we define its *enablings* \mathfrak{A}^\bullet as "replacing any number of wire type by \bullet ". For example $(A \square B)^\bullet = \{\bullet \square \bullet, \bullet \square B, A \square \bullet, A \square B\}$. Then, for any object $(\mathfrak{A}, \ell_{\mathfrak{A}})$ of \mathbf{MW}_W and any world $a \in W$, we define $(\mathfrak{A}, \ell_{\mathfrak{A}}) \cap a$ to be the enabling of \mathfrak{A} in which every $(A : w)$ with $a \in w$ is preserved and every $(A : w)$ with $a \notin w$ is replaced by \bullet . For example $(A : \{a\} \square B : \{b\}) \cap a = A \square \bullet$. To each enabling $\mathfrak{E} \in \mathfrak{A}^\bullet$ we associate a R -module $\mathcal{M}_{\mathfrak{E}}$ as follows:

$$\begin{aligned} \mathcal{M}_{A_1 \square \dots \square A_n} &:= \mathcal{M}_{A_1} \otimes \dots \otimes \mathcal{M}_{A_n} & \mathcal{M}_{A \otimes B} &:= \mathcal{M}_A \otimes \mathcal{M}_B \\ \mathcal{M}_{\text{---}} &:= R & \mathcal{M}_{\bullet} &:= R & \mathcal{M}_{\text{---}} &:= R & \mathcal{M}_{A \oplus B} &:= \mathcal{M}_A \oplus \mathcal{M}_B \end{aligned}$$

We can then define the semantics $\llbracket - \rrbracket_a : \mathbf{MW}_W \rightarrow \mathbf{FdM}_R$ and $\llbracket - \rrbracket : \mathbf{MW}_{\forall} \rightarrow \mathbf{FdM}_R$ on objects as $\llbracket (\mathfrak{A}, \ell_{\mathfrak{A}}) \rrbracket_a := \mathcal{M}_{(\mathfrak{A}, \ell_{\mathfrak{A}}) \cap a}$ and $\llbracket \mathfrak{A} \rrbracket := \bigoplus_{\mathfrak{E} \in \mathfrak{A}^\bullet} \mathcal{M}_{\mathfrak{E}}$.

Then, for the morphisms, we proceed by compositionality for $\llbracket - \rrbracket_a$, meaning that we define $\llbracket - \rrbracket_a$ on every generator and compute the semantics of a diagram by decomposing it

$$\begin{aligned}
 \llbracket \text{w} \times \text{v} \rrbracket_a &= \begin{cases} \text{Id} & \in \mathbf{FdM}_R(\mathcal{M}_A, \mathcal{M}_A) & \text{if } a \in w \setminus v \\ \text{Id} & \in \mathbf{FdM}_R(\mathcal{M}_B, \mathcal{M}_B) & \text{if } a \in v \setminus w \\ h \otimes h' \mapsto h' \otimes h & \in \mathbf{FdM}_R(\mathcal{M}_{A \otimes B}, \mathcal{M}_{B \otimes A}) & \text{if } a \in w \cap v \\ (1) & \in \mathbf{FdM}_R(R, R) & \text{otherwise} \end{cases} \\
 \llbracket \text{w} \cup \text{v} \rrbracket_a &= \begin{cases} h \otimes h' \mapsto \langle h|h' \rangle & \in \mathbf{FdM}_R(\mathcal{M}_{A \otimes A}, R) & \text{if } a \in w \\ (1) & \in \mathbf{FdM}_R(R, R) & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{and the (conjugate)} \\ \text{transposed operator} \\ \text{for the Cap} \end{array} \\
 \llbracket \text{w} \mid \text{v} \rrbracket_a &= \begin{cases} s \cdot \text{Id} & \in \mathbf{FdM}_R(\mathcal{M}_A, \mathcal{M}_A) & \text{if } a \in w \\ (1) & \in \mathbf{FdM}_R(R, R) & \text{otherwise} \end{cases} \\
 \llbracket \text{w} \oplus \text{v} \rrbracket_a &= \begin{cases} \begin{pmatrix} \text{Id} \\ 0 \end{pmatrix} & \in \mathbf{FdM}_R(\mathcal{M}_{A \oplus B}, \mathcal{M}_A) & \text{if } a \in w \\ \begin{pmatrix} 0 \\ \text{Id} \end{pmatrix} & \in \mathbf{FdM}_R(\mathcal{M}_{A \oplus B}, \mathcal{M}_B) & \text{if } a \in v \\ (1) & \in \mathbf{FdM}_R(R, R) & \text{otherwise} \end{cases} \quad \begin{array}{l} \text{and the (conjugate)} \\ \text{transposed operator} \\ \text{for the mirrored} \\ \text{Plus} \end{array}
 \end{aligned}$$

■ **Figure 10** Semantics of the Generators of \mathbf{MW}_W in a World $a \in W$.

with $\llbracket g \circ f \rrbracket_a := \llbracket g \rrbracket_a \circ \llbracket f \rrbracket_a$ and $\llbracket f \square g \rrbracket_a := \llbracket f \rrbracket_a \otimes \llbracket g \rrbracket_a$. The semantics of most generators is given in Figure 10; for all the other generators their semantics is simply the identity operator.

The worlds-agnostic semantics is defined from the world-dependent semantics, as follows. Consider $f \in \mathbf{MW}_W((\mathfrak{A}, \ell_{\mathfrak{A}}), (\mathfrak{B}, \ell_{\mathfrak{B}}))$, we define:

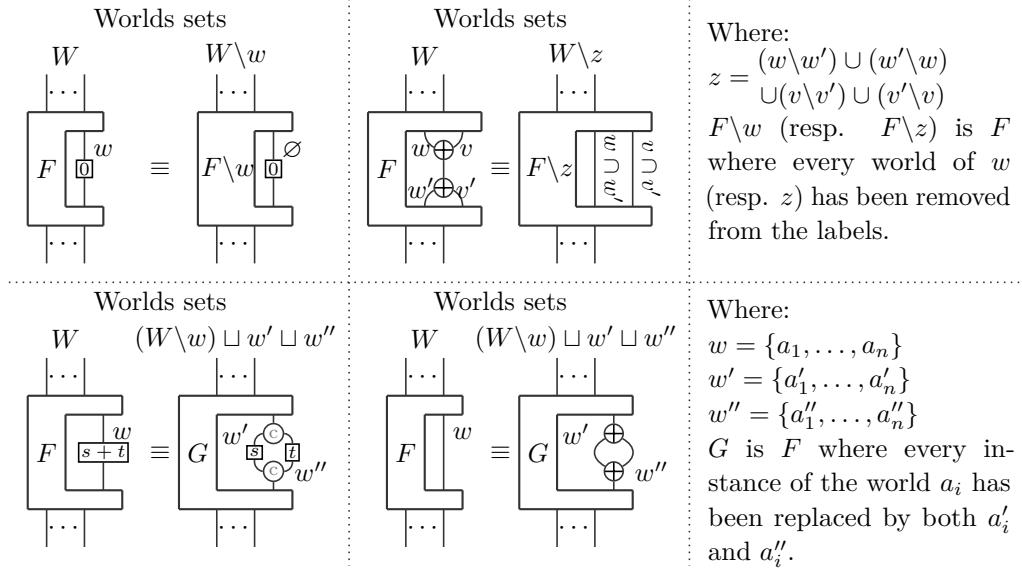
$$\llbracket f \rrbracket_W := \left\{ \sum_{\substack{a \in W \\ (\mathfrak{A}, \ell_{\mathfrak{A}}) \cap a = \mathfrak{A}' \\ (\mathfrak{B}, \ell_{\mathfrak{B}}) \cap a = \mathfrak{B}'}} \llbracket f \rrbracket_a \right\}_{\mathfrak{A}' \in \mathfrak{A}^*, \mathfrak{B}' \in \mathfrak{B}^*} \in \mathbf{FdM}_R \left(\bigoplus_{\mathfrak{C} \in \mathfrak{A}^*} \mathcal{M}_{\mathfrak{C}}, \bigoplus_{\mathfrak{J} \in \mathfrak{B}^*} \mathcal{M}_{\mathfrak{J}} \right)$$

For example, the worlds-agnostic semantics of the Tensor and the Plus are simply the collection of all their world-dependent semantics assembled into a single linear operator:

$$\begin{aligned}
 \left[\begin{array}{c} \text{World set: } \{a, \star\} \\ A : \{a\} \cup B : \{a\} \\ \otimes \\ A \otimes B : \{a\} \end{array} \right] &= \begin{matrix} A \square B & A \square \bullet & \bullet \square B & \bullet \square \bullet \\ \bullet & \begin{pmatrix} \text{Id} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix} \\
 \left[\begin{array}{c} \text{World set: } \{a, b, \star\} \\ A : \{a\} \cup B : \{b\} \\ \oplus \\ A \oplus B : \{a, b\} \end{array} \right] &= \begin{matrix} A \square B & A \square \bullet & \bullet \square B & \bullet \square \bullet \\ \bullet & \begin{pmatrix} 0 & \text{Id} & 0 & 0 \\ 0 & 0 & \text{Id} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{matrix}
 \end{aligned}$$

The worlds-agnostic semantics is functorial (see Appendix C) and is universal in the following sense:

► **Theorem 9** (Universality). *For every $\mathfrak{A}, \mathfrak{B}$ objects of \mathbf{C}_D , and for every linear operator in $U \in \mathbf{FdM}_R \left(\bigoplus_{\mathfrak{C} \in \mathfrak{A}^*} \mathcal{H}_{\mathfrak{C}}, \bigoplus_{\mathfrak{J} \in \mathfrak{B}^*} \mathcal{H}_{\mathfrak{J}} \right)$, there exists a world set W , and a morphism $\llbracket f \rrbracket_W \in \mathbf{MW}_W(\mathfrak{A}, \mathfrak{B})$ such that $\llbracket f \rrbracket = U$.* ◀



■ **Figure 12** Equations with Side-Effects on World Sets

$[f]_W \equiv [g]_V$ we have $\llbracket [f]_W \rrbracket = \llbracket [g]_V \rrbracket$. Additionally if $W = V$, whenever $f \equiv_W g$ we have $\forall a \in W, \llbracket f \rrbracket_a = \llbracket g \rrbracket_a$.

► **Theorem 11** (Completeness). For every $[f]_W : \mathfrak{A} \rightarrow \mathfrak{B}$ and $[g]_V : \mathfrak{A} \rightarrow \mathfrak{B}$, $\llbracket [f]_W \rrbracket = \llbracket [g]_V \rrbracket \iff [f]_W \equiv [g]_V$ ◀

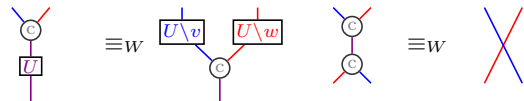
The soundness can be proved by a case-by-case analysis on every equation. The completeness theorem follows from the existence of a normal form for \equiv , as described in Appendix E.1.

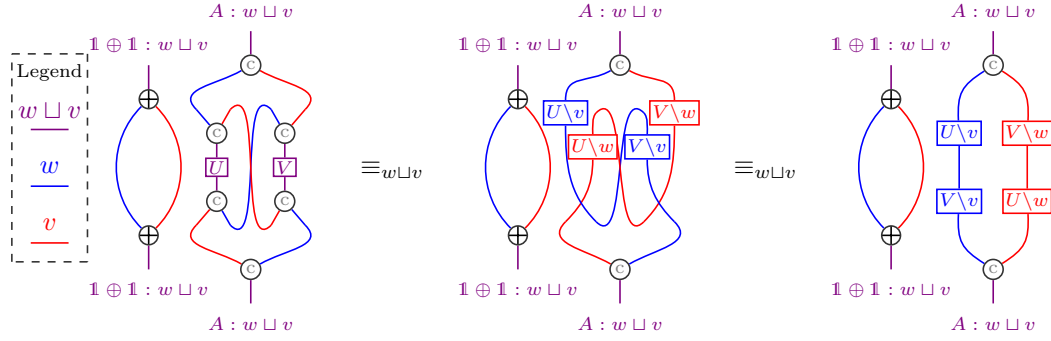
► **Example 12** (The Quantum Switch). Similarly to the "controlled not" in Example 3 where a quantum bit controls whether the identity or a negation is applied to some data, one can consider the case where a quantum bit control whether $U \circ V$ or $V \circ U$ is applied to some data (for U and V two quantum operators on the same type A).

On the rightmost part of Figure 13, one can see the most direct implementation of this "quantum switch", but this implementation uses two copies of U and V . On the leftmost part of the figure, we show another implementation which only relies on one copy of each, and both can be rewritten into another using the equational theory.

In those diagrams, the world set is $W = w \sqcup v$ and we rely on violet, blue and red wires to indicate respectively worlds labels $w \sqcup v$, w and v . Each figure has a control side which operates on a quantum bit (type $1 \oplus 1$) and binds the world w to $|0\rangle$ and the world v to $|1\rangle$; and a computational side which operates on some data of an arbitrary type A , on which could be applied U and/or V which stand for two morphisms of $\mathbf{MW}_W(A : W, A : W)$.

The first rewriting step relies on the two lemmas on the right, both of which being deducible from the equational theory (see Appendix B). The second rewriting step is simply using the properties of a compact closed category.





■ **Figure 13** Rewriting the Quantum Switch

5 Conclusion

We introduced a new sound and complete graphical language based on compact categories with biproducts, along with an equational theory and a worlds system, helping us to build a denotational semantics of our language.

This language is a first step towards the unification of languages based on the tensor \otimes and those based on the biproduct \oplus . This allows us to reason about both systems in parallel, and superposition of executions, as shown by the encoding of the Quantum Switch in Example 12 and the translation from term of the language in Section 2.3.

Following this translation, a natural development of the Many-Worlds calculus consists in accomodating function abstraction and recursion in the language. The question of a complete equational theory for the language on mixed states (e.g. via the discard construction [6]) is also left open.

Finally, while our language allows for quantum control, it does not entirely capture another language that aims at formalizing quantum control, namely the PBS-Calculus [9]. How and in which context could we capture the PBS-Calculus is left for future work.

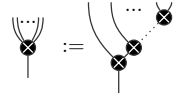
References

- 1 Thorsten Altenkirch and Jonathan Grattage. A functional quantum programming language. In Prakash Panangaden, editor, *Proceedings of the 20th Symposium on Logic in Computer Science, LICS'05*, pages 249–258. IEEE, IEEE Computer Society Press, 2005. doi:10.1109/LICS.2005.1.
- 2 Miriam Backens and Aleks Kissinger. ZH: A complete graphical calculus for quantum computations involving classical non-linearity. In Peter Selinger and Giulio Chiribella, editors, *Proceedings of the 15th International Conference on Quantum Physics and Logic, Halifax, Canada, 3-7th June 2018*, volume 287 of *Electronic Proceedings in Theoretical Computer Science*, pages 23–42, 2019. doi:10.4204/EPTCS.287.2.
- 3 M. Bartha. A finite axiomatization of flowchart schemes. *Acta Inf.*, 8(2):203–217, jun 1977.
- 4 Titouan Carette. *Wielding the ZX-calculus, Flexsymmetry, Mixed States, and Scalable Notations. (Manier le ZX-calcul, flexsymétrie, systèmes ouverts et limandes)*. PhD thesis, University of Lorraine, Nancy, France, 2021. URL: <https://tel.archives-ouvertes.fr/tel-03468027>.
- 5 Titouan Carette, Dominic Horsman, and Simon Perdrix. SZX-Calculus: Scalable Graphical Quantum Reasoning. In Peter Rossmanith, Pinar Heggernes, and Joost-Pieter Katoen, editors, *44th International Symposium on Mathematical Foundations of Computer Science (MFCS 2019)*, volume 138 of *Leibniz International Proceedings in Informatics (LIPIcs)*,

- pages 55:1–55:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: <http://drops.dagstuhl.de/opus/volltexte/2019/10999>, doi:10.4230/LIPIcs.MFCS.2019.55.
- 6 Titouan Carette, Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. Completeness of Graphical Languages for Mixed States Quantum Mechanics. In Christel Baier, Ioannis Chatzigiannakis, Paola Flocchini, and Stefano Leonardi, editors, *46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)*, volume 132 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 108:1–108:15, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: <http://drops.dagstuhl.de/opus/volltexte/2019/10684>, doi:10.4230/LIPIcs.ICALP.2019.108.
- 7 Christophe Chareton, Sébastien Bardin, François Bobot, Valentin Perrelle, and Benoît Valiron. A Deductive Verification Framework for Circuit-building Quantum Programs. [arXiv:2003.05841](https://arxiv.org/abs/2003.05841). To appear in *Proceedings of ESOP’21*.
- 8 G. Chiribella, G. M. D’Ariano, and P. Perinotti. Transforming quantum operations: Quantum supermaps. *EPL (Europhysics Letters)*, 83(3):30004, 2008. doi:10.1209/0295-5075/83/30004.
- 9 Alexandre Clément and Simon Perdrix. Pbs-calculus: A graphical language for coherent control of quantum computations. *arXiv preprint arXiv:2002.09387*, 2020.
- 10 Alexandre Clément, Nicolas Heurtel, Shane Mansfield, Simon Perdrix, and Benoît Valiron. Lov-calculus: A graphical language for linear optical quantum circuits, 2022. URL: <https://arxiv.org/abs/2204.11787>, doi:10.48550/ARXIV.2204.11787.
- 11 Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.
- 12 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.
- 13 Cole Comfort, Antonin Delpeuch, and Jules Hedges. Sheet diagrams for bimonoidal categories, 2020. URL: <https://arxiv.org/abs/2010.13361>, doi:10.48550/ARXIV.2010.13361.
- 14 Dal Lago, Ugo and Faggian, Claudia and Valiron, Benoît and Yoshimizu, Akira. The geometry of parallelism: Classical, probabilistic, and quantum effects. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, page 833–845, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3009837.3009859.
- 15 Giovanni de Felice and Bob Coecke. Quantum linear optics via string diagrams, 2022. URL: <https://arxiv.org/abs/2204.12985>, doi:10.48550/ARXIV.2204.12985.
- 16 Ross Duncan. Generalized proof-nets for compact categories with biproducts. In Gay and Mackie [18], chapter 3, pages 70–134.
- 17 Richard P. Feynman and A. R. Hibbs. *Quantum Mechanics and Path Integrals*. McGraw-Hill Publishing Company, 1965.
- 18 Simon Gay and Ian Mackie, editors. *Semantic Techniques in Quantum Computation*. Cambridge University Press, 2009.
- 19 Jean-Yves Girard. Proof-nets: the parallel syntax for proof-theory. *Lecture Notes in Pure and Applied Mathematics*, pages 97–124, 1996.
- 20 Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: A scalable quantum programming language. In Hans-Juergen Boehm and Cormac Flanagan, editors, *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI’13*, pages 333–342. ACM, 2013. doi:10.1145/2491956.2462177.
- 21 Philip Hackney and Marcy Robertson. On the category of props. *Appl. Categorical Struct.*, 23(4):543–573, 2015. doi:10.1007/s10485-014-9369-4.

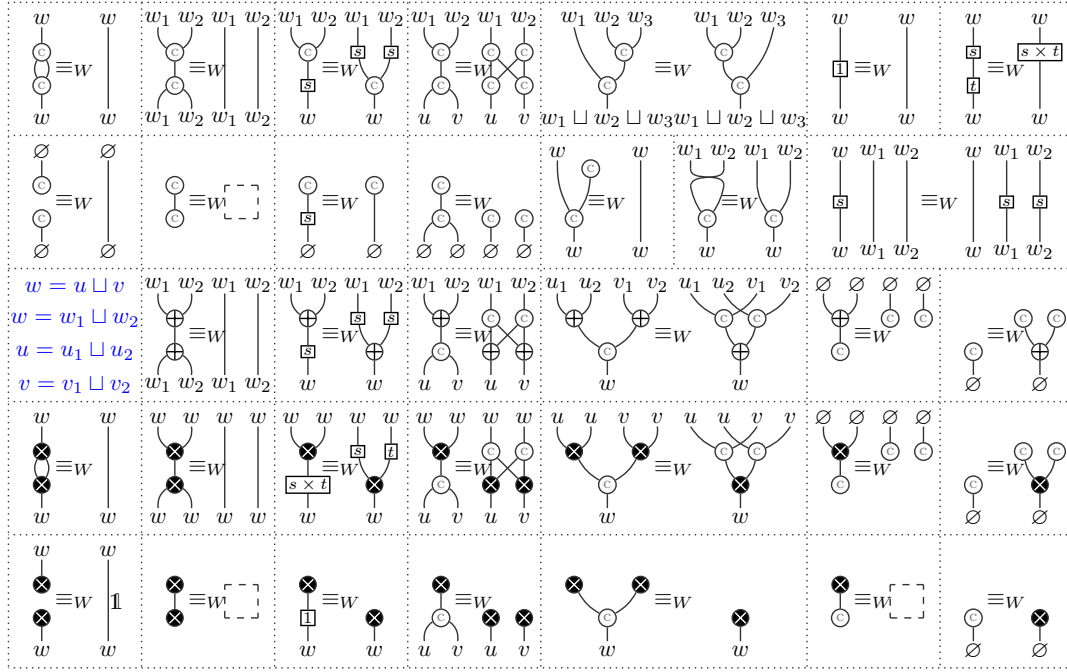
- 22 Amar Hadzihasanovic. A diagrammatic axiomatisation for qubit entanglement. In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 573–584, Jul 2015. doi:10.1109/LICS.2015.59.
- 23 Chris Heunen and Jamie Vicary. *Categories for Quantum Theory*. Oxford University Press, nov 2019. URL: <https://doi.org/10.1093%2Foso%2F9780198739623.001.0001>, doi:10.1093/oso/9780198739623.001.0001.
- 24 Stephen Lack. Composing PROPs. In *Theory and Applications of Categories*, volume 13, pages 147–163, 2004. URL: <http://www.tac.mta.ca/tac/volumes/13/9/13-09abs.html>.
- 25 Yves Lafont. Interaction nets. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '90*, page 95–108, New York, NY, USA, 1989. Association for Computing Machinery. doi:10.1145/96709.96718.
- 26 Paul-André Mellies. Local states in string diagrams. In *Rewriting and Typed Lambda Calculi*, pages 334–348. Springer, 2014.
- 27 Jennifer Paykin, Robert Rand, and Steve Zdancewic. QWIRE: a core language for quantum circuits. In Giuseppe Castagna and Andrew D. Gordon, editors, *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL'17*, pages 846–858. ACM, 2017. doi:10.1145/3009837.3009894.
- 28 Christopher Portmann, Christian Matt, Ueli Maurer, Renato Renner, and Björn Tackmann. Causal boxes: Quantum information-processing systems closed under composition. *IEEE Transactions on Information Theory*, 63(5):3277–3305, 2017. doi:10.1109/TIT.2017.2676805.
- 29 Amr Sabry, Benoît Valiron, and Juliana Kaizer Vizzotto. From symmetric pattern-matching to quantum control. In Christel Baier and Ugo Dal Lago, editors, *Proceedings of the 21st International Conference on Foundations of Software Science and Computation Structures, FoSSaCS 2018*, volume 10803 of *Lecture Notes in Computer Science*, pages 348–364. Springer, 2018. doi:10.1007/978-3-319-89366-2_19.
- 30 Sam Staton. Algebraic effects, linearity, and quantum programming languages. In Sriram K. Rajamani and David Walker, editors, *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL'15*, pages 395–406. ACM, 2015. doi:10.1145/2676726.2676999.
- 31 Augustin Vanrietvelde, Hlér Kristjánsson, and Jonathan Barrett. Routed quantum circuits. *Quantum*, 5:503, 2021. doi:10.22331/q-2021-07-13-503.
- 32 Julian Wechs, Hippolyte Dourdent, Alastair A. Abbott, and Cyril Branciard. Quantum circuits with classical versus quantum control of causal order. *PRX Quantum*, 2:030335, 2021. doi:10.1103/PRXQuantum.2.030335.

A Encoding the Iso Language with the Many Worlds Calculus

For simplicity of the encoding, we give the following syntactic sugar: 

$$\begin{aligned}
 \overline{(|x : A \vdash x : A|)} &= |^A & \overline{(|\vdash \langle \rangle : \mathbb{1}|)} &= \otimes \mathbb{1} & \frac{\xi}{\overline{(\Delta \vdash t : A)}} &= \frac{\Delta}{\overline{(\xi)}} \\
 \frac{\xi}{\overline{(\Delta \vdash t : A)}} &= \frac{\Delta}{\overline{(\xi)}} & \frac{\xi}{\overline{(\Delta \vdash \text{inj}_r t : A \oplus B)}} &= \frac{\Delta}{\overline{(\xi)}} \oplus & \frac{\xi}{\overline{(\Delta \vdash \text{inj}_l t : A \oplus B)}} &= \frac{\Delta}{\overline{(\xi)}} \oplus \\
 \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} & \frac{\xi_1}{\overline{(\Delta \vdash t_1 : A)}} \quad \frac{\xi_2}{\overline{(\Delta \vdash t_2 : A)}} &= \frac{\Delta}{\overline{(\xi_1)}} \oplus \frac{\Delta}{\overline{(\xi_2)}} \\
 \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} & \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} \\
 \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} & \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} \\
 \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} & \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} \\
 \frac{\xi_i}{\overline{(\Delta_i \vdash v_i : A)}} \quad \frac{\xi'_i}{\overline{(\Delta_i \vdash e_i : B)}} &= \frac{\Delta_i}{\overline{(\xi_i)}} \otimes \frac{\Delta_i}{\overline{(\xi'_i)}} & \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} \\
 \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}} & \frac{\xi_1}{\overline{(\Delta_1 \vdash t_1 : A_1 \otimes \dots \otimes A_n)}} \quad \frac{\xi_2}{\overline{(\Delta_2 \vdash t_2 : B)}} &= \frac{\Delta_1}{\overline{(\xi_1)}} \otimes \frac{\Delta_2}{\overline{(\xi_2)}}
 \end{aligned}$$

Figure 14 Inductive translation of terms from Section 2.3 into Many-Worlds diagrams.



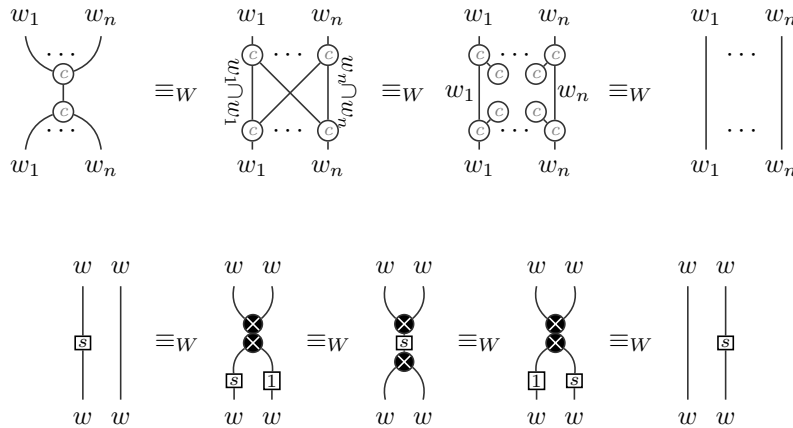
■ **Figure 15** Alternative Presentation of the Equational Theory with a Fixed World Set W

B Induced Equations

In Figure 11, we presented a set of equation reasonably small, by having equations parameterized by the arity of the contraction, and by omitting a lot of useful equations that can be deduced from them. In Figure 15, we take the opposite approach: we give equations using the contractions of arity zero and two (which are sufficient to generate all the other contractions) and we provide additional axioms that follows from Lemma 13 and Lemma 14.

Note that this is only an alternative presentation to the equational theory for \equiv_W , the axioms of Figure 12 are still required for \equiv .

► **Lemma 13.** *Whenever w_i are disjoint sets of worlds, we have the following:*



$$\begin{array}{c} \otimes \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n w_i \equiv_W \begin{array}{c} \otimes \\ | \\ \square \\ | \\ c \\ | \\ \dots \\ | \\ c \\ | \\ \otimes \end{array} \begin{array}{c} w_1 \\ \vdots \\ w_n \end{array} \equiv_W \begin{array}{c} \otimes \\ | \\ \square \\ | \\ \otimes \end{array} w_1 \dots \begin{array}{c} \otimes \\ | \\ \square \\ | \\ \otimes \end{array} w_n$$

► **Lemma 14.** *Whenever w_i are disjoint sets of worlds, and that the v_i s are disjoint sets of worlds too, we have the following:*

$$\begin{array}{c} \otimes \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n w_i \equiv_W \begin{array}{c} \otimes \dots \otimes \\ | \\ c \end{array} \sqcup^n w_i \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \otimes \end{array} \sqcup^n w_i \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \otimes \end{array} \sqcup^n w_i \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i) \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i)$$

Proof. We provide a proof for the third equation, the first two are proven similarly.

$$\begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \oplus \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i) \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \oplus \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i) \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \oplus \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i) \equiv_W \begin{array}{c} w_1 \dots w_n \\ | \\ c \\ | \\ \oplus \\ | \\ \square \\ | \\ \otimes \end{array} \sqcup^n (w_i \sqcup v_i)$$

► **Corollary 15** (Naturality of the Binary Contraction). *For every $f : \square^n(A_i : w_i) \rightarrow \square^m(B_j : v_j)$ with world set W and every $u \subseteq W$, we have*

$$\begin{array}{c} w_1 \setminus u \dots w_n \setminus u \\ | \\ \dots \\ | \\ c \\ | \\ \dots \\ | \\ c \\ | \\ \dots \\ | \\ f \\ | \\ \dots \\ | \\ v_1 \dots v_m \end{array} \equiv_W \begin{array}{c} w_1 \setminus u \dots w_n \setminus u \\ | \\ \dots \\ | \\ f \setminus u \\ | \\ \dots \\ | \\ c \\ | \\ \dots \\ | \\ c \\ | \\ \dots \\ | \\ v_1 \dots v_m \end{array}$$

where $f \setminus u : \square^n(A_i : w_i \setminus u) \rightarrow \square^m(B_j : v_j \setminus u)$ is equal to f where every worlds label w has been replaced by $w \setminus u$, and similarly for $f \cap u$.

This is simply proven by induction over f . All the generator cases (including Cup and Cap) follow directly from the equations given in Figure 11 and Lemma 14 together with the properties of a compact close category.

► **Lemma 16** (Empty World). *For every $f : \square^n(A_i : \emptyset) \rightarrow \square^m(B_j : \emptyset)$ with world set W but such that every worlds label of f is \emptyset , we have*

$$\begin{array}{c} \emptyset \dots \emptyset \\ | \\ \dots \\ | \\ f \\ | \\ \dots \\ | \\ \emptyset \dots \emptyset \end{array} \equiv_W \begin{array}{c} \emptyset \dots \emptyset \\ | \\ c \dots c \\ | \\ \emptyset \dots \emptyset \end{array}$$

This is simply proven by replacing every wire by two contractions of arity zero (sixth axiom of Figure 11 with $n = 0$), and then using the naturality of the contraction of arity zero (last two lines of Figure 11 with $n = 0$) to consume every generator.

C Proof of Functoriality of the World-Agnostic Semantics

► **Proposition 17.** *The worlds-agnostic semantics $\llbracket - \rrbracket$ defined in Section 3 is a monoidal functor from \mathbf{MW}_\forall to \mathbf{FdM}_R .*

Proof. We recall here the definition of the worlds-agnostic semantics of $[f]_W : \mathfrak{A} \rightarrow \mathfrak{B}$:

$$\llbracket [f]_W \rrbracket := \left\{ \sum_{\substack{a \in W \\ (\mathfrak{A}, \ell_{\mathfrak{A}}^f) \cap a = \mathfrak{A}' \\ (\mathfrak{B}, \ell_{\mathfrak{B}}^f) \cap a = \mathfrak{B}'}} \llbracket f \rrbracket_a \right\}_{\mathfrak{A}' \in \mathfrak{A}^*, \mathfrak{B}' \in \mathfrak{B}^*}$$

From the definition of the worlds-agnostic compositions, we directly have:

$$\llbracket [f]_W \sqcap [g]_V \rrbracket_{(a,b)} = \llbracket [f]_W \rrbracket_a \otimes \llbracket [g]_V \rrbracket_b \quad \llbracket [g]_V \circ [f]_W \rrbracket_{(a,b)} = \llbracket [g]_V \rrbracket_b \circ \llbracket [f]_W \rrbracket_a$$

The functoriality with respect to the parallel composition is then immediate:

$$\llbracket [f]_W \sqcap [g]_V \rrbracket = \left\{ \sum \llbracket [f]_W \sqcap [g]_V \rrbracket_{(a,b)} \right\} = \left\{ \sum \llbracket [f]_W \rrbracket_a \right\} \otimes \left\{ \sum \llbracket [g]_V \rrbracket_b \right\} = \llbracket [f]_W \rrbracket \otimes \llbracket [g]_V \rrbracket$$

The functoriality with respect to the sequential composition is more subtle, as one must carefully manipulate the indices of the sum and remark that the set of worlds w eliminated by the worlds-agnostic composition satisfies the following:

$$(a, b) \notin w \iff (\mathfrak{B}, \ell_{\mathfrak{B}}^f) \cap a = (\mathfrak{B}, \ell_{\mathfrak{B}}^g) \cap b \text{ where } \begin{array}{l} f : (\mathfrak{A}, \ell_{\mathfrak{A}}^f) \rightarrow (\mathfrak{B}, \ell_{\mathfrak{B}}^f) \\ g : (\mathfrak{B}, \ell_{\mathfrak{B}}^g) \rightarrow (\mathfrak{C}, \ell_{\mathfrak{C}}^g) \end{array}$$

Then, we have

$$\llbracket [g]_V \circ [f]_W \rrbracket = \left\{ \sum \llbracket [g]_V \circ [f]_W \rrbracket_{(a,b)} \right\} = \left\{ \sum \llbracket [g]_V \rrbracket_b \right\} \circ \left\{ \sum \llbracket [f]_W \rrbracket_a \right\} = \llbracket [g]_V \rrbracket \circ \llbracket [f]_W \rrbracket$$

◀

D

 Proof of Soundness

Given that most of the time, $\llbracket \text{gen} \rrbracket_a$ is the identity, the equations defining \equiv_W are quite straightforward to verify. We immediately have that \equiv_W is sound with respect to $\llbracket - \rrbracket_a$ for every $a \in W$. Since $\llbracket - \rrbracket$ is defined from $\llbracket - \rrbracket_a$, soundness with respect to $\llbracket - \rrbracket$ is also correct. We then handle the five additional equations of \equiv .

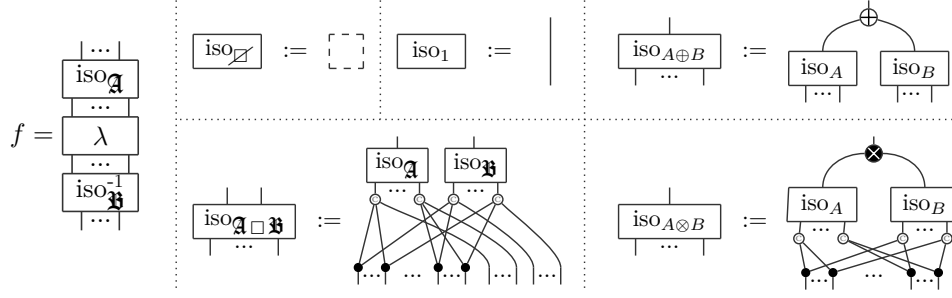
Renaming Applying a bijection to the worlds set W does not change the result computed by $\sum_{a \in W} \dots$, hence this equation is sound with respect to $\llbracket - \rrbracket$.

Annihilation due to Scalars This equation simply removes elements equal to zero from the sum $\sum_{a \in W} \dots$, hence it is sound with respect to $\llbracket - \rrbracket$.

Annihilation due to Plus Since \oplus is a biproduct in \mathbf{FdHilb} , we have $\text{proj}_H^{H \oplus K} \circ \text{inj}_H^{H \oplus K} = \text{id}_H$, $\text{proj}_K^{H \oplus K} \circ \text{inj}_K^{H \oplus K} = \text{id}_K$, $\text{proj}_K^{H \oplus K} \circ \text{inj}_H^{H \oplus K} = 0$ and $\text{proj}_H^{H \oplus K} \circ \text{inj}_K^{H \oplus K} = 0$. One can then simply remove from the $\sum_{a \in W} \dots$ the elements equal to zero, which proves that *Annihilation due to Plus* is sound with respect to $\llbracket - \rrbracket$.

Splitting due to Scalars Since \mathbf{FdHilb} is a vector space, we have $(s + t) \cdot f = s \cdot f + t \cdot f$, which is exactly the property required for this equation to be sound for $\llbracket - \rrbracket$.

Splitting due to Plus Similarly, we have in \mathbf{FdHilb} the property that $\text{id}_{H \oplus K} = \text{inj}_H^{H \oplus K} \circ \text{proj}_H^{H \oplus K} + \text{inj}_K^{H \oplus K} \circ \text{proj}_K^{H \oplus K}$, which is the property required for this equation to be sound for $\llbracket - \rrbracket$.



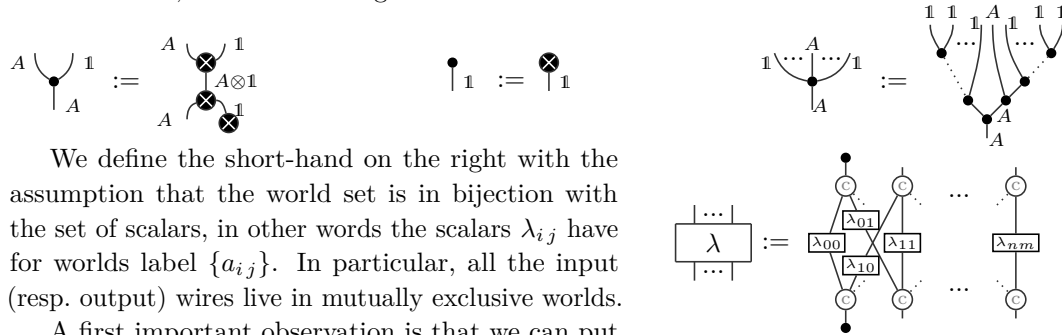
■ **Figure 16** The Normal Form (left) and the Inductive Definition of $\text{iso}_{\mathfrak{a}}$ (right)

E Normal Form, Universality and Completeness

We can prove that the equational theory of Section 4 is complete, by defining a normal form on the morphisms, and by showing that all diagrams can be put in this normal form, which is unique.

E.1 Normal Form

It will be practical for our normal forms to define the following syntactic sugar, which we call the *unitor*, its *unit* and its generalized form:

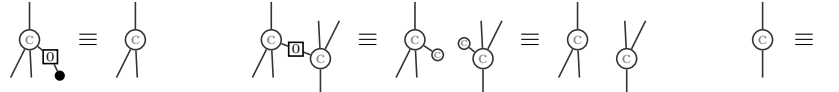


We define the short-hand on the right with the assumption that the world set is in bijection with the set of scalars, in other words the scalars λ_{ij} have for worlds label $\{a_{ij}\}$. In particular, all the input (resp. output) wires live in mutually exclusive worlds.

A first important observation is that we can put this diagram in the following alternative form, with exactly the same assumption on the world set:

$$\boxed{\lambda} = \text{diagram with nodes } \lambda_{00}, \lambda_{01}, \lambda_{10}, \lambda_{11}, \dots, \lambda_{nm} \quad (1)$$

We will freely change between the two representations in the following. Another important observation is that any permutation of wires (all mutually exclusive, and of type $\mathbb{1}$) can easily be put in this form using the following equations:



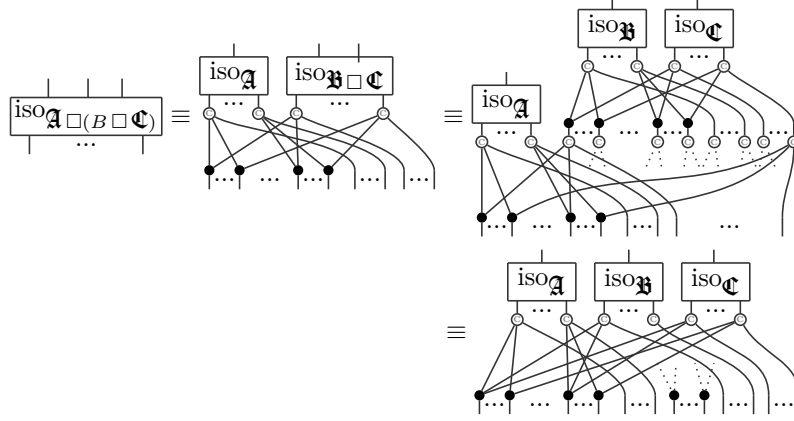
The normal form of a morphism $f : \mathfrak{a} \rightarrow \mathfrak{b}$ is defined as the form of the diagram on the left of Figure 16, where the morphism $\text{iso}_{\mathfrak{a}}$ is defined inductively right.

The output wires of $\text{iso}_{\mathfrak{a}}$ for any \mathfrak{a} live in mutually exclusive worlds, but once again, we don't overload the diagrams with unitors or world names encoding this information, although it will be used in the following.

Notice that graphically, there is no difference between $\mathcal{A} \square (\mathcal{B} \square \mathcal{C})$ and $(\mathcal{A} \square \mathcal{B}) \square \mathcal{C}$, in other words \square is strictly associative. However $\text{iso}_{\mathcal{A} \square (\mathcal{B} \square \mathcal{C})}$ and $\text{iso}_{(\mathcal{A} \square \mathcal{B}) \square \mathcal{C}}$ are different, but they are equivalent up to a rearranging of the output wires:

► **Lemma 18.** *There exists a wire permutation σ such that $\text{iso}_{\mathcal{A} \square (\mathcal{B} \square \mathcal{C})} = \sigma \circ \text{iso}_{(\mathcal{A} \square \mathcal{B}) \square \mathcal{C}}$.* ◀

Proof. First notice that in both $\text{iso}_{\mathcal{A} \square (\mathcal{B} \square \mathcal{C})}$ and $\text{iso}_{(\mathcal{A} \square \mathcal{B}) \square \mathcal{C}}$, we can use the bialgebra between contractions and unitors, followed by their respective fusions in the following way:



and similarly for $\text{iso}_{(\mathcal{A} \square \mathcal{B}) \square \mathcal{C}}$. It then suffices to check which contractions the bottom unitors are linked to. Naming the i th contraction existing iso_A as a_i , and similarly for B and C , we can see that for each triple (a_i, b_j, c_k) there is exactly one unitor connected to precisely contraction a_i , b_j and c_k , in both diagrams. The same is true for every pair (a_i, b_j) , (a_i, c_k) and (b_j, c_k) , as well as for every 1-tuple $(a_i,)$, $(b_j,)$ and $(c_k,)$. This shows that both diagrams are equal up to rearranging of the outputs. ◀

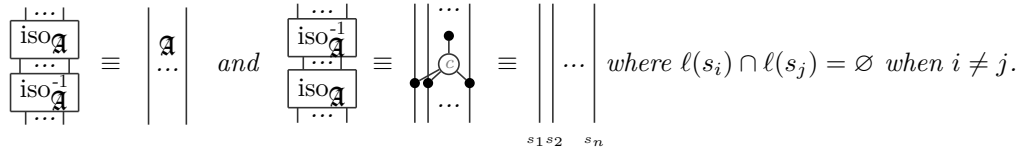
We hence have a choice to make here for canonicity, and choose $\text{iso}_{A_0 \square A_1 \square A_2 \square \dots} := \text{iso}_{(\dots((A_0 \square A_1) \square A_2) \square \dots)}$.

We define iso_A^{-1} inductively in the same way, but upside-down.

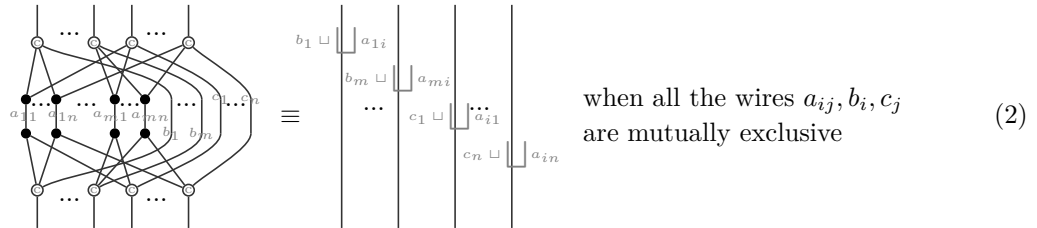
We note that $\text{iso}_A^{-1} \circ \text{iso}_A$ is the normal form of id_A .

► **Lemma 19.** *Notice that $\text{iso}_{A \oplus B} \equiv \text{iso}_A \oplus \text{iso}_B$ and $\text{iso}_{A \otimes B} \equiv \text{iso}_A \otimes \text{iso}_B$.*

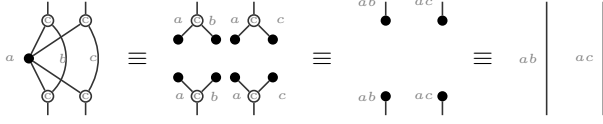
► **Lemma 20.** *We have the following identities:*



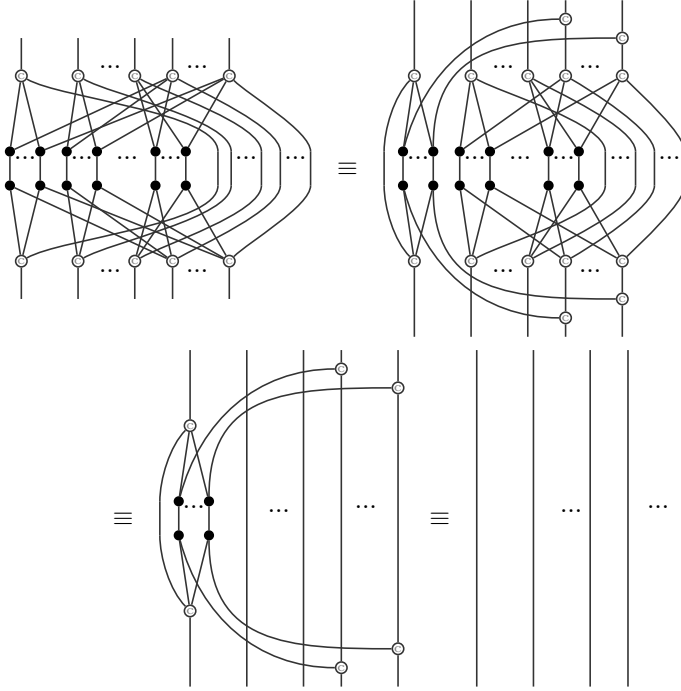
Proof of Lemma 20. We will use the following identities:



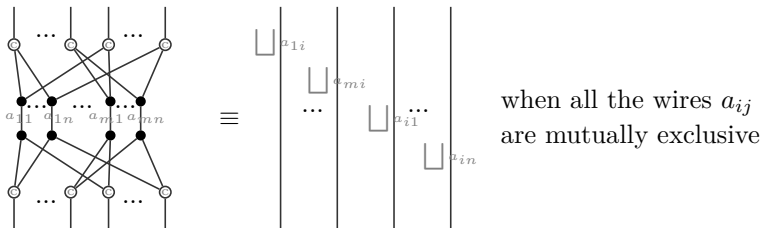
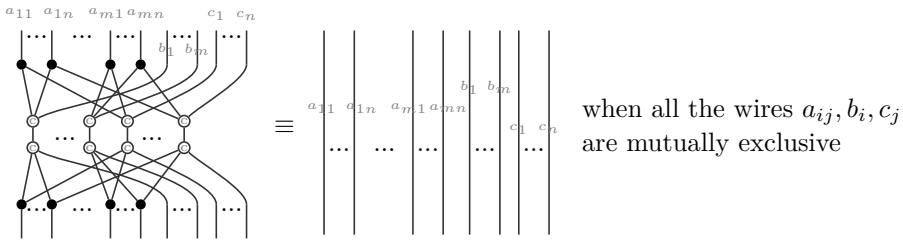
We can show this result by induction on n and m . Case $(0, m)$ is obvious. Case $(1, 1)$ can be proven easily using world sets:

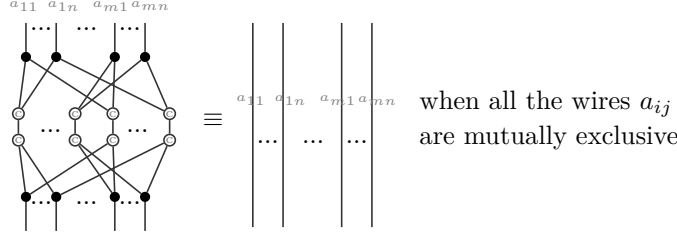


For any n and m , we can then prove the case $(n + 1, m)$ using the cases (n, m) and $(1, m)$ (the case $(n, m + 1)$ is completely symmetric):

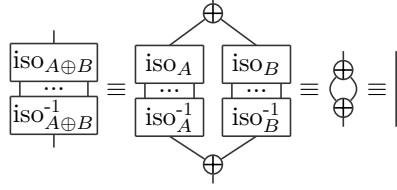


In a similar way, it is possible to show the following three identities:





- $[\text{iso}_{\mathbf{a}}^{-1} \circ \text{iso}_{\mathbf{a}}]$: The result is obvious in cases $\mathbb{1}$ and \mathbb{Z} . For $A \oplus B$:



The proof is similar for \otimes and \square using the previous identities.

- $[\text{iso}_{\mathbf{a}} \circ \text{iso}_{\mathbf{a}}^{-1}]$: The result is again obvious for $\mathbb{1}$ and \mathbb{Z} . The general result is easy to prove by induction using the above identities.

◀

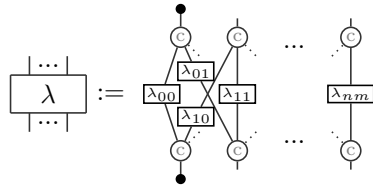
E.2 Universality

Now that we have defined a normal form, we may use it to show the universality (Theorem 9) of the language, i.e. that for every \mathbf{a}, \mathbf{b} objects of $\mathbf{C}_{\mathbf{D}}$, and for every linear operator

$$\Lambda = \begin{pmatrix} \lambda_{11} & \cdots & \lambda_{n1} & \lambda_{10} \\ \vdots & & \vdots & \vdots \\ \lambda_{1m} & \cdots & \lambda_{nm} & \lambda_{m0} \\ \lambda_{01} & \cdots & \lambda_{0n} & \lambda_{00} \end{pmatrix} \in \text{FdM}_R \left(\bigoplus_{\mathbf{c} \in \mathbf{a}^\bullet} \mathcal{H}_{\mathbf{c}}, \bigoplus_{\mathbf{f} \in \mathbf{b}^\bullet} \mathcal{H}_{\mathbf{f}} \right)$$

then the morphism $f = \text{iso}_{\mathbf{b}}^{-1} \circ \lambda \circ \text{iso}_{\mathbf{a}}$ defined in Figure 16 satisfies $\llbracket f \rrbracket = \Lambda$.

As stated in the corresponding section, there is one world for each scalar in λ , so we write a_{ij} the world associated to λ_{ij} and W the set of all those worlds. Let us write $\mathbb{1}_0^k = \bullet^k$ for $\bullet \square \dots \square \bullet$ (with k elements) and $\mathbb{1}_i^k$ for \bullet^k where the i -th \bullet has been replaced by $\mathbb{1}$ if $1 \leq i \leq k$. Beware that we consider $\mathbb{1}_0^k$ to be the *last* element of the canonical basis. We then have $\llbracket \lambda \rrbracket_{a_{ij}} : \mathbb{1}_i^n \rightarrow \mathbb{1}_j^m : x \mapsto \lambda_{ij} \cdot x$ where:



Additionally, one can show by induction that $\llbracket \text{iso}_{\mathbf{a}} \rrbracket_{a_{ij}}$ is simply the projection on the i -th element of the canonical basis, and $\llbracket \text{iso}_{\mathbf{b}}^{-1} \rrbracket_{a_{ij}}$ is simply the injection on the j -th element of the canonical basis. Since we have $\llbracket f \rrbracket_a = \llbracket \text{iso}_{\mathbf{b}}^{-1} \rrbracket_a \circ \llbracket \lambda \rrbracket_a \circ \llbracket \text{iso}_{\mathbf{a}} \rrbracket_a$ for every $a \in W$, and $\llbracket f \rrbracket$ being the collection of all the $\llbracket f \rrbracket$, we obtain that $\llbracket f \rrbracket = \Lambda$.

E.3 Uniqueness of the Normal Form

An crucial feature of the normal form for the completeness is its uniqueness:

► **Proposition 21.** *The normal form is unique.*

Proof. Let f and g be two diagrams in normal form (with respectively λ and μ as inner block), such that $\llbracket f \rrbracket = \llbracket g \rrbracket$ (the naming of the worlds is taken to be the same in both diagrams, and is the same as in the previous proof). By the definition of $\llbracket \cdot \rrbracket$, we have $\llbracket f \rrbracket_a = \llbracket g \rrbracket_a$ for every $a \in W$. We hence have $\llbracket \text{iso}_{\mathfrak{B}}^{-1} \rrbracket_a \circ \llbracket \lambda \rrbracket_a \circ \llbracket \text{iso}_{\mathfrak{A}} \rrbracket_a = \llbracket \text{iso}_{\mathfrak{B}}^{-1} \rrbracket_a \circ \llbracket \mu \rrbracket_a \circ \llbracket \text{iso}_{\mathfrak{A}} \rrbracket_a$.

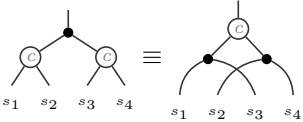
Denoting $e_i^{\mathfrak{A}}$ (resp. $e_i^{\mathfrak{B}}$) the i -th element of the basis of \mathfrak{A} (resp. \mathfrak{B}), we have:

$$\lambda_{ij} = e_j^{\mathfrak{B}^\dagger} \llbracket \text{iso}_{\mathfrak{B}}^{-1} \rrbracket_{a_{ij}} \circ \llbracket \lambda \rrbracket_{a_{ij}} \circ \llbracket \text{iso}_{\mathfrak{A}} \rrbracket_{a_{ij}} e_i^{\mathfrak{A}} = e_j^{\mathfrak{B}^\dagger} \llbracket \text{iso}_{\mathfrak{B}}^{-1} \rrbracket_{a_{ij}} \circ \llbracket \mu \rrbracket_{a_{ij}} \circ \llbracket \text{iso}_{\mathfrak{A}} \rrbracket_{a_{ij}} e_i^{\mathfrak{A}} = \mu_{ij}$$

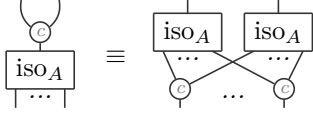
Hence, all coefficients in the scalars of f and g are the same. Since the structure is otherwise the same for f and g , they are the same diagram. ◀

E.4 Completeness

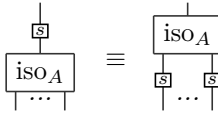
We can now use this normal form to show that our equational theory is complete for arbitrary morphisms. To do so, we need to show that all the generators can be put in normal form, and then that any composition of morphisms in normal form can be put in normal form. To do so we will first derive a few lemmas:

► **Corollary 22** (of Corollary 15).  when $s_1 \cap s_4 = s_2 \cap s_3 = \emptyset$

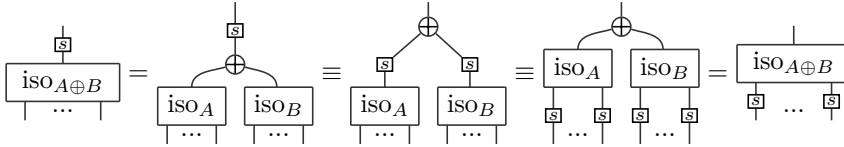
► **Corollary 23** (of Corollary 15). *Single-colored isos distribute over the contraction:*



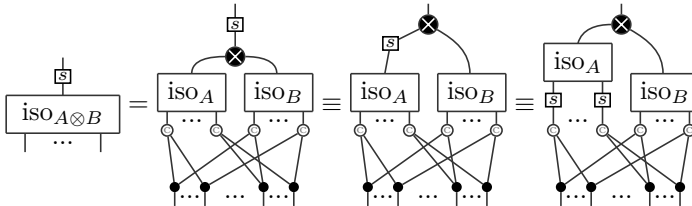
► **Lemma 24.** *Scalars distribute over single-colored isos:*

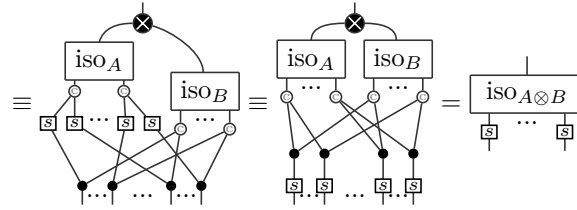


Proof. The result is obvious for $\mathbf{1}$ and ∇ . For $A \oplus B$:

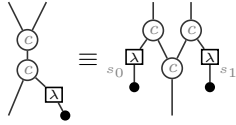


For $A \otimes B$:



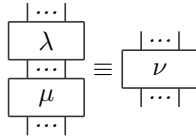


► **Corollary 25** (of Corollary 15).



with $s_0 \cap s_1 = \emptyset$.

► **Lemma 26.**



with $\nu_{ij} = \sum_k \lambda_{ik} \mu_{kj}$.

Proof. Suppose the worlds of λ are the $\{a_{ij}\}_{ij}$ and that of μ are the $\{b_{kl}\}_{kl}$. We count inputs/outputs starting at 1, hence $p \geq 1$ in the following.

The p -th output of λ has world set $\{a_{ip}\}_i$ and the p -th input of μ has world set $\{b_{p\ell}\}_\ell$. When composing the two in sequence, in the first step, each singleton world $\{a_{ij}\}$ (resp. $\{b_{kl}\}$) is mapped to $\{(a_{ij}, b_{kl})\}_{kl}$ (resp. $\{(a_{ij}, b_{kl})\}_{ij}$), and unions of singleton worlds to unions of the worlds each is mapped to.

In particular, the p -th output of λ now has world set $\{(a_{ip}, b_{k\ell})\}_{ik\ell}$, and the p -th input of μ now has world set $\{(a_{ij}, b_{p\ell})\}_{ij\ell}$. After composition, the two sets have to match. They become $\{(a_{ip}, b_{p\ell})\}_{i\ell}$, and all pairs of world that were removed from these two sets are removed globally.

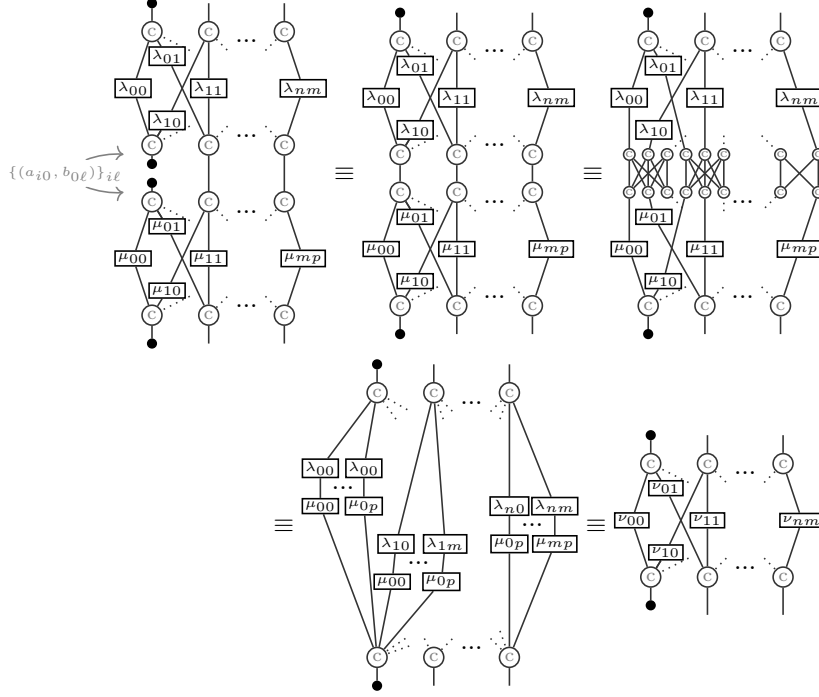
We do so for all $p \geq 1$, which means all worlds in:

$$\{(a_{ij}, b_{k\ell}) \mid j \geq 1, j \neq k\} \cup \{(a_{ij}, b_{k\ell}) \mid k \geq 1, j \neq k\} = \{(a_{ij}, b_{k\ell}) \mid j \neq k\}$$

are removed. Crucially, this means that the world sets $\{a_{i0}\}_i$ and $\{b_{0\ell}\}_\ell$ are mapped to the same world set:

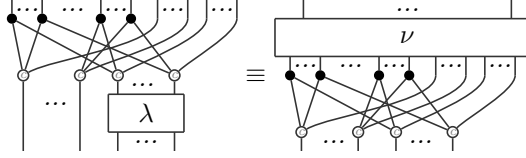
$$\begin{aligned} \{a_{i0}\}_i &\mapsto \{(a_{i0}, b_{k\ell})\}_{ik\ell} \setminus \{(a_{ij}, b_{k\ell}) \mid j \neq k\} \\ &= \{(a_{i0}, b_{0\ell})\}_{i\ell} = \{(a_{ij}, b_{0\ell})\}_{ij\ell} \setminus \{(a_{ij}, b_{k\ell}) \mid j \neq k\} \leftarrow \{b_{0\ell}\}_\ell \end{aligned}$$

Hence, the sequential composition of λ and μ becomes:



with $\nu_{ij} = \sum_k \lambda_{ik} \mu_{kj}$.

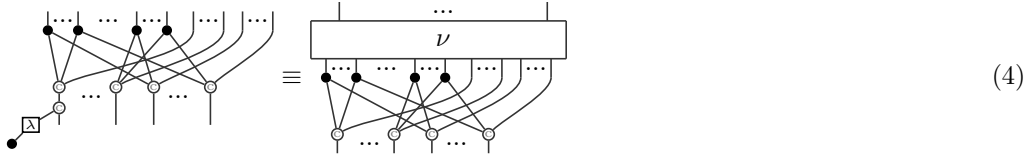
► **Lemma 27.** For any "matrix block" λ , there exists a "matrix block" ν such that:



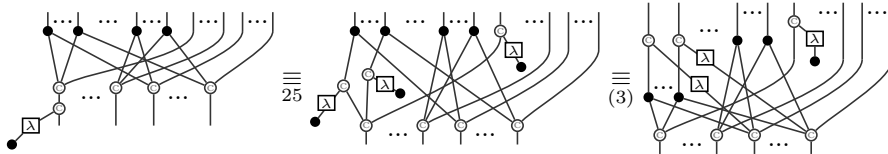
Proof. We show the result through several steps. First we show the following:

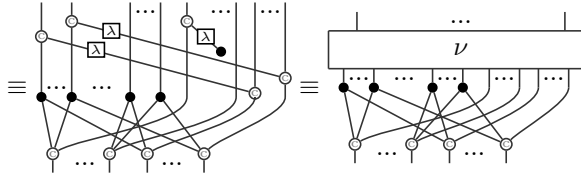


Then, we deal with the top dangling scalars and prove that:



Indeed:



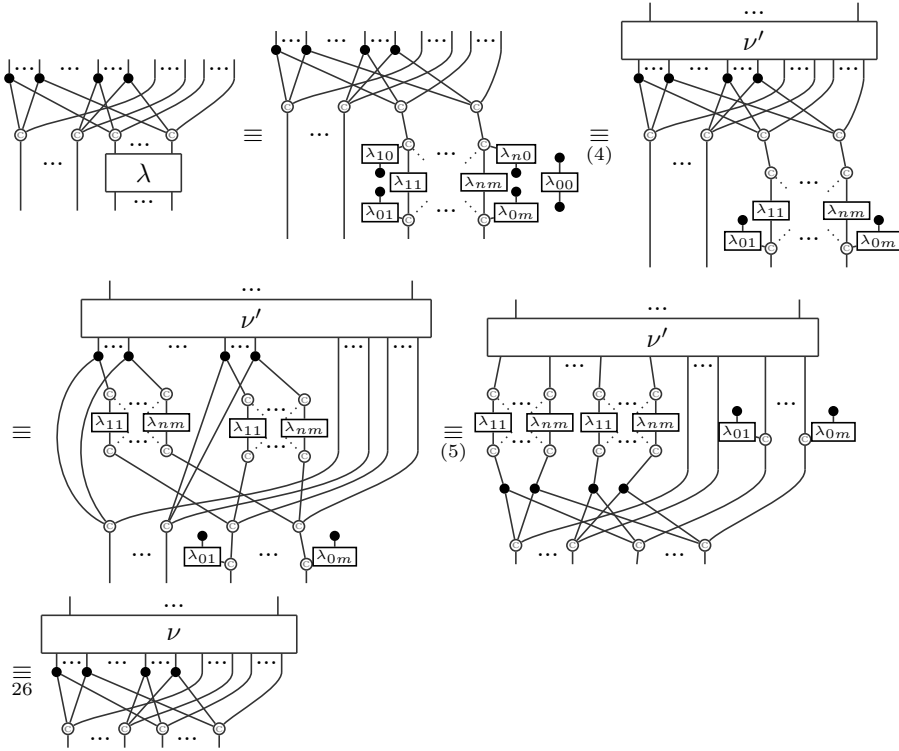


This case generalizes to any bottom wire the dangling scalar is applied on. When we have several of them, we can make them go through the top part in turns, then aggregate them under a single "matrix block" using Lemma 26.

We then derive the following equation:

(5)

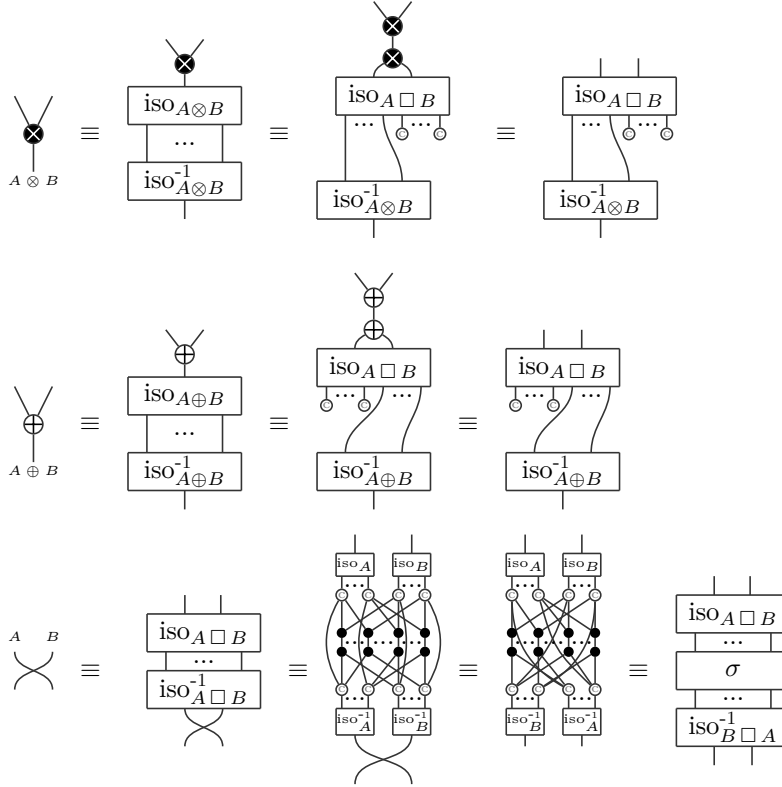
We can finally prove the lemma:



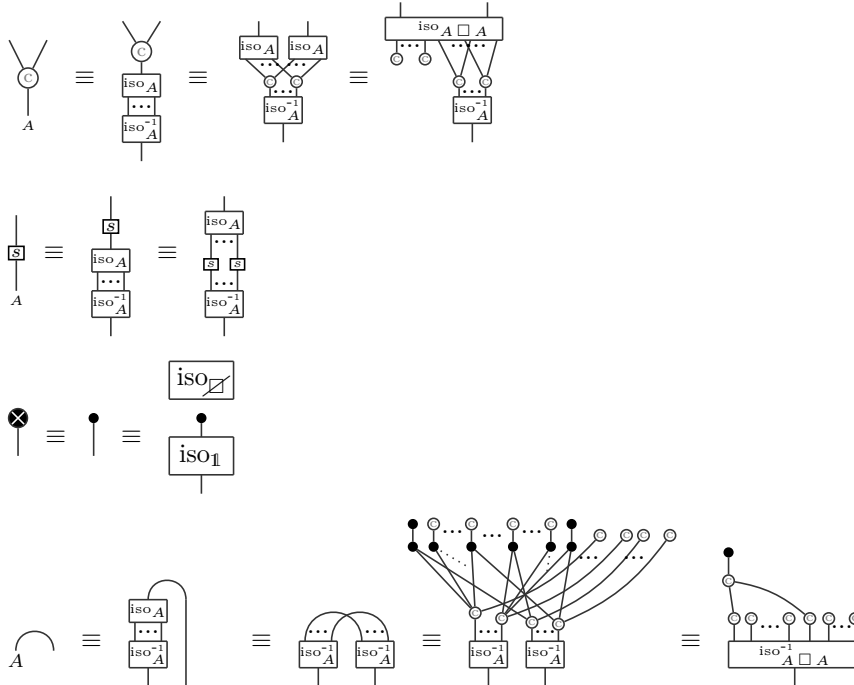
We can now move on to show that generators can be put in normal form:

► **Proposition 28.** *The generators can be put in normal form.*

Proof.



with σ a simple permutation of wires.

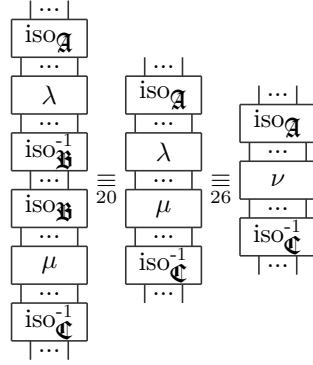


The upside-down versions of the generators are provided in exactly the same way (but upside-down). ◀

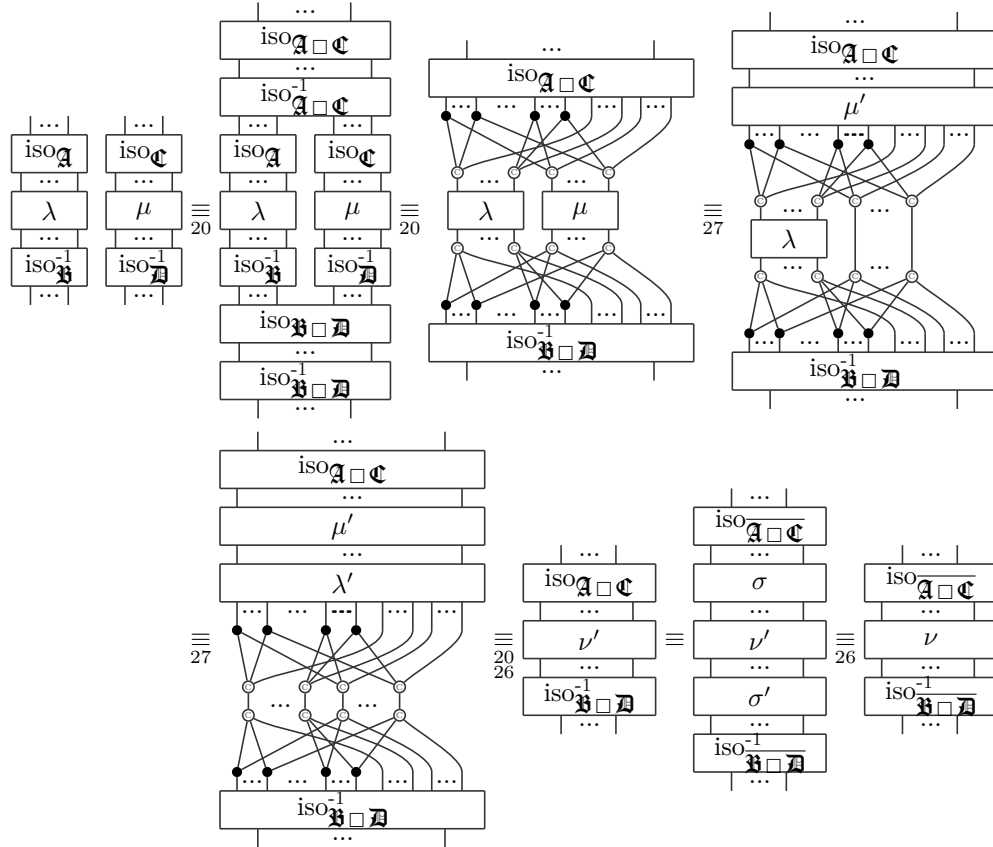
► **Proposition 29.** *Compositions of diagrams in normal form can be put in normal form.*

It is then possible to show that compositions of diagrams in normal form can be put in normal form:

Proof. In the case of sequential composition:



In the case of parallel composition:



where $\overline{a \square c}$ represents the canonical choice of composition with \square (and similarly for $\overline{b \square d}$). ◀

This finally allows us to prove the completeness theorem claimed above:

Proof of Theorem 11. The right-to-left direction of the equivalence can be directly checked by verifying that all the axioms preserve the semantics.

Let f_1 and f_2 be two morphisms such that $\llbracket f_1 \rrbracket = \llbracket f_2 \rrbracket$. Both morphisms can be put in normal form, resp. f_1^{NF} and f_2^{NF} , with $f_i \equiv f_i^{NF}$ and thus $\llbracket f_i^{NF} \rrbracket = \llbracket f_i \rrbracket$. By uniqueness of the normal form, and since $\llbracket f_1^{NF} \rrbracket = \llbracket f_2^{NF} \rrbracket$, we get $f_1^{NF} \equiv f_2^{NF}$, which ends the proof that $f_1 \equiv f_2$. ◀