arXiv:2012.01847v2 [cs.LO] 3 Feb 2022

String Diagram Rewrite Theory I: Rewriting with Frobenius Structure

FILIPPO BONCHI and FABIO GADDUCCI, University of Pisa, Italy
ALEKS KISSINGER, University of Oxford, United Kingdom
PAWEL SOBOCINSKI, Tallinn University of Technology, Estonia
FABIO ZANASI, University College London, United Kingdom

String diagrams are a powerful and intuitive graphical syntax, originating in theoretical physics and later formalised in the context of symmetric monoidal categories. In recent years, they have found application in the modelling of various computational structures, in fields as diverse as Computer Science, Physics, Control Theory, Linguistics, and Biology.

In several of these proposals, transformations of systems are modelled as rewriting rules of diagrams. These developments require a mathematical foundation for string diagram rewriting: whereas rewriting theory for terms is well-understood, the two-dimensional nature of string diagrams poses quite a few additional challenges.

This work systematises and expands a series of recent conference papers, laying down such a foundation. As first step, we focus on the case of rewriting systems for string diagrammatic theories that feature a Frobenius algebra. This common structure provides a more permissive notion of composition than the usual one available in monoidal categories, and has found many applications in areas such as concurrency, quantum theory, and electrical circuits. Notably, this structure provides an exact correspondence between the syntactic notion of string diagrams modulo Frobenius structure and the combinatorial structure of hypergraphs.

Our work introduces a combinatorial interpretation of string diagram rewriting modulo Frobenius structures in terms of double-pushout hypergraph rewriting. We prove this interpretation to be sound and complete and we also show that the approach can be generalised to rewriting modulo multiple Frobenius structures. As a proof of concept, we show how to derive from these results a termination strategy for Interacting Bialgebras, an important rewriting theory in the study of quantum circuits and signal flow graphs.

CCS Concepts: • **Theory of computation → Categorical semantics**; **Equational logic and rewriting**.

Additional Key Words and Phrases: string diagram, double-pushout rewriting, category theory, Frobenius algebra

Reference Format:
Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. String Diagram Rewrite Theory I: Rewriting with Frobenius Structure. CoRR abs/2201.00233 (2022), 57 pages. To appear in Journal of the ACM.

1 INTRODUCTION

This is the first of a series of papers [BGK+20, BGK+21] giving a comprehensive foundation for the rewriting theory of string diagrams. String diagrams are a graphical syntax that is particularly well-suited for capturing the behaviour of structures whose basic operations take many inputs to many outputs. This can be contrasted with term syntax, which is best suited for algebraic structures, whose basic operations take many inputs to a single output. In recent years, structures that mix

Authors' addresses: Filippo Bonchi, filippo.bonchi@unipi.it; Fabio Gadducci, fabio.gadducci@unipi.it, University of Pisa, Pisa, Italy; Aleks Kissinger, University of Oxford, Oxford, United Kingdom, aleks.kissinger@cs.ox.ac.uk; Pawel Sobocinski, Tallinn University of Technology, Tallinn, Estonia, sobocinski@gmail.com; Fabio Zanasi, University College London, London, United Kingdom, f.zanasi@ucl.ac.uk.

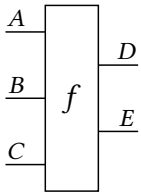
© 2022

algebraic (i.e. many-to-1 operations) and coalgebraic (i.e. 1-to-many) operations have increasingly found applications in a variety of fields, such as concurrency theory, control theory, quantum physics, biology, computational linguistics, and even cognition and consciousness.

Notable features of string diagrams include their flexibility and intuitive aspects. Two diagrams that can be topologically deformed into each other without cutting or joining wires must necessarily describe the same map. This makes string diagrams into a powerful language for reasoning about interacting processes, but also introduces unique challenges when it comes to formalising and implementing string diagrammatic equational reasoning and rewriting theory.

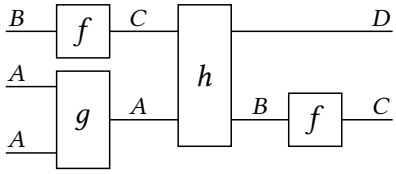
The formal basis for string diagrammatic syntax is the theory of symmetric monoidal categories (SMCs). SMCs provide a minimal setting for reasoning about processes that can compose in parallel or in sequence. String diagrams describe compositions of morphisms in an SMC, where boxes represent the morphisms themselves and wires represent objects, which serve as inputs and outputs

$$f : A \oplus B \oplus C \to D \oplus E \qquad \Longrightarrow$$

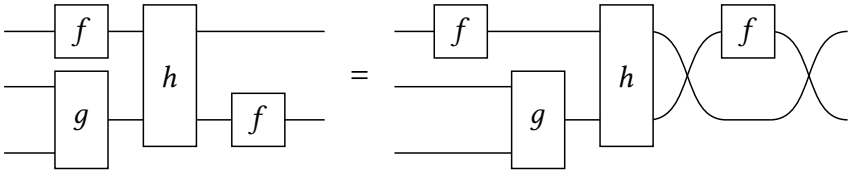


Composition in parallel ("$\oplus$") and in sequence ("$;$") are then depicted diagrammatically. For example

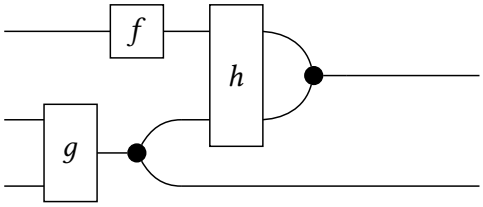
$$(f \oplus g)\,;\, h\,;\, (1 \oplus f) \qquad \Longrightarrow$$



The relevant data of such a composition is the connectivity, rather than the physical layout of the boxes and wires. Hence, it makes sense to consider 'crossing' of wires, and the SMC axioms make the interpretation invariant under deformation, e.g.



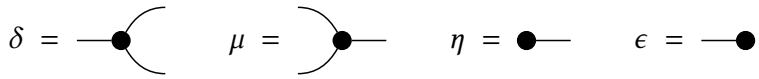
There are many contexts where it furthermore makes sense to consider 'wires' that do not have just two endpoints, but many. Or, put another way, we may wish to allow wires to branch and merge in string diagrams, like in this example



Rather than representing a path that information flows along, a branching wire represents a more general kind of interface between components. For example, branching wires in a digital circuit can be used to represent a physical wire, where junctions are used to join multiple wires into one. Systems whose primitive pieces compose via shared names (e.g. indices in tensor networks, variables in programming languages, or channels in process calculi) also naturally have such splitting structures, unless we specifically impose that names only occur in matched pairs. For non-deterministic, probabilistic, or quantum processes, branching wires represent the appropriate

notion of correlation between their endpoints, e.g. statistical perfect correlations or (GHZ-like) quantum entanglement.

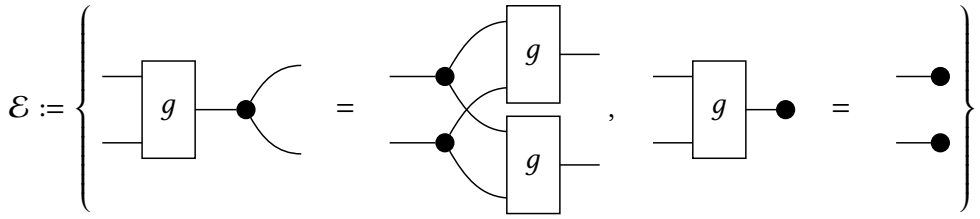
These branching wires can be accommodated categorically by requiring that each object in the category comes equipped with certain basic operations for 'splitting' and 'merging' wires, as well as 'initialising' and 'terminating' them



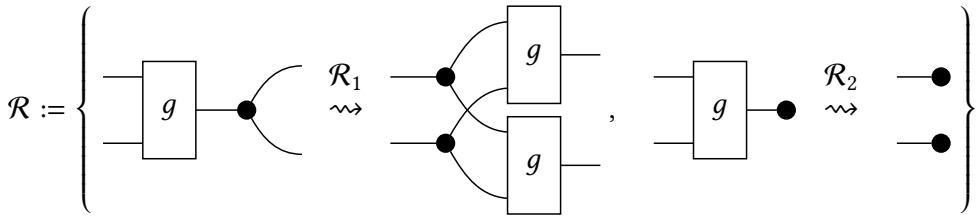
satisfying some rules that again ensure that only the connectivity (i.e. the set of endpoints of a connected component) matters. These generators and rules are known in the literature as a *special commutative Frobenius algebra* (SCFA). A category where every object is equipped with an SCFA is called a *hypergraph category*.

In this paper, we will focus on the study of hypergraph categories, and their associated string diagrams. While this may seem counter-intuitive at first, the rewriting theory for hypergraph categories is actually simpler than that for generic symmetric monoidal categories. Hence, it will serve as a stepping stone toward doing rewriting for string diagrams with non-branching wires, which is the topic of [BGK+20].

One can model composition and interaction between processes in hypergraph categories using equational reasoning. That is, we can introduce some primitive 'boxes' into our theory and impose a set $\mathcal{E}$ of (diagram) equations that those boxes satisfy. For example, we may introduce a box g and requires that it satisfies

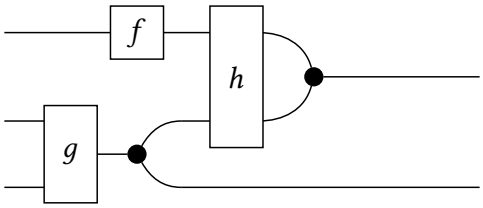


Just like in the case of equational reasoning with terms, when it comes to automated equational reasoning with diagrams, it becomes more useful to consider equations as rewriting rules, with a preferred orientation from left-to-right



It is then natural to ask questions about how well-behaved such a rewriting system is, for example whether it terminates and in that case, whether it yields unique normal forms. However, more fundamentally, one can ask what rewriting in the context of hypergraph categories means. A simple answer is that diagram rewriting is simply term rewriting, performed modulo the axioms of a hypergraph category. Indeed all of the diagram equations above can be represented as terms over the basic generators of the theory, combined via two connectives $\oplus$ and $;$, representing parallel

and sequential composition, respectively. For example, adding boxes f and h, we may have



$$:= \quad (1 \oplus g)\,;\,(f \oplus \delta)\,;\,(h \oplus 1)\,;\,(\mu \oplus 1)$$

Then, we can apply the axioms of a hypergraph category in order to produce the LHS of one of our rewriting rules (i.e. a reducible expression, or redex) as a subterm. For example, the *interchange law*

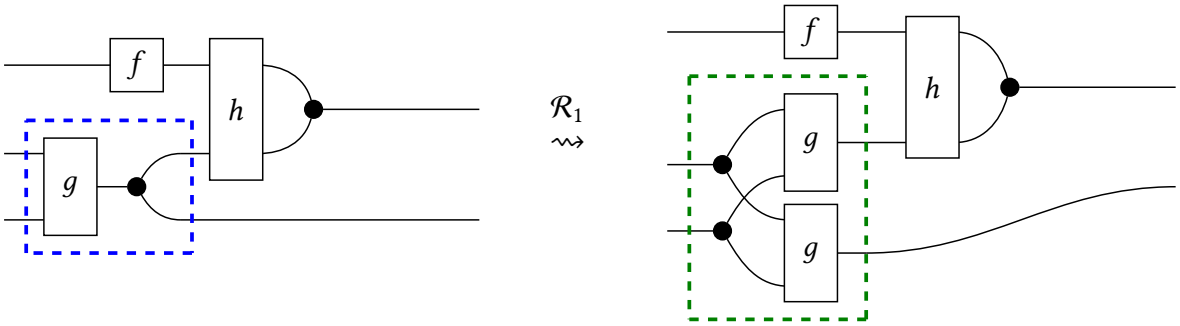
$$(a \oplus b)\,;\,(c \oplus d) = (a\,;\,c) \oplus (b\,;\,d) \tag{1}$$

allows us to produce the subterm $g;\delta$, which is the LHS of the rule $\mathcal{R}_1$ above, at which point we can replace the LHS of $\mathcal{R}_1$ with the RHS

$$(1 \oplus g)\,;\,(f \oplus \delta)\,;\,(h \oplus 1)\,;\,(\mu \oplus 1) = ((1\,;\,f) \oplus (g\,;\,\delta))\,;\,(h \oplus 1)\,;\,(\mu \oplus 1)$$

$$\overset{\mathcal{R}_1}{\rightsquigarrow} ((1\,;\,f) \oplus ((\delta \oplus \delta)\,;\,(1 \oplus \sigma \oplus 1)\,;\,(g \oplus g)))\,;\,(h \oplus 1)\,;\,(\mu \oplus 1)$$

In other words, we do term rewriting in the usual way, modulo the SMC and Frobenius equations. This rewriting-modulo step can be seen as the formal, syntactic underpinning to the intuitive notion of rewriting defined directly on string diagrams. For example, the term rewrite above can be depicted diagrammatically as



However, there are numerous drawbacks to this rewriting-modulo approach. First, some care must be taken in applying the axioms of a hypergraph category to terms. For example, depending on the types of a, b, c, d it is possible for the LHS of the interchange law (1) to be well-defined while the RHS is not. Second, and more importantly, even well-behaved rewriting systems quickly become intractable when doing rewriting-modulo. Even a very simple rewriting system $\mathcal{R}$ becomes very difficult to work with when taken modulo the 12 additional equations implied by the axioms of a hypergraph category. Working purely with terms, this requires a great deal of careful book-keeping to 'reshuffle' a term in such a way that it contains an exact copy of the LHS of a given rule as a subterm, then applying it. This complexity then transfers into all the aspects of the rewriting theory, making it difficult to define what it means, e.g. to enumerate all of the distinct places that a rule *matches* or to determine whether the application of two rules overlap.

Computationally, a much better approach is to take seriously the notion that 'only connectivity matters', and represent a morphism in a hypergraph category as a combinatorial object capturing exactly this connectivity data. Indeed, as the name would suggest, the best choice for that object is a hypergraph. In this paper (and in [BGK+20]) we combine and integrate the results of a series of recent works [BGK+16, BGK+17, BGK+18] to develop a complete rewriting theory for hypergraph categories based on hypergraphs with interfaces. In particular, we show the free hypergraph category over a collection of generators can be represented combinatorially in terms of hypergraphs and show that diagram substitution can be performed via double-pushout hypergraph rewriting.

This corresponds exactly to rewriting modulo the axioms of a hypergraph category, hence all of the hypergraph category axioms are subsumed by hypergraph isomorphism.

Often it is interesting to consider rewriting modulo in systems where each object is equipped with more than one Frobenius algebra. For example, the ZX-calculus used in quantum computation, as well as the system $\mathbb{IB}$ and its derivatives used to model linear systems, signal-flow graphs, and concurrency, involve two interacting Frobenius algebras at their heart. We will show that such systems can be accommodated within this framework while still preserving its good computational properties. As a demonstration, we define a theory of two Frobenius algebras interacting as a bialgebra and illustrate a simple, terminating strategy for transforming diagrams to a pseudo-normal form using hypergraph rewriting.

In Section 2 we provide an introduction to the rationale of string diagram rewriting, while at the same time recalling the preliminary notions of symmetric monoidal categories and PROPs, as well as their respective extensions incorporating a Frobenius structure, with a particular attention for the multi-sorted case. In Section 3 we introduce hypergraphs with interfaces, and the main tool for manipulating them, i.e., *double pushout* (shortly, DPO) rewriting. The section also provides a purely combinatorial interpretation for string diagrams as cospans of hypergraphs, and this result in exploited in Section 4 to prove the completeness of such interpretation and the precise correspondence between string diagram rewriting and DPO rewriting on hypergraphs with interfaces. Section 5 offers a first proof-of-concept for our approach, showing that the equational reasoning on the algebra of groups can be modelled via string diagrams and then by DPO rewriting with a terminating reduction strategy. The main formalisation in this paper captures rewriting modulo a single Frobenius algebra per sort, but we show in Section 6 that this can be easily extended to multiple Frobenius algebras per sort. This latter result is then put to use in Section 7 by tackling a more comprehensive example, which is represented by interacting bialgebras, one of the most-often studied structures in graphical quantum theory. As for the group algebra, our graphical interpretation of such diagrams allows for recasting the associated equational reasoning in term of DPO rewriting with a terminating reduction strategy. A concluding section finally wraps up the paper.

Related work. There is a long tradition of works, both in mathematics and computer science, exploring the link between syntactic and combinatorial representations of formal systems. In our work, we make the following two contributions:

(i) a correspondence between symmetric monoidal categories with Frobenius structure, and cospans of hypergraphs.

(ii) a correspondence between rewriting of string diagrams modulo Frobenius, and double pushout rewriting of hypergraphs with interfaces.

Regarding (i), the research on the precise correspondence between visual languages for monoidal categories and combinatorial structures witnessed a renewed interest in the 1990s [JS91, JSV96], and we refer to [Sel11] for an extensive survey. The case of monoidal categories with Frobenius structures is particularly important: their role in visually modelling circuit-like systems, and possibly their manipulation, has been recognised early on [CW87, GH98], and more recently it has appeared in categorical models of quantum processes [CK17], signal flow graphs [BSZ17a, BE15], electrical circuits [BPSZ19, BCR17], Petri nets [BHPS17] and more.

Also the formal link between Frobenius monoids and cospans (both of sets and of graphs) have been observed, with some variations, in several works [BG01, Lac04, RSW05, ASW09], and most recently in [FS19]. Besides its fully worked out connection with PROPs, what is definitely new of our formulation is the presentation of the multi-sorted version, and especially its lifting to the "multi-Frobenius" case (Section 6).

Regarding (ii) above, the study of string diagram rewriting dates back at least to Burroni's work on polygraphs [Bur93]. In that tradition, the driving motivation is to generalise term rewriting to higher dimensions, including the three-dimensional case of string diagram rewriting: see e.g. [Mim14] for a survey. The approach does not rely on graph rewriting, and goes instead via a completely "syntactic" route: the laws of SMCs are considered as explicit rewriting rules, resulting in rather elaborate rewriting systems, whose analysis is often challenging (see e.g. [Laf03]). Our approach is instead to "absorb" the structural equations of string diagrams into the graph interpretation, and study as rewriting rules only the "domain-specific" equations of the theory under consideration.

A first attempt at modelling string diagram rewriting as graph rewriting was proposed in [DDK10, DK13], and was subsequently used in the Quantomatic proof assistant [KZ15]. That formalism differs from ours in that it does not capture rewriting modulo Frobenius structure, and instead assumes the presence of a categorical trace. Moreover, we directly work in an *adhesive category* [LS05], while the category of "open graphs" used in [DK13] is not adhesive, but instead inherits its good rewriting properties from an embedding into a larger adhesive category. An important aspect of our rewriting is the requirement that DPO rewriting respects a fixed mapping into a graph that serves as an interface to a larger, possibly unknown context. As we will see in Section 3.4, the presence of an interface has implications on which rewrites are applicable: only those that respect the interface will be liftable to a larger context. This has been studied in the DPO literature, most notably within the *rewriting with borrowed contexts* approach [EK04], and it will play an important role in our study of confluence in a sequel to this paper [BGK+20]. The construction in [DK13] also considers rewriting relative to an interface, but here we consider much more general kinds of interfaces, as afforded by rewriting modulo Frobenius structure. This is explored in Section 4.5.

2 SYNTACTIC FOUNDATIONS OF STRING DIAGRAM REWRITING

We will develop two interlinked perspectives on string diagrams and the (co)algebraic structures they express: a *syntactic* perspective based on symmetric monoidal theories and PROPs and a *combinatorial* perspective based on hypergraphs. We will assume the reader is familiar with the basics of category theory in general, and symmetric monoidal categories (of which cartesian monoidal categories are a special case) in particular. The standard reference is [ML98].

To begin, one can ask simply: why is string diagram rewriting interesting? One could give many different answers to this question, but perhaps the one that fits most naturally into a story about rewriting is the following

> *String diagram rewriting is the natural extension of term rewriting for operations with many outputs.*

Term rewriting allows us to recast (universal) algebra as *computation*. That is, it translates an algebraic structure – like a monoid, a ring, or some more exotic formalism – into a system for performing 'computations' in a very general sense.

An *algebraic theory* consists of a signature, i.e. a collection of symbols with arities, as well as a collection of equations between terms built from those symbols and some free variables. Arities are natural numbers that tell us how many inputs a symbol should take, where an arity of 0 indicates something has no inputs, i.e. it is a constant. A simple example is the theory of monoids, consisting of the signature $\Sigma_{\mathbf{Mon}} = \{\cdot : 2, \epsilon : 0\}$ and the equations $E_{\mathbf{Mon}} = \{(x \cdot y) \cdot z = x \cdot (y \cdot z), x \cdot \epsilon = x, \epsilon \cdot x = x\}$.

By making a tiny tweak to an algebraic theory, one obtains a *term rewriting system*. The equations of an algebraic theory are effectively unordered pairs of terms that share some variables. A term rewriting system consists of a signature and a collection of *ordered* pairs of terms called *rewriting rules*. A possible rewriting system for monoids is the following: $R_{\mathbf{Mon}} = \{(x \cdot y) \cdot z \Rightarrow x \cdot (y \cdot z), x \cdot \epsilon \Rightarrow x, \epsilon \cdot x \Rightarrow x\}$.

Unlike the equational theory, the rewriting system has some computational content. Namely, if we apply the rules in a rewriting system from left-to-right in an arbitrary order until no rule applies any more, we obtain (non-deterministic, possibly non-terminating) computations on terms. In the case of $R_{\mathbf{Mon}}$, a particularly nice rewriting theory, the computation in fact always terminates with a unique answer: the term obtained from removing any extra units and bracketing to the right. For example:

$$((a \cdot b) \cdot (c \cdot \epsilon)) \cdot d \Rightarrow ((a \cdot b) \cdot c) \cdot d \Rightarrow (a \cdot b) \cdot (c \cdot d) \Rightarrow a \cdot (b \cdot (c \cdot d))$$

One could also refine this notion of computation, e.g. by considering different rewriting strategies and/or termination criteria. Classical term rewriting, as a discipline, studies the properties of such theories and their associated computations. Namely, it gives techniques for proving a rewriting system is well-behaved in various ways, like being terminating or admitting unique normal forms. It also provides techniques such as Knuth-Bendix completion [KB70] that turn ill-behaved rewriting systems into well-behaved ones. Such techniques have proved to have far-reaching applications in programming languages, computer algebra, and automated theorem proving, see e.g. [DJ90, Vis01] for a survey.

However, algebraic theories, and hence term rewriting systems, can only handle signatures where every operation produces exactly one output. This is so fundamental to the concept of what a 'formula' or a 'term' is, that this limitation may even go unnoticed. Returning to the monoid example, we could write the arities (i.e. numbers of inputs) as well as the *co-arities* (i.e. number of outputs) explicitly as: $\Sigma_{\mathbf{Mon}} = \{\mu : 2 \to 1, \epsilon : 0 \to 1\}$. Note that we have switched from '$\cdot$' to μ, which we will shortly find more convenient.

Suppose we wanted to consider an operation with the signature $\delta : 1 \to 2$, i.e. something which takes 1 input, but produces 2 outputs. If this symbol is meant to represent a function of some kind, this is not a problem. Instead of using just one operation, we can simply give two: $\delta_1 : 1 \to 1$, which gives the first output of δ, and $\delta_2 : 1 \to 1$, which gives the second output. However, this translation makes use of a fundamental property of functions (or more generally, of morphisms in a cartesian category) that the two maps $\langle \delta_1, \delta_2 \rangle$ contain exactly the same data as the overall map δ. If we wish to generalise from functions to other kinds of maps (e.g. non-deterministic operations, probabilistic or quantum processes, etc.), this may not be true any more. For example, if we interpret symbols probabilistically, a constant of the form $p : 0 \to 1$ could represent a fixed probability distribution for a single value, whereas $q : 0 \to n$ could represent a joint distribution over n values. In this case, we would clearly lose some information if we attempt to factorise q as $q_1, q_2, \ldots, q_n$. Indeed, probabilistic maps are most naturally expressed in a (non-cartesian) symmetric monoidal category.

Even in cartesian categories, there may be good computational reasons for wanting to consider operations with multiple outputs. For example, suppose $f : 1 \to 2$ is a function that does some really difficult computation, then returns two copies of the output. While it is surely possible to represent the same function as a pair of functions $f_1 : 1 \to 1$ and $f_2 : 1 \to 1$, each doing the same difficult computation, in some contexts it might be incredibly wasteful. Following this idea to its natural conclusion leads to generalisation from terms to termgraphs [Plu99, BvEG+87].

We argue that the appropriate generalisation of algebraic theory that allows operations with multiple outputs is a *symmetric monoidal theory* (SMT). Indeed, while the syntax of algebraic theories is terms, we will see in the following section that the syntax of SMTs is *string diagrams*.

2.1 Symmetric Monoidal Theories and PROPs

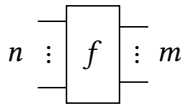
Much like an algebraic theory axiomatises a structure in a cartesian category (typically the category of sets), a symmetric monoidal theory axiomatises a structure in a more general symmetric monoidal

category. It consists of two parts: a signature, which we will call a *monoidal signature* to avoid confusion with the notion of signature in an algebraic theory, and a set of equations.

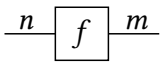
A monoidal signature consists of a set Σ of operations $f : n \to m$ with a fixed *arity* n and *coarity* m, for $n, m \in \mathbb{N}$. Symmetric monoidal theories are defined using Σ-*terms*. The set of Σ-terms is obtained by combining the operations in Σ, *identities* $\mathrm{id}_n : n \to n$ and *symmetries* $\sigma_{m,n} : m + n \to n + m$ for each $m, n \in \mathbb{N}$, by sequential ($;$) and parallel ($\oplus$) composition. This is a purely formal process: given Σ-terms $r : m \to n$, $s : n \to o$, and $t : m' \to n'$, we construct new Σ-terms $r ; s : m \to o$ and $r \oplus t : m + m' \to n + n'$.

Definition 2.1. A *symmetric monoidal theory* is a pair $(\Sigma, \mathcal{E})$ where Σ is a monoidal signature and $\mathcal{E}$ is a set of equations, namely pairs $\langle l, r \rangle$ of Σ-terms $l, r : v \to w$ with the same arity and coarity.

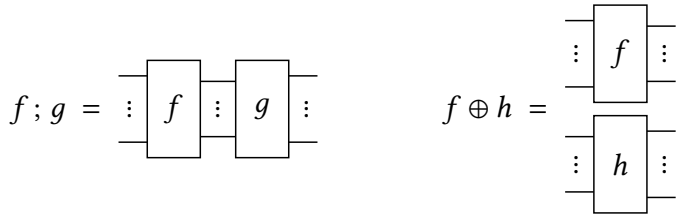
Typically we depict Σ-terms (or more accurately: equivalence classes of Σ-terms) graphically as *string diagrams*. The operation $f : n \to m$ is represented as a box with n wires in a m wires out



We sometimes also write f as a box with single wires in and out, labelled by their (co)arities

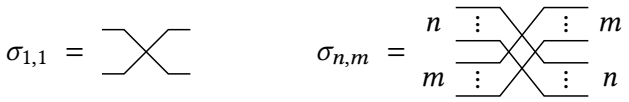


Sequential and parallel composition are depicted as one would expect

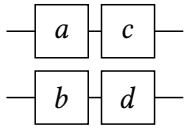


Identities are (sets of) blank wires

and the symmetries are represented by wire-crossings



However this notation is ambiguous. For example



could represent either the Σ-term $(a \oplus b) ; (c \oplus d)$ or the Σ-term $(a ; c) \oplus (b ; d)$. Hence string diagram notation does not represent just one Σ-term, but instead an equivalence class of Σ-terms modulo some equations. It is noteworthy that the appropriate set of equations for this are exactly the axioms of a symmetric monoidal category [Sel11]. Figure 1 shows these equations, adapted to the special case of Σ-terms. This allows us to give a fully syntactic definition of string diagram.

Definition 2.2. A *string diagram* is an equivalence class of Σ-terms, taken modulo the equations in Figure 1.

$$(s\,;\,t)\,;\,u \equiv s\,;\,(t\,;\,u) \qquad \mathrm{id}_n\,;\,s \equiv s \equiv s\,;\,\mathrm{id}_m$$
$$(s \oplus t) \oplus u \equiv s \oplus (t \oplus u) \qquad \mathrm{id}_0 \oplus s \equiv s \equiv s \oplus \mathrm{id}_0$$
$$(s\,;\,u) \oplus (t\,;\,v) \equiv (s \oplus t)\,;\,(u \oplus v)$$
$$\mathrm{id}_m \oplus \mathrm{id}_n \equiv \mathrm{id}_{m+n}$$
$$(\sigma_{m,n} \oplus \mathrm{id}_o)\,;\,(\mathrm{id}_n \oplus \sigma_{m,o}) \equiv \sigma_{m,n+o}$$
$$\sigma_{m,n}\,;\,\sigma_{n,m} \equiv \mathrm{id}_{m+n}$$
$$(s \oplus \mathrm{id}_m)\,;\,\sigma_{m,n} \equiv \sigma_{m,o}\,;\,(\mathrm{id}_m \oplus s)$$

Fig. 1. Equivalences of Σ-terms that generate the same string diagram.

It is often useful to consider not just the SMT itself, but a particularly simple kind of symmetric monoidal category that it presents, called a PROP. This somewhat cryptic name was coined by MacLane [Mac65] as shorthand for '(monoidal) **PRO**duct and **P**ermutation category'.

Definition 2.3. A PROP is a symmetric strict monoidal category with objects the natural numbers, where $\oplus$ on objects is addition. PROP-morphisms are identity-on-objects symmetric strict monoidal functors. PROPs and their morphisms form a category PROP.

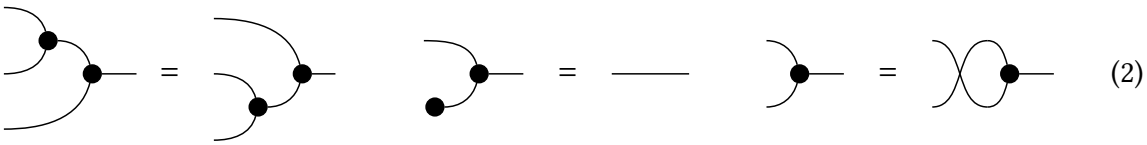
In particular, no restriction is made on how either the morphisms of a PROP or the tensor product of morphisms are defined.

Definition 2.4. Let $\mathrm{S}_{\Sigma,\mathcal{E}}$ be the free (i.e. 'syntactic') PROP presented by the SMT $(\Sigma, \mathcal{E})$. Namely, arrows $u \longrightarrow v$ are Σ-terms $u \longrightarrow v$ modulo the laws of symmetric monoidal categories given in Figure 1 and the smallest congruence containing the equations $t = t'$ for any $\langle t, t' \rangle \in \mathcal{E}$. When $\mathcal{E}$ is empty, we will denote $\mathrm{S}_{\Sigma,\varnothing}$ as S_Σ.

Example 2.5. Consider the SMT $(\Sigma_{\mathbf{CMon}}, \mathcal{E}_{\mathbf{CMon}})$, where

$$\Sigma_{\mathbf{CMon}} := \left\{ \; : 2 \longrightarrow 1, \; : 0 \longrightarrow 1 \right\}$$

and $\mathcal{E}_{\mathbf{CMon}}$ is the set consisting of the following three equations



(2)

Intuitively, the leftmost and the rightmost equations state associativity and commutativity of , while the central equation that is the unit of . For this reason, the PROP freely generated by $(\Sigma_{\mathbf{CMon}}, \mathcal{E}_{\mathbf{CMon}})$ is called the PROP of commutative monoids, denoted by **CMon**.

Just as an algebraic structure can have many different presentations via generators and equations, many different SMTs can generate the same PROP. For example, we could (redundantly) add a second unit law to (2) and the PROP **CMon** would be unchanged. In that sense, a PROP captures the 'essence' of an algebraic structure in a presentation independent way, much like monads or Lawvere theories do for algebraic theories [HP07].

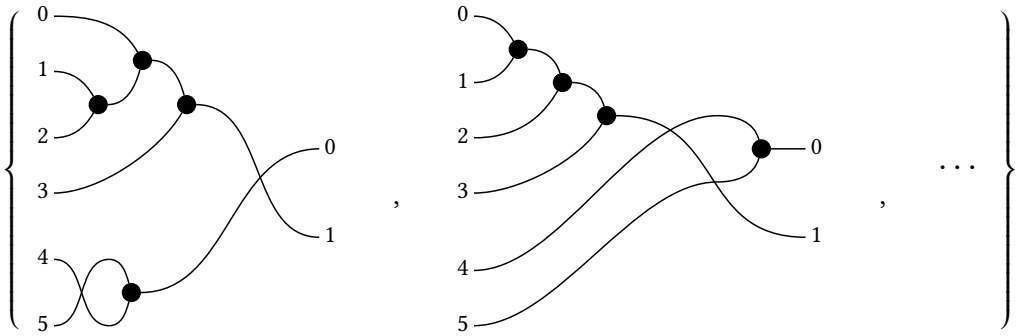
Remark 2.6. Note that we have abused notation in Example 2.5 by giving the equations of the SMT directly as string diagrams, which are technically equivalence classes of Σ-terms, not Σ-terms themselves. Since we form the PROP associated with commutative monoids by quotienting over the axioms of a symmetric monoidal category, this distinction becomes irrelevant. In other words,

choosing any Σ-term to represent the lefthand-side and righthand-sides of the equations (2) will yield the same PROP.

It is often interesting to not only consider PROPs freely generated from an SMT, but also more concrete PROPs, built out of more combinatorial structures. A common theme in the study of PROPs is to initially define a PROP syntactically, then give a more convenient concrete characterisation. A canonical example is the following.

Proposition 2.7. ***CMon*** $\cong \mathbb{F}$*, where* $\mathbb{F}$ *is the PROP whose morphisms* $f : m \to n$ *are functions from the finite set* $[m] := \{0, \ldots, m-1\}$ *to* $[n] := \{0, \ldots, n-1\}$ *and* $[m] \oplus [n] := [m] + [n]$ *is given by the disjoint union of finite cardinals.*

The formal proof of Proposition 2.7 can be found e.g. in [Lac04]. However, for our purposes, it will suffice to give some intuition as to why $\mathbb{F}$ is the PROP for commutative monoids. A morphism $d : m \to n$ in **CMon** can be described as an equivalence class of string diagrams with generators and , modulo the equations (2). Such an equivalence class is precisely identified by specifying, for each input $i \in [m]$ the associated output $j \in [n]$, connected to i by means of multiplication, identity, and/or swap maps. For example, consider this equivalence class of string diagrams, modulo the monoid laws



Every diagram in this class has the property that the inputs $\{0, 1, 2, 3\}$ connect to output 1 and the inputs $\{4, 5\}$ connect to output 0. Hence, we can identify the above equivalence class of diagrams with the function $f : [6] \to [2]$ given by

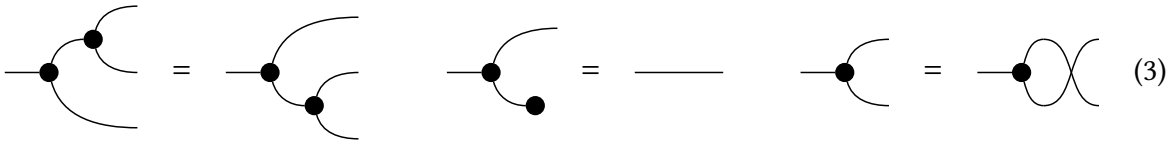
$$f :: \{\ 0 \mapsto 1,\ 1 \mapsto 1,\ 2 \mapsto 1,\ 3 \mapsto 1,\ 4 \mapsto 0,\ 5 \mapsto 0\ \}$$

Of course, commutative monoids can also be presented as an algebraic theory and reasoned about using the usual machinery of terms and term rewriting. A simple example of an SMT that does not have an evident presentation as an algebraic theory is a *comonoid*, which simply turns all of the generators around.

Example 2.8. The SMT $(\Sigma_{\mathbf{CComon}}, \mathcal{E}_{\mathbf{CComon}})$ of *cocommutative comonoids* has a monoidal signature

$$\Sigma_{\mathbf{CComon}} := \left\{ \text{ } : 1 \to 2,\ \text{} : 1 \to 0 \right\}$$

with equations $\mathcal{E}_{\mathbf{CComon}}$ given by

 (3)

By essentially the same argument we gave before, one can see that **CComon** $\cong \mathbb{F}^{op}$.

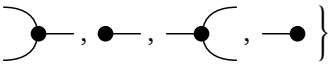
As algebraic structures are presented by SMTs where all of the generators have co-arity 1, it is natural to think of *coalgebraic* structures as SMTs where all of the generators have arity

1. Cocommutative comonoids, as described above, are a simple example of such a coalgebraic structure.

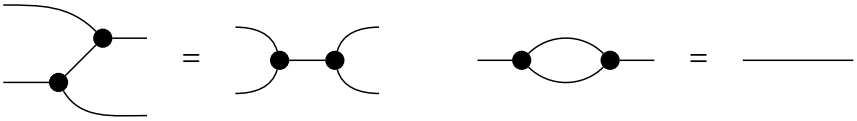
2.2 Frobenius algebras and cospans

In section 2.1, we saw an SMT for an algebraic structure (commutative monoids) as well as an SMT for a coalgebraic structure (cocommutative comonoids). The most interesting SMTs are the ones that have both algebraic *and* coalgebraic parts interacting with each other. One such SMT, and its associated PROP, will play a central role throughout this paper.

Example 2.9. Consider the SMT $(\Sigma_{\mathbf{Frob}}, \mathcal{E}_{\mathbf{Frob}})$, where

$\Sigma_{\mathbf{Frob}} :=$ 

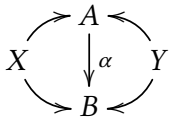
and $\mathcal{E}_{\mathbf{Frob}}$ contains the equations in (2)-(3) and the following ones

 (4)

The PROP freely generated by $(\Sigma_{\mathbf{Frob}}, \mathcal{E}_{\mathbf{Frob}})$ is called the PROP of *special commutative Frobenius algebras* and denoted by **Frob**.

Note that we will often refer to special commutative Frobenius algebras simply as Frobenius algebras. Just like we could give convenient alternative characterisations for PROPs of monoids and comonoids, there is a concrete characterisation of the PROP of Frobenius algebras. To obtain this characterisation, we need to generalise from functions to cospans.

Definition 2.10 ($\mathrm{Csp}(\mathbb{C})$*).* Let $\mathbb{C}$ be a category with all finite colimits. A *cospan* from X to Y is a pair of arrows $X \to A \leftarrow Y$ in $\mathbb{C}$. A morphism $\alpha\colon (X \to A \leftarrow Y) \Rightarrow (X \to B \leftarrow Y)$ is an arrow $\alpha\colon A \to B$ in $\mathbb{C}$ such that the diagram below commutes



Cospans $X \to A \leftarrow Y$ and $X \to B \leftarrow Y$ are *isomorphic* if there exists a morphism of cospans as above, where $\alpha : A \to B$ is an isomorphism. For $X \in \mathbb{C}$, the *identity cospan* is $X \xrightarrow{\mathrm{id}_X} X \xleftarrow{\mathrm{id}_X} X$. The composition of $X \to A \xleftarrow{f} Y$ and $Y \xrightarrow{g} B \leftarrow Z$ is $X \to A +_{f,g} B \leftarrow Z$, obtained by taking the pushout of f and g. This data is the category $\mathrm{Csp}(\mathbb{C})$: the objects are those of $\mathbb{C}$ and the arrows are isomorphism classes of cospans. Finally, $\mathrm{Csp}(\mathbb{C})$ has a monoidal product given by the coproduct in $\mathbb{C}$, with unit the initial object $0 \in \mathbb{C}$.

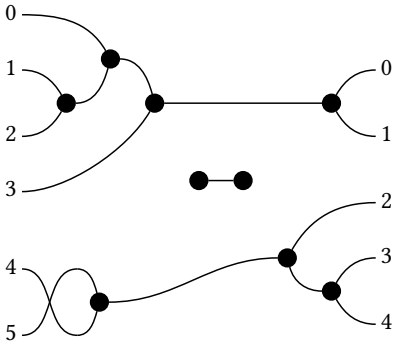
Remark 2.11. It is natural to consider the bicategory of cospans [Bén67], where cospans form the 1-cells and cospan morphisms form the 2-cells. In this case, composition is only associative up to isomorphism. However, taking isomorphism classes makes composition associative on-the-nose, which allows us to define $\mathrm{Csp}(\mathbb{C})$ simply as a category.

Note that, when $\mathbb{C} = \mathbb{F}$, $\mathrm{Csp}(\mathbb{C})$ is a PROP, and furthermore the following holds.

Proposition 2.12 ([BG01, Lac04]). *There is an isomorphism of PROPs* $\mathbf{Frob} \cong \mathrm{Csp}(\mathbb{F})$.

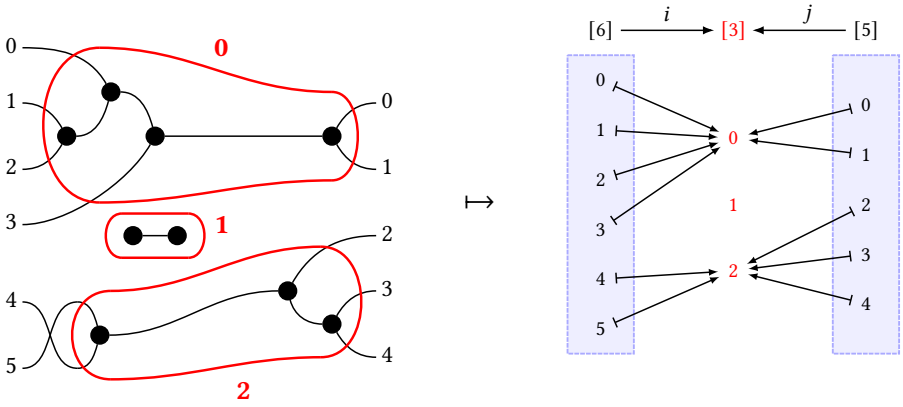
Again we refer to [Lac04] for a formal proof and give some intuition of why this is the case. As with monoids, the relevant data of an equivalence class of morphisms in **Frob** is the connectivity

of inputs and outputs. However, since we have both algebraic and coalgebraic generators, it is possible for many inputs to be connected to many outputs



(5)

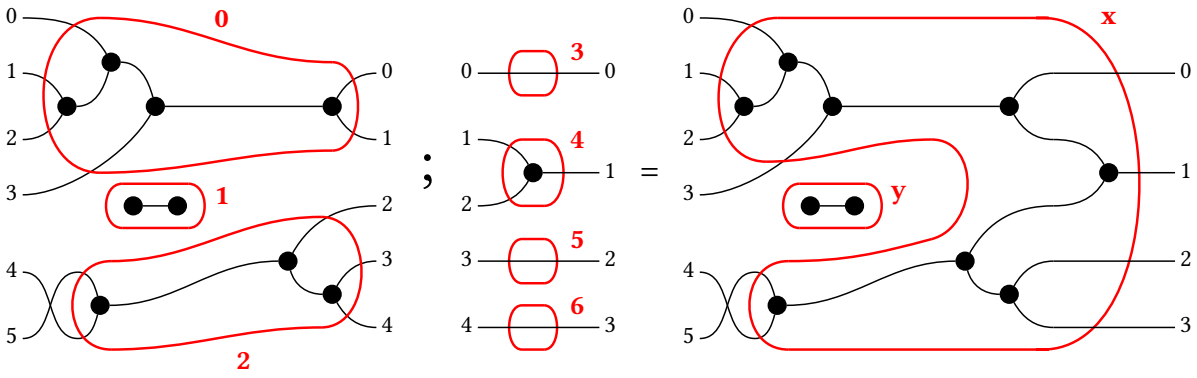
To represent a morphism in **Frob** as a cospan, the inputs become the set on the left, the outputs the set on the right, and the connected components of generators in the diagram form the middle set. The two functions in the cospan then respectively map each input or output to its associated connected component



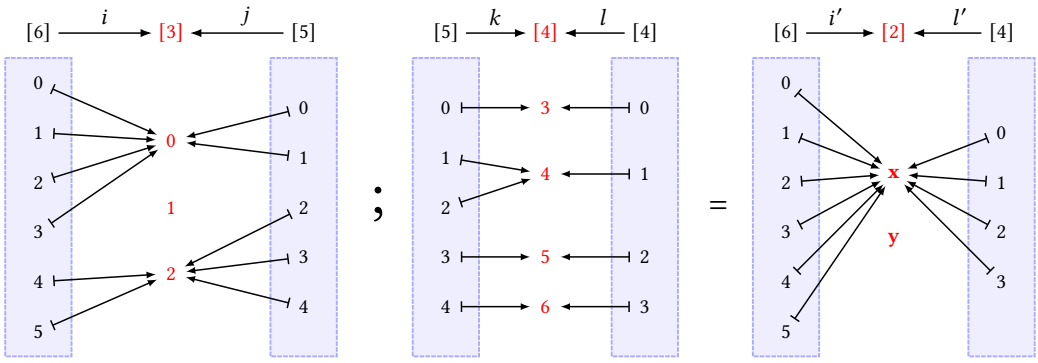
For any cospan, we can construct a diagram using the generators , , , and whose components induce that cospan. Conversely, any two $\Sigma_{\mathbf{Frob}}$ terms inducing the same cospan are equivalent by the Frobenius equations— see e.g. Example 2.13 below.

For $\Sigma_{\mathbf{Frob}}$ terms s, t, the connected components of $s \oplus t$ are the disjoint union of the components of s and t respectively, which matches the tensor product in $\mathrm{Csp}(\mathbb{F})$. The composition $s\,;\,t$ acts on connected components by first taking the disjoint union, then amalgamating together those components that become connected, which is exactly what happens when taking the pushout. This is easiest to see by means of an example.

Example 2.13. Consider the following composition of **Frob**-morphisms, where we label the connected components on the LHS and RHS



The composition of the associated cospans in $\mathrm{Csp}(\mathbb{F})$ is computed as follows



That is, the pushout on the RHS is computed as the disjoint union $\{0,1,2\} + \{3,4,5,6\}$ modulo the equivalence relation generated by $\{j(a) \sim k(a) \mid a \in [5]\} = \{0 \sim 3, 0 \sim 4, 2 \sim 4, 2 \sim 5, 2 \sim 6\}$. The $\sim$-equivalence classes are given by $\mathbf{x} = \{0,2,3,4,5,6\}$ and $\mathbf{y} = \{1\}$ and the new cospan maps i', l' on the RHS are induced by composing i and j with the pushout injections.

2.3 Models of SMTs and PROPs

For algebraic theories, one often wishes to study not the theory in isolation, but its concrete realisation in a category. A model of an SMT is similar in spirit to a model of an algebraic theory.

Definition 2.14. A *model* of an SMT $(\Sigma, \mathcal{E})$ in a symmetric monoidal category $(C, \otimes, I)$ consists of an object $A \in C$ and a morphism $\hat{f} : A^{\otimes n} \to A^{\otimes m}$ for each $f : n \to m \in \Sigma$ such that the equations in $\mathcal{E}$, interpreted as equations between compositions of C-morphisms, are all satisfied.

Note that $A^{\otimes n}$ is shorthand for the n-fold monoidal product of A with itself, where $A^{\otimes 0} := I$. When we refer to a structure defined by an SMT *in* a monoidal category, we really mean a model of that SMT in that category.

A nice feature of PROPs is they allow us to give a presentation-independent notion of model.

Definition 2.15. A *model* of a PROP $\mathbb{A}$ in a symmetric monoidal category $(C, \otimes, I)$ is a strong symmetric monoidal functor $F : \mathbb{A} \to C$.

For a PROP $\mathrm{S}_{\Sigma,\mathcal{E}}$ presented by an SMT $(\Sigma, \mathcal{E})$, these two notions of model coincide. Namely, the chosen 'carrier' object A from Definition 2.14 is $F(1)$ and since each generator of Σ can be regarded as a morphism in the associated PROP, we can let $\hat{f} := F(f)$. The fact that F is a strong symmetric monoidal functor forces all of the equations $\mathcal{E}$ to hold in C, since they hold, by definition, in $\mathrm{S}_{\Sigma,\mathcal{E}}$.

Since our primary interest will be in theories, rather than their models, we will not go into further details here, and conclude our discussion with some examples.

Examples 2.16. For the following symmetric monoidal categories

- $(\mathbf{Set}, \times, 1)$ the category of sets with cartesian product,
- $(\mathbf{Ab}, \otimes, \mathbb{N})$ the category of abelian groups with the tensor product, and
- $(\mathbf{Vect}_k, \otimes, k)$ the category of vector spaces over a field k with the tensor product

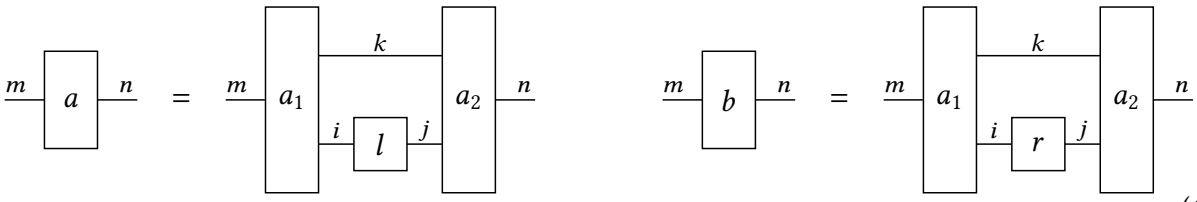
the models of **CMon** are commutative monoids, commutative rings, and associative, commutative k-algebras, respectively. The models of **CComon** in **Set** are just sets, since the only comonoids in a cartesian category come from the diagonal map $\Delta : A \to A \times A$ on an object A.

The only model of **Frob** in **Set** (or any cartesian category) is the trivial one on the 1-element set 1. The models of **Frob** in $\mathbf{Vect}_k$ are in 1-to-1 correspondence with bases [CPV13].

2.4 Syntactic rewriting for PROPs

Equational reasoning on algebraic theories can be mechanised by means of term rewriting. Rewriting plays the same role for SMTs, but here terms are more sophisticated structures than trees.

Definition 2.17. A *rewriting system* $\mathcal{R}$ in a PROP $\mathbb{A}$ consists of a set of *rewriting rules*, i.e. pairs $\langle l, r\rangle$ of morphisms $l, r\colon i \to j$ in $\mathbb{A}$ with the same arities and coarities. Given $a, b\colon m \to n$ in $\mathbb{A}$, a rewrites into b via $\mathcal{R}$, written $a \Rightarrow_{\mathcal{R}} b$, if they are decomposable as follows, for some rule $\langle l, r\rangle \in \mathcal{R}$



(6)

In this case, we say that a contains a *redex* for $\langle l, r\rangle$.

When the PROP under consideration $\mathbb{A}$ is S_Σ for some signature Σ, the notion of rewriting step can be reformulated as follows: in order to apply a rewriting rule $\langle l, r\rangle$ for $l, r : i \to j$ to a diagram d in S_Σ we need to find a redex of l in d. This means finding a *context* C: A term in $\mathrm{S}_{\Sigma+\{\star\colon i\to j\}}$ with exactly one occurrence of $\star$, such that $d = C[l/\star]$. The rewrite then takes $d \Rightarrow_{\mathcal{R}} C[r/\star]$.

With these definitions, diagrammatic reasoning can now be seen as a special case of rewriting. Given an arbitrary SMT $(\Sigma, \mathcal{E})$ we can obtain a rewriting system $\mathcal{R}_\mathcal{E}$ as

$$\mathcal{R}_\mathcal{E} = \{\, \langle t, t'\rangle \mid (t, t') \in \mathcal{E}\} \cup \{\, \langle t', t\rangle \mid (t, t') \in \mathcal{E}\,\}.$$

Proposition 2.18. *Let c, d be two diagrams in S_Σ. Then $c = d$ in the PROP freely generated by $(\Sigma, \mathcal{E})$ iff $c \Rightarrow^*_{\mathcal{R}_\mathcal{E}} d$.*

In order to find a redex in the string diagram a as in (6) one needs to transform a according to the laws of SMCs (Figure 1). Thus rewriting in a PROP always happens modulo these laws.

2.5 Frobenius theories and FROPs

In this section, we will specialise the notions of symmetric monoidal theory and PROP to the situation where there is a fixed, 'default' Frobenius algebra. There are two reasons we might want to do this. The first, as explained in the introduction, is to allow structures that naturally admit "many-to-many" wiring between boxes. For example, wires might capture names or variables that can be shared across many processes, or data that can be copied and deleted.

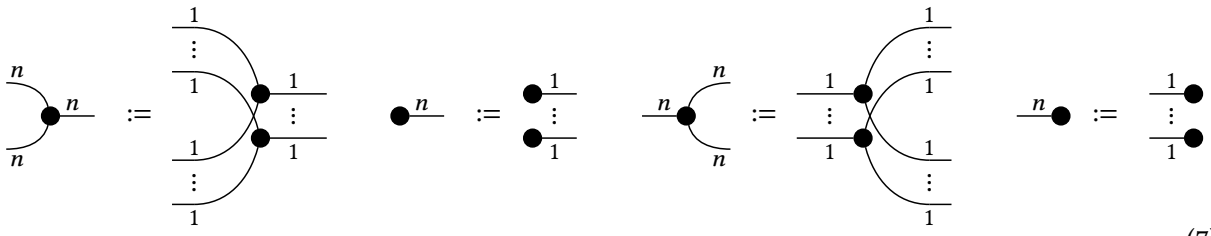
The second, formal reason that it is useful to consider theories with a fixed Frobenius algebra is that, as we'll see in the following two sections, such structures correspond exactly to hypergraphs. This enables us to get a direct combinatoric handle on symmetric monoidal theories and PROPs, and conversely, to capture the structure of hypergraphs in a purely syntactic way.

We will call the 'Frobeniated' versions of SMTs and PROPs *Frobenius theories* and *FROPs*, respectively.

Definition 2.19. A *Frobenius theory* is a pair $(\Sigma, \mathcal{E})$ of a symmetric monoidal signature Σ and a set $\mathcal{E}$ of well-typed equations over Σ'-terms, where $\Sigma' := \Sigma + \Sigma_{\mathbf{Frob}}$.

Definition 2.20. A *FROP* is a PROP $\mathbb{A}$ equipped with a fixed special commutative Frobenius algebra ( ) on the object $1 \in \mathrm{ob}(\mathbb{A})$.

Note that we only assume that the object 1 carries a Frobenius algebra. However, this extends in the obvious way to a Frobenius algebra on every object $n \in \mathrm{ob}(\mathbb{A})$

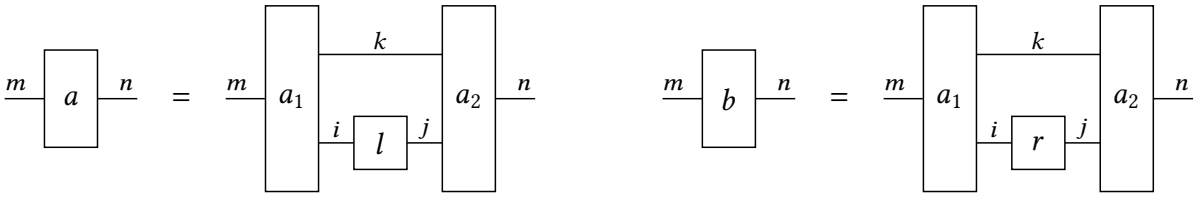


(7)

Just as SMTs present PROPs, Frobenius theories present FROPs. The FROP $\mathbf{H}_{\Sigma,\mathcal{E}}$ presented by a Frobenius theory $(\Sigma, \mathcal{E})$ has morphisms Σ'-terms over $\Sigma' := \Sigma + \Sigma_{\mathbf{Frob}}$, modulo the SMC equations in Figure 1 as well as the equations $\mathcal{E}' := \mathcal{E} + \mathcal{E}_{\mathbf{Frob}}$. The chosen Frobenius algebra on the object $1 \in \mathbb{N}$ of the presented FROP is then the one arising from the generators in $\Sigma_{\mathbf{Frob}}$.

Remark 2.21. While it should be clear why we used the notation $\mathbf{S}_{\Sigma,\mathcal{E}}$ to represent the 'syntactic' PROP, defined by an SMT, it is not yet obvious why we chose the notation $\mathbf{H}_{\Sigma,\mathcal{E}}$ for the analogous concept involving Frobenius theories and FROPs. This is meant to indicate that we have formed the free *hypergraph category* over $(\Sigma, \mathcal{E})$, where a hypergraph category is a symmetric monoidal category where each object is equipped with a fixed choice of Frobenius algebra (see e.g. [Kis14]).

Syntactic rewriting in a FROP is defined just as it was for PROPs in Definition 2.17, except when defining a rule $l \rightsquigarrow r$, the equations



are taken modulo the Frobenius equations in addition to the SMC equations.

2.6 Coloured PROPs and FROPs

We will conclude our discussion on syntactic foundations by introducing multi-sorted versions of the constructions we have introduced before. Aside from being of interest as a generalisation of multi-sorted algebraic structures, these concepts will come in handy in Section 6 when it comes to formalising rewriting modulo multiple, interacting Frobenius algebras.

For a set C, let $C^\star$ denote the set of words over C. Then $C^\star$ carries the structure of a monoid, where the unit is the empty word (written ε) and binary operation is word concatenation (written vw for all words $v, w \in C^\star$).

A multi-sorted SMT is a triple $(C, \Sigma, \mathcal{E})$ where C is a set of sorts, or *colours*, Σ a set of operations having arities and coarietes in $C^\star$, and $\mathcal{E}$ is a set of equations between Σ-terms with matching arities and co-arities.

Just as single-sorted SMTs present PROPs, multi-sorted SMTs present *coloured PROPs*.

Definition 2.22 (Coloured PROP). Given a finite set C of colours, a *C-coloured PROP* $\mathbb{A}$ is a symmetric strict monoidal category where the set of objects is $C^\star$ and the monoidal product on objects is word concatenation. A morphism from a C-coloured PROP $\mathbb{A}$ to a C'-coloured PROP $\mathbb{A}'$ is a symmetric strict monoidal functor $H \colon \mathbb{A} \to \mathbb{A}'$ acting on objects as a monoid homomorphism induced by a function $C \to C'$. Coloured PROPs and their morphisms form a category CPROP.

It is worth to note that fixing a colour $\bullet$ and restricting to $\{\bullet\}$-coloured PROPs and morphisms between them yields PROP as a full sub-category of CPROP.

Both categories CPROP and PROP have coproducts, which will be useful for the constructions to follow. It is instructive to see how the coproduct in PROP differs from the one in CPROP. In PROP, coproducts consist of formal compositions of morphisms from the two consistituent components, on a single sort. For example, the coproduct of **Frob** with itself will consist of diagrams made from two copies of the Frobenius algebra generators, which we will typically depict in two different colours

modulo two copies of the Frobenius equations, one for each colour. On the other hand, the coproduct in CPROP will yield morphisms on two *different* sorts

again modulo two copies of the Frobenius equations. In particular, the generators of the $\bullet$ Frobenius algebra cannot be composed with the generators of the $\bullet$ Frobenius algebra. To emphasise this difference, we write the coproduct in CPROP as $\mathbf{Frob}_{\bullet} + \mathbf{Frob}_{\bullet}$. More generally for a set of colours C, we will write $\mathbf{Frob}_C$ for $\sum_{c \in C} \mathbf{Frob}$.

The category $\mathbf{Frob}_C$ will play an important role in the coming sections. Much like for **Frob**, we can make a combinatoric version of $\mathbf{Frob}_C$ using cospans. However, rather than cospans of sets, we should use cospans of sets whose elements are labelled by colours in C. Formally, we can express C-coloured sets as objects of the slice category $\mathbb{F} \downarrow C$, whose objects are pairs of a finite cardinal $[n]$ and a labelling function $w : [n] \to C$ assigning a colour to each element in $[n]$ and whose morphisms are functions respecting the labelling.

Proposition 2.23. *$\mathbb{F} \downarrow C$ is a coloured PROP.*

Proof. The pair $([n], w : [n] \to C)$ can equivalently be given as a word in $C^\star$ of length n whose i-th letter is $w(i)$. As a slice category, $\mathbb{F} \downarrow C$ inherits coproducts from $\mathbb{F}$ that make it a strict symmetric monoidal category. It is straightforward to check the inherited coproduct acts on objects by concatenation of words. □

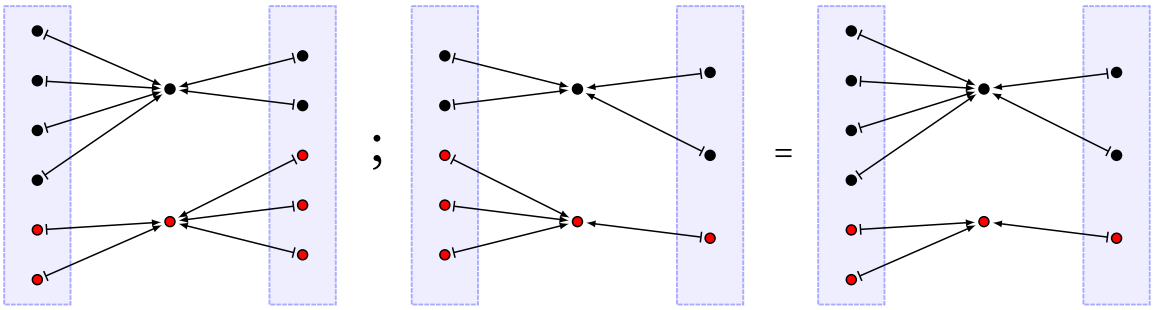
Theorem 2.24. *For a finite cardinal $C \in \mathbb{F}$, $\mathbf{Frob}_C \cong \mathrm{Csp}(\mathbb{F} \downarrow C)$ is an isomorphism of coloured PROPs.*

Proof. Using a simple inductive argument, Lack showed in [Lac04] that $\mathbf{Frob} \cong \mathrm{Csp}(\mathbb{F})$. This generalises easily to the multi-sorted case since

$$\mathbf{Frob}_C = \sum_{c \in C} \mathbf{Frob} \cong \sum_{c \in C} \mathrm{Csp}(\mathbb{F}) \cong \sum_{c \in C} \mathrm{Csp}(\mathbb{F} \downarrow 1) \cong^\star \mathrm{Csp}(\mathbb{F} \downarrow \sum_{c \in C} 1) \cong \mathrm{Csp}(\mathbb{F} \downarrow C)$$

The first equality is just the definition of $\mathbf{Frob}_C$, the first isomorphism follows from Lack's result, the next isomorphism follows from the fact that $\mathbb{F} \cong \mathbb{F} \downarrow 1$, and the last isomorphism is again obvious. Thus we need to justify only the isomorphism marked $\star$. The reason for this is, essentially, the fact that coproducts and pushouts commute, and indeed *as categories* $\sum_{c \in C} \mathrm{Csp}(\mathbf{Set}_f \downarrow 1) \cong^\dagger \mathrm{Csp}(\mathbf{Set}_f \downarrow \sum_{c \in C} 1)$, where $\mathbf{Set}_f$ is the category of finite sets and functions. As coloured PROPs, one additionally needs to keep in mind the objects are not mere sets but words. This boils down to the argument given in the proof of Proposition 2.23, but let us elaborate. The categories $\sum_{c \in C} \mathrm{Csp}(\mathbf{Set}_f \downarrow 1)$ and $\mathrm{Csp}(\mathbf{Set}_f \downarrow \sum_{c \in C} 1)$ can both be considered to have the objects of $\mathbf{Set}_f \downarrow C$, i.e. functions $X \to C$ where X is a finite set. Choosing an ordering of the elements of X, as in the proof of Proposition 2.23, is a uniform way of passing from categories to coloured PROPSs in all three cases, meaning that the $\dagger$ isomorphism of categories above implies the $\star$ isomorphism of coloured PROPs. □

Morphisms and composition look much like they did in the example at the end of Section 2.2, but with sets replaced by coloured sets. Taking $C = [2] \cong \{\bullet, \bullet\}$, an example of composition in $\mathsf{Csp}(\mathbb{F} \downarrow C)$ is the following



Even though coproducts in CPROP yield disjoint colours, the colours in two C-coloured PROPs can be identified by using a pushout, as we see in the following example.

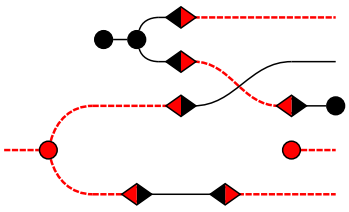
Example 2.25. The free C-coloured PROP on the theory $(C, \varnothing)$ with an empty signature is written $\mathbf{P}_C$ and has arrows $w \longrightarrow v$ the permutations of w into v (thus arrows exist only when the word v is an anagram of the word w). Given C-coloured PROPs $\mathbb{A}$ and $\mathbb{A}'$, we use notation $\mathbb{A} +_C \mathbb{A}'$ for the *pushout* in CPROP of the span of the inclusions $\mathbb{A} \leftarrow \mathbf{P}_C \rightarrow \mathbb{A}'$: in a nutshell, $\mathbb{A} +_C \mathbb{A}'$ is the coproduct $\mathbb{A} + \mathbb{A}'$ where we have identified the copy from $\mathbb{A}$ and from $\mathbb{A}'$ of each $c \in C$. Thus $\mathbb{A} +_C \mathbb{A}$ is also a C-coloured PROP.

Remark 2.26. For reasons that will become clear later, it will often be more valuable if, rather than identifying the two colours, we formally introduce an isomorphism between them. That is, we introduce two new *colour change* generators $\{$, $\}$ and impose the equations

In this case, we will obtain a coloured PROP that is equivalent (but not isomorphic) to the one we described in Example 2.25.

The notions of Frobenius theory and FROP extend in the obvious way to C-coloured Frobenius theories $(C, \Sigma, \mathcal{E})$ and FROPs $\mathbf{H}_{C,\Sigma,\mathcal{E}}$. Namely, each colour $c \in C$ is equipped with a distinct Frobenius algebra. Then, similar to equations (7), these induce a Frobenius algebra on any word $w \in C^\star$.

Example 2.27. Fix a set $C = \{\bullet, \bullet\}$ of colours and a signature Γ consisting of the two colour-change operations, $\{$, $\}$. We may construct the free coloured FROP $\mathbf{H}_{C,\Gamma,\varnothing}$. Here is an example of a string diagram in this category



(8)

We claim that $\mathbf{H}_{C,\Gamma,\varnothing}$ is the same as the category of finite directed *bipartite graphs* (with interfaces). This will become clear in Example 4.6, after the characterisation provided by Corollary 4.2.

With the addition of colours, we now have all of the tools we need to understand string diagram rewriting from the purely syntactic point of view, where we are rewriting Σ-terms modulo a set of structural equations.

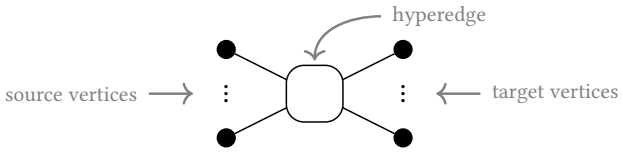
3 COMBINATORIAL FOUNDATIONS OF STRING DIAGRAM REWRITING

While one can get quite some mileage out of treating string diagrams purely syntactically, we argued in Section 1 that this point of view is often unwieldy. This comes from the fact that we are never doing rewriting on Σ-terms themselves, but rather Σ-terms modulo the SMC (and Frobenius) equations. This phenomenon is not unique to string diagrams: complications inevitably arise whenever one considers rewriting modulo a set of equations [Hue80, PS81, BD89].

One way to avoid the complexity of rewriting modulo equations is to choose a better representation for string diagrams that 'absorbs' the SMC and Frobenius equations directly into the representation. This is analogous to the way, in term rewriting systems, it can be fruitful to consider multisets of expressions directly, rather than terms modulo associativity and commutativity. In this section, we will see that

Hypergraphs give a canonical, combinatorial representation for string diagrams.

A hypergraph is a generalisation of a graph, which comes in a few different variations. In all of these variations, edges, which connect precisely two nodes, are replaced by *hyperedges*, which connect arbitrary numbers of nodes together. The particular variation we will focus on have hyperedges that are both directed and ordered. That is, each hyperedge has an ordered list of source nodes and an ordered list of target nodes. We will depict these hyperedges using a symbol that looks rather like a box in a string diagram, with source nodes connecting to the left and target nodes connecting to the right



This is by no means a coincidence: these will indeed play the role of the boxes that represent morphisms in string diagrams.

Once we represent string diagrams using hypergraphs, we will show that rewriting of string diagrams can be accomplished using double pushout (DPO) rewriting. This is a standard technique for performing rewrites on graphs and graph-like structures, where the lefthand-side of a rule is first 'cut out' of the target graph using an operation called the *pushout complement*, then the righthand-side is 'glued in' using a pushout. The whole process results in a diagram of two pushout squares side-by-side (cf. equation (13)), hence the name *double* pushout.

The only extra complexity we need to handle when applying the DPO approach to string diagrams is to account for the inputs and outputs of a string diagram, i.e. the wires left dangling to the left and the right. These form an *interface* to a possibly larger diagram (e.g. one consisting of multiple string diagrams plugged together), and this interface should be respected by rewriting. To account for this, we introduce double-pushout rewriting with interfaces (DPOI) in Section 3.4. This will give us the right tool for establishing a formal correspondence between the syntactic notion of string diagram rewriting in the previous section and the combinatorial one developed in this section.

3.1 The category of labelled hypergraphs

DPO rewriting makes sense in any category with pushouts, but it is often considered in a category where those pushouts obey certain well-behavedness conditions, such as adhesive categories [LS05]. For our purposes, we will skip an abstract overview of DPO rewriting in an arbitrary adhesive category and focus on the specific category **Hyp** of directed hypergraphs.

An object G of **Hyp** is a hypergraph, which consists of a set of *nodes* $G_\star$ and for each $k, l \in \mathbb{N}$ a (possibly empty) set of *hyperedges* $G_{k,l}$ with k (ordered) sources and l (ordered) targets. That is, for each $0 \leq i < k$ we have the i^{th} source map $s_i \colon G_{k,l} \to G_\star$, and for each $0 \leq j < l$, the j^{th} target

map $t_j : G_{k,l} \to G_\star$. The arrows of **Hyp** are hypergraph homomorphisms: functions $G_\star \to H_\star$ such that, for each k, l, $G_{k,l} \to H_{k,l}$, they respect the source and target maps in the obvious way. The following definition characterises **Hyp** as a presheaf topos, and as such, it is adhesive [LS05].

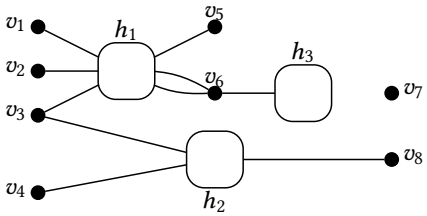
Definition 3.1 (Hypergraphs). The category of finite directed hypergraphs **Hyp** is the functor category $\mathbb{F}^{\mathbf{I}}$ where **I** has as objects pairs of natural numbers $(k, l) \in \mathbb{N} \times \mathbb{N}$ together with one extra object $\star$. For each $k, l \in \mathbb{N}$, there are $k + l$ arrows from (k, l) to $\star$.

Nodes will be drawn as dots and a (k, l) hyperedge h will be drawn as a rounded box, whose connections on the left represent the list $[s_1(h), \ldots, s_k(h)]$, ordered from top to bottom, and whose connections on the right give $[t_1(h), \ldots, t_l(h)]$.

Example 3.2. Let G be the hypergraph with nodes $\{v_1, \ldots, v_8\}$, a $(3, 3)$-hyperedge h_1, a $(2, 1)$-hyperedge h_2, and a $(1, 0)$-hyperedge h_3, and the following source and target maps

$$\begin{array}{lll} s_1(h_1) := v_1 & t_1(h_1) := v_5 & \\ s_2(h_1) := v_2 & t_2(h_1) := v_6 & , \\ s_3(h_1) := v_3 & t_3(h_1) := v_6 & \end{array} \qquad \begin{array}{ll} s_1(h_2) := v_3 & \\ s_2(h_2) := v_4 & , \\ t_1(h_2) := v_8 & \end{array} \qquad s_1(h_3) := v_6$$

Then G is drawn as follows



(9)

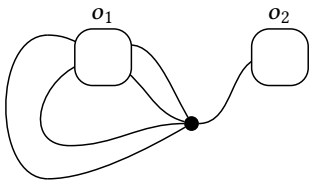
There are often many ways one can generalise concepts from graphs to hypergraphs. In order to fix conventions, we will define some basic concepts for hypergraphs that will be useful for later.

Definition 3.3. For a hyperedge h, node v and $\alpha \in \{s, t\}$, a *connection* for v is a triple (α, h, i) such that $\alpha_i(h) = v$. The *degree* $\deg(v)$ of a node v is the number of distinct connections in G.

Connections are sometimes called *tentacles* in the hypergraph literature. For us, their main utility is obtaining the correct notion of degree of a node when it is connected multiple times to the same hyperedge. For example, v_6 in (9) has degree 3, even though it is only connected to 2 distinct hyperedges.

Definition 3.4. A *path* in a hypergraph is an alternating list $p = [p_1, \ldots, p_n]$ of hyperedges and nodes such that for all hyperedges p_i, the nodes p_{i-1} and p_{i+1} are a source and target for p_i, when they are defined (i.e. when $i > 1$ and $i < n$, respectively). A hypergraph is said to be *acyclic* if it has no path containing the same node twice.

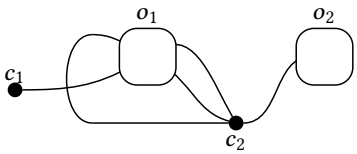
A monoidal signature Σ can be regarded as a directed hypergraph G_Σ with a single node. Each symbol in the signature is depicted as a hyperedge with a number of sources equal to the arity and a number of targets equal to the coarity, all connected to the single node. For example, $\Sigma = \{o_1 : 2 \to 2, o_2 : 1 \to 0\}$ yields

$G_\Sigma :=$ 

(10)

This extends naturally to multi-sorted signatures (C, Σ) by letting the nodes of $G_{(C,\Sigma)}$ correspond to colours in C and for a symbol $o : u \to v$ in Σ, adding a corresponding hyperedge where the i^{th}

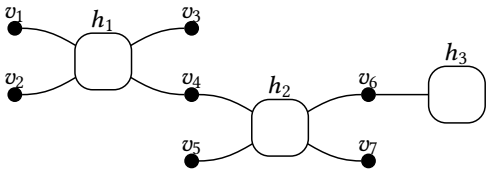
colour in u is the i^{th} source node and the j^{th} colour in v is the j^{th} target node. For instance the hypergraph for (C, Σ) where $C = \{c_1, c_2\}$ and $\Sigma = \{o_1 \colon c_1c_2 \to c_2c_2, o_2 \colon c_2 \to \epsilon\}$ is depicted as follows



(11)

The utility of writing a monoidal signature (C, Σ) as a hypergraph is that we can now define hypergraphs labelled by (C, Σ) as the slice category $\mathbf{Hyp}_{C,\Sigma} := \mathbf{Hyp} \downarrow G_{C,\Sigma}$. That is to say, an object of $\mathbf{Hyp}_{C,\Sigma}$ consists of a hypergraph G together with a graph homomorphism $l \colon G \to G_{C,\Sigma}$. Intuitively l *labels* each node of G with a colour in C and each hyperedge with an operation in Σ. Observe that this definition ensures that a Σ-operation $o \colon u \to v$ labels a hyperedge only when the label of its input (resp. output) nodes forms the word u (resp. v). We call such objects (C, Σ)-hypergraphs and we visualise them as hypergraphs whose nodes n are coloured by $l(n)$ and whose hyperedges h are labelled by $l(h)$.

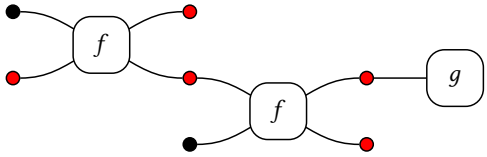
Example 3.5. Considered the following (unlabelled) hypergraph



labelled by the signature (10) as follows

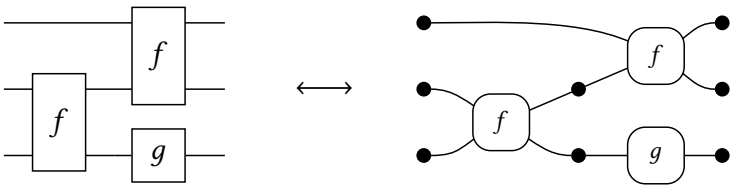
$$l := \begin{cases} v_1, v_5 \mapsto \bullet \\ v_2, v_3, v_4, v_6, v_7 \mapsto \bullet \\ h_1, h_2 \mapsto f \\ h_3 \mapsto g \end{cases}$$

with $c_1 := \bullet$ and $c_2 := \bullet$. This is depicted as



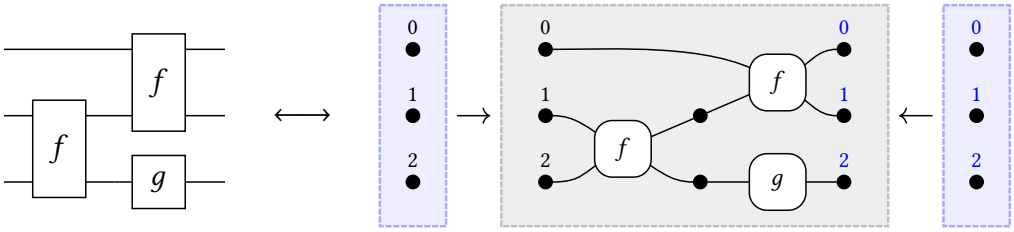
3.2 String diagrams as cospans of hypergraphs

We have already written hypergraphs in a way that is suggestive of how, in the next section, we will relate them to string diagrams. Namely, hyperedges play the role of the 'boxes' in a string diagram and nodes play the role of 'wires', insofar as they allow us to connect boxes to each other. So, it should seem plausible how one could interpret a string diagram as a hypergraph and vice-versa. For example



We will formalise this translation in Section 4, but before we do so, we need to answer a couple of open questions. First, in translating from a string diagram to a hypergraph, we seem to have lost some data. Namely, we no longer know which nodes should be treated as inputs and which as

ouputs and in what order. We can solve this by replacing hypergraphs with cospans of hypergraphs $M \to G \leftarrow N$ where M and N are discrete hypergraphs embedding the inputs and outputs, respectively



Here, the labels are used to indicate how the two graphs M and N are embedded into G.

For generic M, N, this does not impose an ordering on the inputs and outputs, but we always take M and N to be finite cardinals, i.e. sets of the form $[n] = \{0, 1, \ldots, n-1\}$, considered as hypergraphs. More formally, there is a faithful, coproduct-preserving functor $D : \mathbb{F} \to \mathbf{Hyp}_\Sigma$ sending each $i \in \mathrm{ob}\mathbb{F} = \mathbb{N}$ to a hypergraph whose set of nodes is $[i]$ and sending each function to the induced homomorphism of discrete hypergraphs. Then, we can let $M := Dm, N := Dn$.

By introducing a functor that 'picks out' the objects, we get a more refined notion of a cospan category than the one encountered in section 2.2.

Theorem 3.6. *Let $\mathbb{X}$ be a PROP whose monoidal product is a coproduct, $\mathbf{C}$ a category with finite colimits, and $F : \mathbb{X} \to \mathbf{C}$ a coproduct-preserving functor. Then there exists a PROP $\mathrm{Csp}_F(\mathbf{C})$ whose arrows m to n are isomorphism classes of $\mathbf{C}$ cospans $Fm \to C \leftarrow Fn$.*

Proof. Composition in $\mathrm{Csp}_F(\mathbf{C})$ is given by pushout as in Definition 2.10. Given $Fm \to C \leftarrow Fm' \in \mathrm{Csp}_F(\mathbf{C})(m, m')$ and $Fn \to D \leftarrow Fn' \in \mathrm{Csp}_F(\mathbf{C})(n, n')$ their monoidal product is the cospan

$$F(m+n) \xrightarrow{\sim} Fm + Fn \to C + D \leftarrow Fm' + Fn' \xleftarrow{\sim} F(m'+n')$$

where the leftmost and rightmost maps are iso since F preserves the monoidal product (given by the coproduct). It follows that this data defines a strict monoidal category.

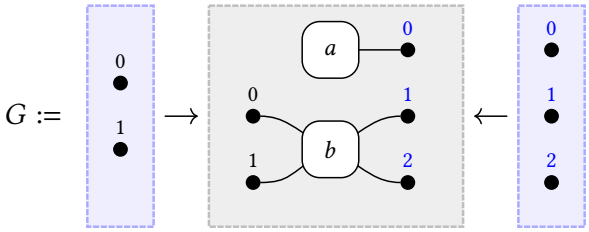
The symmetries are inherited from $\mathbb{X}$, being the following cospans

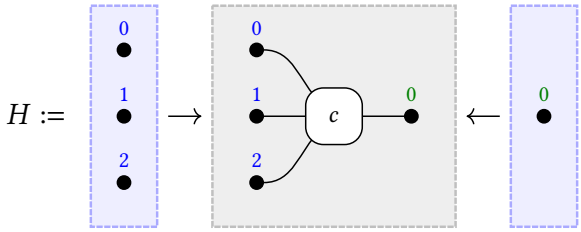
$$F(m+n) \xrightarrow{F\sigma_{m,n}} F(n+m) \leftarrow F(n+m)$$

To see that the symmetries are natural, it suffices to note that the symmetry structure in $\mathbb{X}$ is determined by the universal property of the coproduct, which is preserved by F. □

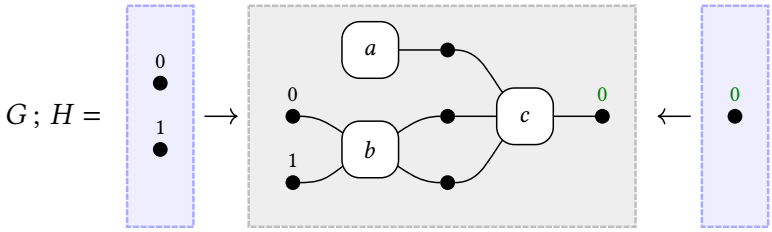
Our main example is $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$, which we sometimes refer to as the category of *discrete cospans of hypergraphs*. Composition and monoidal product in this category correspond exactly to the intuitive notions of plugging string diagrams together and putting diagrams side-by-side.

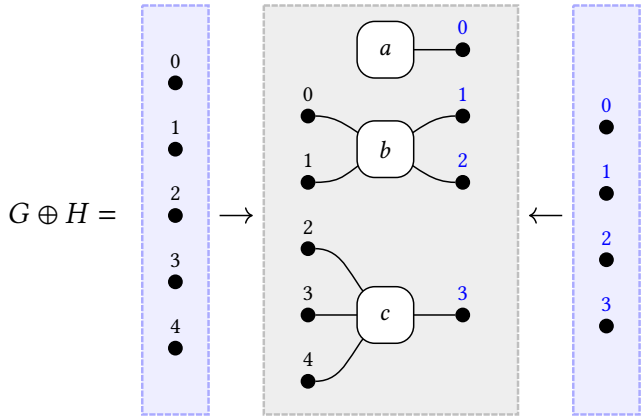
Example 3.7. Consider the following morphisms $G : 2 \to 3, H : 3 \to 1$ in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$



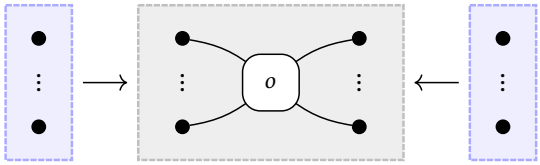


We can form the composition and monoidal product as follows





The composition and tensor product in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ now give us a way to interpret string diagrams (or more precisely, Σ-terms) as hypergraphs. For each of the generators $o : m \to n$ in Σ, we can form a cospan $m \to O \leftarrow n$ where O consists of a single o-labelled hyperedge with m input nodes and n output nodes



Thus, any Σ-term is interpreted by taking compositions and monoidal products of such cospans.

This gives us a recipe for interpreting any string diagram as a cospan of hypergraphs, but what about interpreting a generic cospan of hypergraphs as a string diagram? In particular, note that all of the examples above have involved hypergraphs where each node has at most one in-hyperedge and one out-hyperedge. We made no such restriction when we defined hypergraphs, so how can we make sense of more general hypergraphs where a single node is connected to many hyperedges?

Thankfully, we already answered this question without realising it back in Section 2.2, when we showed that $\mathrm{Csp}(\mathbb{F})$ is isomorphic to the PROP of Frobenius algebras. The only missing ingredient is to find a relationship between $\mathrm{Csp}(\mathbb{F})$ and $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$. It turns out that the first embeds faithfully in the latter, a fact that arises as a special case of the following theorem.

THEOREM 3.8. *Let $\mathbb{X}$ be a PROP whose monoidal product is a coproduct and $\mathbf{C}$ a category such that $\mathbb{X}$ and $\mathbf{C}$ have finite colimits, and $F : \mathbb{X} \to \mathbf{C}$ a colimit-preserving functor. Then there is a homomorphism*

of PROPs $\widetilde{F} : \mathrm{Csp}(\mathbb{X}) \to \mathrm{Csp}_F(\mathbf{C})$ *that sends* $m \xrightarrow{f} X \xleftarrow{g} n$ *to* $Fm \xrightarrow{Ff} FX \xleftarrow{Fg} Fn$. *If* F *is full and faithful, then* $\widetilde{F}$ *is faithful.*

Proof. Since F preserves finite colimits, $\widetilde{F}$ preserves composition (since it preserves pushouts) and monoidal product (since it preserves coproducts). Symmetries, which are inherited from $\mathbb{X}$, are clearly preserved. Finally, suppose that $\widehat{F}(m \xrightarrow{f} X \xleftarrow{g} n) = \widehat{F}(m \xrightarrow{f'} Y \xleftarrow{g'} n)$. Then we have a commutative diagram in $\mathbf{C}$

$$\begin{array}{ccccc} & Ff & FX & Fg & \\ Fm & \nearrow & \downarrow \psi & \nwarrow & Fn \\ & Ff' & FY & Fg' & \end{array}$$

where ψ is an iso. Since F is full there exists $\varphi : X \to Y$ with $F\varphi = \psi$. Since F is faithful, φ is an isomorphism. Hence, the cospans $m \xrightarrow{f} X \xleftarrow{g} n$ and $m \xrightarrow{f'} X \xleftarrow{g'} n$ are equal in $\mathrm{Csp}(\mathbb{X})$, so $\widehat{F}$ is faithful. □

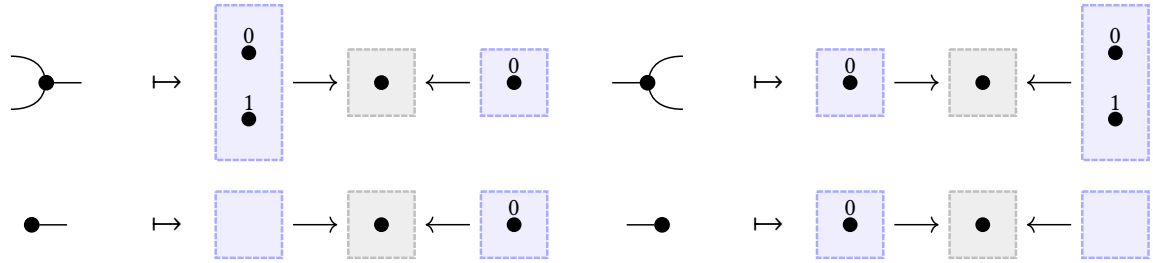
From this, we get the following corollary.

Corollary 3.9. *There is a faithful PROP homomorphism* $\widetilde{D} : \mathrm{Csp}(\mathbb{F}) \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$.

That is, we get an embedding of the PROP $\mathbf{Frob} \cong \mathrm{Csp}(\mathbb{F})$ into our PROP $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ of 'combinatorial string diagrams'.

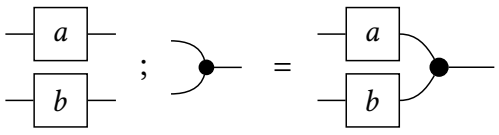
Definition 3.10. Let $[\cdot] : \mathbf{Frob} \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ be the homomorphism obtained by composing the isomorphism of Proposition 2.12 with the homomorphism of Corollary 3.9.

Following the recipe from Section 2.2 relating Frobenius algebra diagrams to cospans, the four basic generators of a Frobenius algebra map to cospans as follows

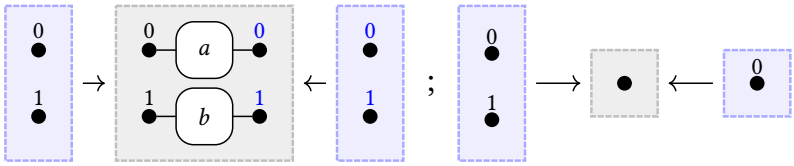


where all of the cospan maps are given by the unique function into the one-element set. These capture in a compositional way all the ways in which a node in a hypergraph could have different numbers of in- and out-hyperedges.

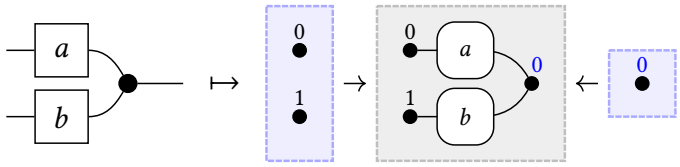
Example 3.11. Consider the following composition of string diagrams



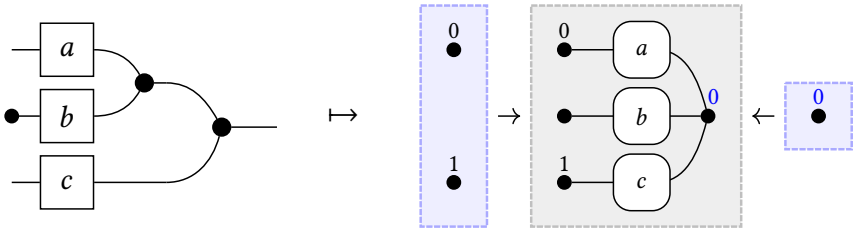
The corresponding composition in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ looks like this



This is computed by taking the pushout, which identifies the two nodes connected to the outputs of the a and b-labelled hyperedges. Hence, we obtain



By composing with the Frobenius algebra generators, we can obtain even more general cospans. For example



Note how, like in Section 2.2, connected components of Frobenius algebra generators become single nodes in the combinatorial picture. In the next section, we will see that any cospan in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ arises from a string diagram, possibly with some Frobenius algebra generators, in this way.

Remark 3.12. For coloured PROPs, we have a similar situation, except that we get a different Frobenius algebra for each colour and we end up with hypergraphs whose nodes are labelled by the colours associated to those Frobenius algebras.

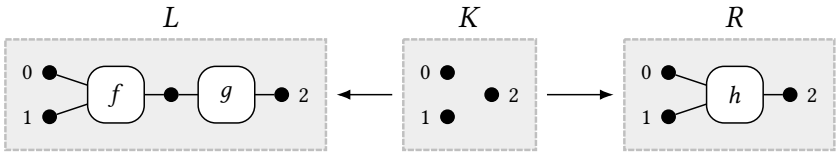
Fix a set of colours $C \in \mathbb{F}$. As shown in Theorem 2.24, $\mathrm{Csp}(\mathbb{F} \downarrow C)$ is isomorphic to the coproduct $\sum_{c \in C} \mathbf{Frob}_c$ consisting of one copy of **Frob** for each of the colours $c \in C$. We define a coproduct preserving, full and faithful functor D_C from $\mathbb{F} \downarrow C$ to the category of finite hypergraphs $\mathbf{Hyp}_{C,\Sigma}$, which sends a coloured set to its corresponding discrete (node-labelled) hypergraph.

This extends to a faithful coloured PROP homomorphism $\widetilde{D_C}$ in the same way as in Corollary 3.9. Hence, following Definition 3.10, we get an interpretation $[.]_C : \sum_{c \in C} \mathbf{Frob} \to \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$ from coloured Frobenius algebras into cospans of hypergraphs with coloured nodes.

3.3 DPO rewriting for hypergraphs

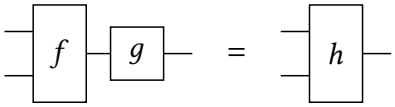
We now introduce the basics of DPO rewriting, applied to Σ-hypergraphs. We recall the DPO approach to rewriting applied to a category $\mathbb{C}$ with pushouts. A *DPO rule* is a span $L \leftarrow K \rightarrow R$ in $\mathbf{Hyp}_\Sigma$. L gives the LHS of the rule, R gives the RHS, whereas K gives the *invariant subgraph* associated with the rule. For our purposes, K will always be a discrete hypergraph consisting of the inputs and the outputs of the two sides of the rule.

Example 3.13. Consider the following hypergraph DPO rule

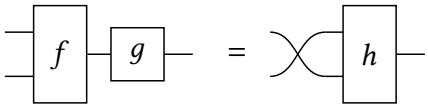


(12)

where the numbering is used to indicate how K embeds into L and R, respectively. While we will not establish a formal correspondence between string diagram equations and DPO rules until Section 4, one can see intuitively how this corresponds to the following equation between string diagrams



Namely, hyperedges play the role of boxes and vertices track how the boxes are connected to each other. The embeddings of K into L and R play an important role in the rule above: they maintain the correspondence between inputs/outputs on the LHS and inputs/outputs on the RHS. If we change these embeddings, we will change the rule. For example, if $K \to R$ in (12) is mapped $0 \mapsto 1, 1 \mapsto 0, 2 \mapsto 2$, the rule would become



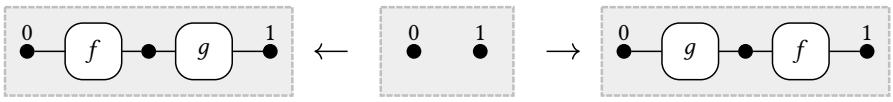
Remark 3.14. Note that if we considered L and R as cospans $2 \xrightarrow{i} L \xleftarrow{o} 1$ and $2 \xrightarrow{i'} R \xleftarrow{o'} 1$ following Section 3.2, $K \cong 2 + 1$, the coproduct of the input and the output sets. The maps from K to L and R are given by the induced copairings, $[i, o] : K \to L$ and $[i', o'] : K \to R$. This will play a role in the next section when we establish a formal correspondence between syntax and combinatorics.

A *DPO rewriting system* $\mathcal{R}$ is a set of DPO rules. Given hypergraphs G and H, we say that G *rewrites* into H —notation $G \rightsquigarrow_{\mathcal{R}} H$— if there exists $L \leftarrow K \to R$ in $\mathcal{R}$, object C and morphisms such that the squares below are pushouts

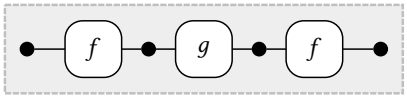
$$\begin{array}{ccccc} L & \longleftarrow & K & \longrightarrow & R \\ {\scriptstyle m}\downarrow & \ulcorner & \downarrow & \urcorner & \downarrow \\ G & \longleftarrow & C & \longrightarrow & H \end{array} \tag{13}$$

Typically the object C and the arrows $K \to C \to G$ are computed from $K \to L \xrightarrow{m} G$ in such a way that the left square above forms a pushout. In this case, the arrows $K \to C \to G$ are called a *pushout complement*. Hence, DPO rewriting can be seen as two distinct steps: first computing the pushout complement $K \to C \to G$, then pushing out the span $C \leftarrow K \to R$ to produce the rewritten object H. A *derivation* from G into H is a sequence of such rewriting steps.

Example 3.15. Consider a rewriting system with the following rule

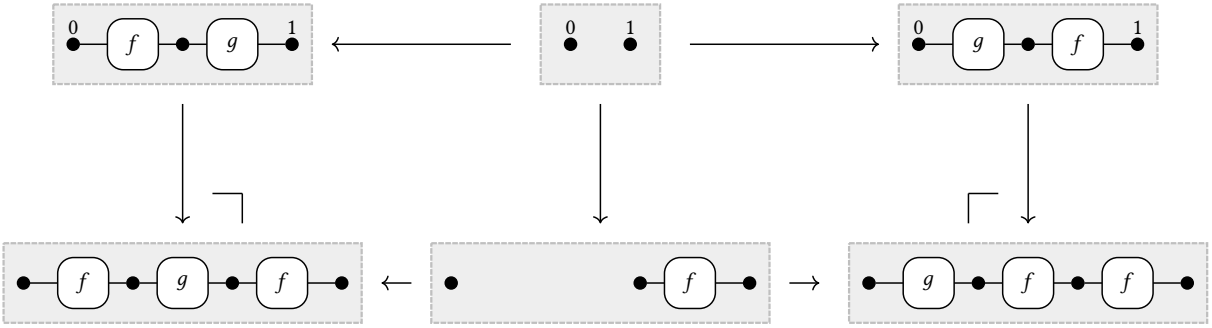


The rule has a matching into the following hypergraph



(14)

yielding the following DPO rewriting diagram



The above rewrite is the hypergraph equivalent of applying the rule $f; g \Rightarrow g; f$ to the term $f; g; f$ to obtain $g; f; f$.

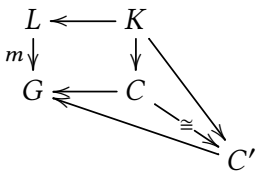
Pushout complements do not necessarily have to exist or be unique. For a fixed rule $L \leftarrow K \rightarrow R$, a *matching* $m : L \rightarrow G$ is a morphism with the additional property that the pushout complement of $K \rightarrow L \xrightarrow{m} G$ exists. For the example $\mathbb{C} := \mathbf{Hyp}_{C,\Sigma}$, we can give the precise conditions under which m is a matching, drawing e.g. from [CMR+97].

Definition 3.16. Morphisms $K \xrightarrow{i} L \xrightarrow{m} G$ satisfy the *no-dangling* condition if, for every hyperedge not in the image of m, every node of its source and target is either (i) not in the image of m or (ii) in the image of i ; m. They satisfy the *no-identification* condition if any two nodes merged by m are in the image of i.

Proposition 3.17. *A pushout complement of $K \xrightarrow{i} L \xrightarrow{m} G$ exists if and only if it satisfies the no-dangling and no-identification conditions.*

Uniqueness is a simpler story, as it only relies on i to be mono.

Proposition 3.18. *If $i : K \rightarrow L$ is mono, the pushout complement of $K \xrightarrow{i} L \xrightarrow{m} G$ is unique (up to commuting isomorphism), when it exists. That is, when C, C' are two pushout complements, there exists an isomorphism $C \xrightarrow{\cong} C'$ such that the following diagram commutes*



If i is not mono, the pushout may not be unique, see Section 4.5 below. In fact, the property holds for any adhesive category, of which labelled directed hypergraphs are an example [LS05]. The case where $K \rightarrow L$ is mono is important, because the matching m fully determines the resulting hypergraph H, due to the uniqueness of pushout complements. A rule $L \leftarrow K \rightarrow R$ is said to be *left-linear* if the morphism $K \rightarrow L$ is mono.

On the other hand, we will occasionally study rules that are *not* left-linear, in which case the pushout complement does not need to be unique. In the case of hypergraphs, all of the distinct pushout complements can still be effectively enumerated [HJKS11], which in our setting is closely related to enumeration of contexts when rewriting modulo the Frobenius equations. This will be covered in detail in Section 4.5.

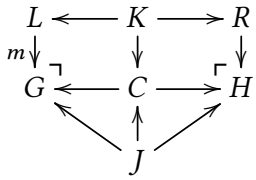
3.4 Double-Pushout Rewriting with Interfaces

Our first observation is that an extension of the traditional DPO rewriting, acting on hypergraphs *with interfaces*, actually fits best our purposes. This is for two main reasons. First, taking interfaces into account is essential to adequately map syntactic rewriting into DPO rewriting. Indeed, string diagrams are interpretable as *cospans* of hypergraphs, and the source and target of such cospans constitute the interface of the diagram. Second, DPO rewriting with interfaces (henceforth, DPOI) is of independent interest in the context of confluence. It is well-known [Plu10] that local confluence for DPO is undecidable. As shown in the sequel to this paper (after [BGK+17, BGK+21]), DPOI enjoys the Knuth-Bendix property: joinability of critical pairs is the same as local confluence. Hence, DPOI in this case closely matches standard term rewriting, whereas DPO is analogous to restricting rewriting to ground terms, where confluence is undecidable.

Remark 3.19. DPOI has appeared in different guises in the graph rewriting literature, such as rewriting with *borrowed contexts* [EK04], the graphical encodings of process calculi [Gad07, BGK09], and some foundational studies connecting DPO rewriting with computads in cospans [GH98, SS05].

Fix a category $\mathbb{C}$ with pushouts. We provide a definition of rewriting for morphisms $G \leftarrow J$ in $\mathbb{C}$. When $\mathbb{C}$ is $\mathbf{Hyp}_\Sigma$, we call them *(hyper)graphs with interface*. The intuition is that G is a hypergraph and J is an interface that allows G to be "glued" to a context.

Given $G \leftarrow J$ and $H \leftarrow J$ in $\mathbb{C}$, *G rewrites into H with interface J* — notation $(G \leftarrow J) \rightsquigarrow_{\mathcal{R}} (H \leftarrow J)$ — if there exist rule $L \leftarrow K \rightarrow R$ in $\mathcal{R}$ and object C with suitable morphisms in $\mathbb{C}$ such that the diagram below commutes, where the marked squares are pushouts



(15)

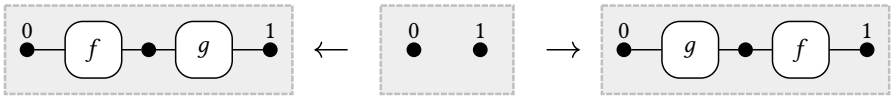
Hence, the interface J is preserved by individual rewriting steps.

Remark 3.20. Similar to the case of rewriting rules (cf. Remark 3.14) if G comes from a cospan representing a string diagram, the interface J is the coproduct of the inputs and the outputs.

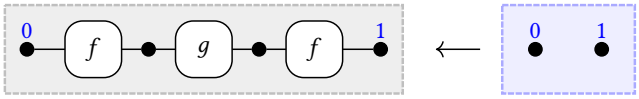
When $\mathbb{C}$ has a (strict) initial object 0 (e.g. in $\mathbf{Hyp}_\Sigma$, 0 is the empty hypergraph), ordinary DPO rewriting can be considered as a special case, by taking J to be 0. Like for traditional DPO, rewriting steps are considered up to isomorphism: $G_1 \leftarrow J : f_1$ and $G_2 \leftarrow J : f_2$ are isomorphic if there is an isomorphism $\varphi \colon G_1 \rightarrow G_2$ with $f_1 \,;\, \varphi = f_2$.

In Section 3.3 we said that a morphism is called a *match* if a suitable pushout complement exists. In DPOI, the condition for being a match is strictly stronger, as this pushout complement must furthermore respect the interface in (15).

Example 3.21. We now extend Example 3.15 to DPOI rewriting. We consider the same rewriting rule as before

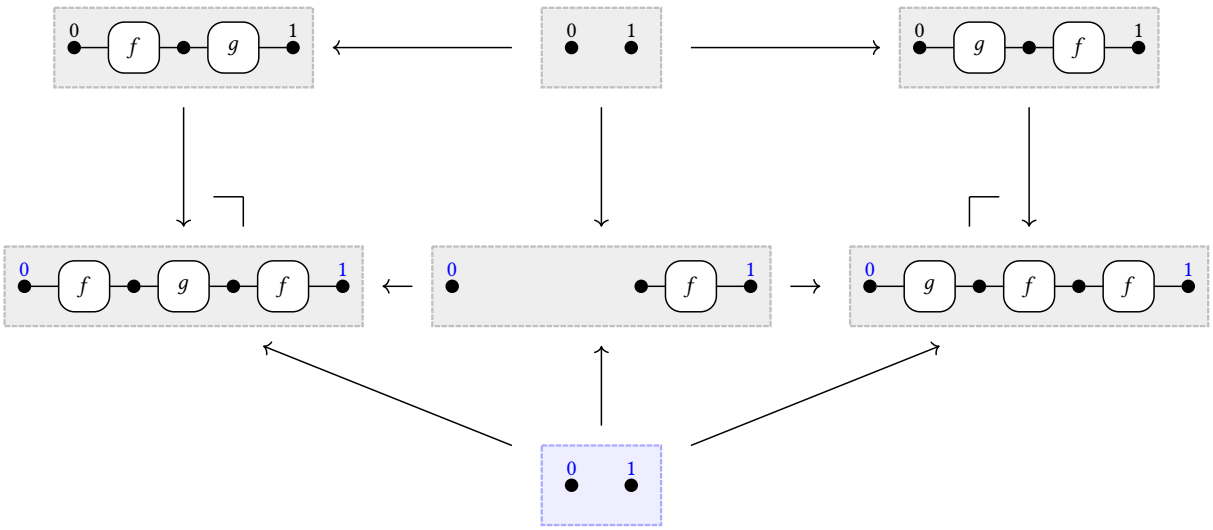


but now the target graph also has an interface

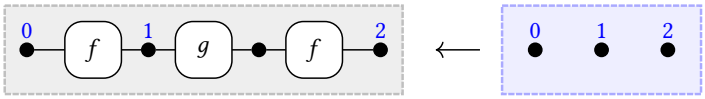


(16)

Hence, rewriting produces the following DPOI diagram, where the interface of the target graph is tracked on the bottom row



Note that, while the top two pushout squares in this diagram are identical to the ones in Example 3.15, the presence of an interface has an effect on whether a rewriting rule applies. For example, if the hypergraph in (16) is given the following, different interface



(17)

the LHS now has no match, as the new interface no longer factors through the pushout complement.

While the interface in (17) would not arise from a morphism in a generic monoidal category, it can arise when we include Frobenius algebra structure. Namely, it corresponds to one of the following diagrams

Whether it corresponds to the left or right diagram above depends on whether the node '1' came from an input or an output in the associated cospan (cf. Remark 3.20).

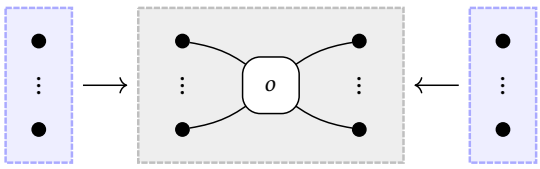
4 EQUIVALENCE OF SYNTAX AND COMBINATORICS

4.1 Terms as cospans

In Section 2, we defined $\mathbf{S}_\Sigma$ as the free PROP generated by a signature Σ. By freeness, we can completely define the PROP homomorphism

$$[\![\cdot]\!] : \mathbf{S}_\Sigma \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$$

by specifying how it acts on the generators in Σ. We do this by sending each generator $o : k \to l$ to a cospan $k \to O \leftarrow l$ where k, l are discrete hypergraphs and O is hypergraph containing a single o-labelled hyperedge connected to a single distinct node for each input and output. The cospan maps embed the inputs and outputs in the obvious way



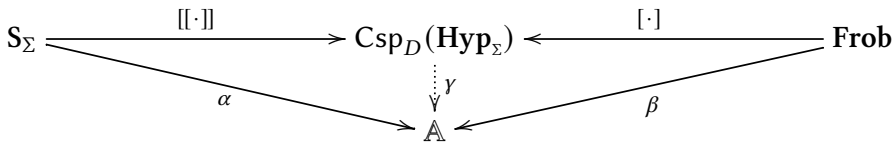
This mapping of generators extends by universal property to a PROP homomorphism $[\![\cdot]\!] : \mathbf{S}_\Sigma \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ sending any Σ-term in $\mathbf{S}_\Sigma$ to its associated composition of cospans.

4.2 Characterisation theorem for string diagrams

We now have all the ingredients needed to prove the equivalence of the free (i.e. syntactic) category $\mathbf{H}_\Sigma$ of string diagrams modulo Frobenius structure and its combinatoric representation $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$. We begin by noting that $\mathbf{H}_\Sigma$ is isomorphic to the coproduct of PROPs $\mathbf{S}_\Sigma + \mathbf{Frob}$, and in the following we will use them interchangeably, depending on what is most convenient. Then, we define $\langle\!\langle \cdot \rangle\!\rangle$ as the copairing in PROP of the faithful functors $[\![\cdot]\!] : \mathbf{S}_\Sigma \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ and $[\cdot] : \mathbf{Frob} \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$.

Theorem 4.1. *$\langle\!\langle \cdot \rangle\!\rangle : \mathbf{S}_\Sigma + \mathbf{Frob} \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ is an isomorphism of PROPs.*

Proof. It suffices to verify that $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ satisfies the universal property of the coproduct $\mathbf{S}_\Sigma + \mathbf{Frob}$ in PROP.



(18)

Given α, β, and a PROP $\mathbb{A}$ as in (18), we need to show the existence of a unique γ making the diagram commute. Now, because all morphisms in (18) are identity-on-objects, it suffices to show that any arrow of $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ can be decomposed in an essentially unique way into an expression where all the basic constituents lie in the image of $[[\cdot]]$ or $[\cdot]$.

To this aim, fix a cospan $n \xrightarrow{f} G \xleftarrow{g} m$ in $\mathrm{Csp}_D(\mathbf{Hyp}_{\Sigma,C})$, where G has set of nodes N, set of hyperedges E and a labelling function $\chi\colon E \to \Sigma$. We pick an order $e_1, \ldots, e_j$ on the hyperedges in E. Let $\tilde{n} \xrightarrow{i} \tilde{E} \xleftarrow{o} \tilde{m}$ be the cospan defined as $\bigoplus_{1\leq i\leq j}[[\chi(e_i)]]$. Intuitively, $\tilde{E}$ piles up all the hyperedges of G, but disconnected from each other. $\tilde{n}$ and $\tilde{m}$ are the concatenations of all the inputs, respectively outputs of these hyperedges.

There are obvious functions $f\colon n \to N$, $g\colon m \to N$, $j\colon \tilde{n} \to N$ and $p\colon \tilde{m} \to N$ mapping nodes to their occurrence in the set N of all nodes of G. All this information is now gathered in the following composition of cospans [1]

$$(n \xrightarrow{f} N \xleftarrow{(id,j)} N \oplus \tilde{n}) \;;\; (N \oplus \tilde{n} \xrightarrow{id\oplus i} N \oplus \tilde{E} \xleftarrow{id\oplus o} N \oplus \tilde{m}) \;;\; (N \oplus \tilde{m} \xrightarrow{(id,p)} N \xleftarrow{g} m) \tag{19}$$

Copairing maps (id, j) and (id, p) are well-defined as $\oplus$ is also a coproduct in $\mathbf{Hyp}_\Sigma$. One can compute that the result of composing (19) (by pushout) is indeed isomorphic to $n \xrightarrow{f} G \xleftarrow{g} m$.

Towards a definition of γ, we need to check that every component of (19) is in the image of either $[[\cdot]]$ or $[\cdot]$. First, the middle cospan is clearly in the image of $[[\cdot]]$, as it is the monoidal product of the identity cospan $w \xrightarrow{id} w \xleftarrow{id} w$ with $\tilde{n} \xrightarrow{i} \tilde{E} \xleftarrow{o} \tilde{m}$, which itself is the monoidal product of cospans each in the $[[\cdot]]$-image of some generator in Σ. Second, we have that the two outmost cospans are in the image of $[\cdot]$: this is because, by Definition 3.10 and Proposition 2.12, any morphism $u_1 \to u_3 \leftarrow u_2$ of $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$, with u_1, u_2, u_3 discrete, is in the image of $[\cdot]$.

Therefore, γ can be defined on $n \xrightarrow{f} G \xleftarrow{g} m$ by the values of $[[\cdot]]$ and $[\cdot]$ on its decomposition as in (19). This is a correctly and uniquely defined assignment: in the construction of the decomposition (19), the only variable parts are the different orderings that are picked for nodes and for hyperedges in $\tilde{E}$, but these are immaterial since all the involved categories are symmetric monoidal. □

We now observe two interesting consequences of this theorem. Recall from Section 2.5 that $\mathbf{H}_\Sigma$ is the free FROP over the signature Σ.

Corollary 4.2. $\mathbf{H}_\Sigma \cong \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$

Proof. $[\cdot]\colon \mathbf{Frob} \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ defines a FROP structure on $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ and the isomorphism of Theorem 4.1 extends to one of FROPs. Then the result follows by Proposition 2.12 and Theorem 4.1. □

The next corollary states that there is no 'information loss' in passing from the free PROP to the free FROP on Σ.

Corollary 4.3. $[[\cdot]]\colon \mathbf{S}_\Sigma \to \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ *is faithful.*

[1] Note in (19) N is seen both as a discrete hypergraph — when appearing in the carrier — and as an object of $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ — when appearing in the domain or codomain of a cospan. We did not emphasise this distinction to not overload notation.

Proof. We use the fact that coproducts of PROPs can be computed as pushouts in the category **SymCat** of small symmetric monoidal categories. In particular, $\mathbf{S}_\Sigma + \mathbf{Frob}$ in PROP arises as

$$\begin{array}{ccc} \mathbf{P} & \xrightarrow{!_2} & \mathbf{Frob} \\ {\scriptstyle !_1}\downarrow & & \downarrow{\scriptstyle [\cdot]} \\ \mathbf{S}_\Sigma & \xrightarrow[[[\cdot]]]{} & \mathbf{S}_\Sigma + \mathbf{Frob} \cong \mathrm{Csp}_D(\mathbf{Hyp}_\Sigma) \end{array} \tag{20}$$

in **SymCat**, where **P** is the category of finite sets and bijections (the initial object in the category PROP, see Example 2.25) and the maps $!_1$ and $!_2$ are given by initiality of **P**. Intuitively, in (20) $\mathbf{S}_\Sigma + \mathbf{Frob}$ is built as the disjoint union of the categories $\mathbf{S}_\Sigma$ and **Frob** where one identifies the two copies of each object $n \in \mathbb{N}$ and the symmetric monoidal structure of the two categories, i.e. the morphisms in the image of **P**.

Now, in order to prove that $[[\cdot]]$ is faithful, we can use a result [MS09, Th. 3.3] about amalgamation in **Cat** (which transfers to **SymCat**). As all the functors in (20) are identity-on-objects and $!_1$, $!_2$ are faithful, it just requires to show that $!_1$ and $!_2$ satisfy the so-called 3-for-2 property: for $!_1$, this means that, given $h = f\,;\, g$ in $\mathbf{S}_\Sigma$, if any two of f, g, h are in the image of $!_1$, then so is the third. This trivially holds as every arrow of **P** is an isomorphism. The argument for $!_2$ is identical. □

4.3 Characterisation theorem for coloured PROPs

Of crucial importance for the results in the latter half of this paper is that all of the results from the previous section extend in the obvious way to coloured PROPs. Notably, we can state a version of Theorem 4.1 for C-coloured PROPs, with $\langle\!\langle \cdot \rangle\!\rangle_C$ the copairing in CPROP, modulo the identification of the colours, of the faithful functors $[[\cdot]]_C \colon \mathbf{S}_{C,\Sigma} \to \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$ and $[\cdot]_C \colon \mathbf{Frob}_C \to \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$, the C-coloured versions of $[[\cdot]]$ and $[\cdot]$, respectively.

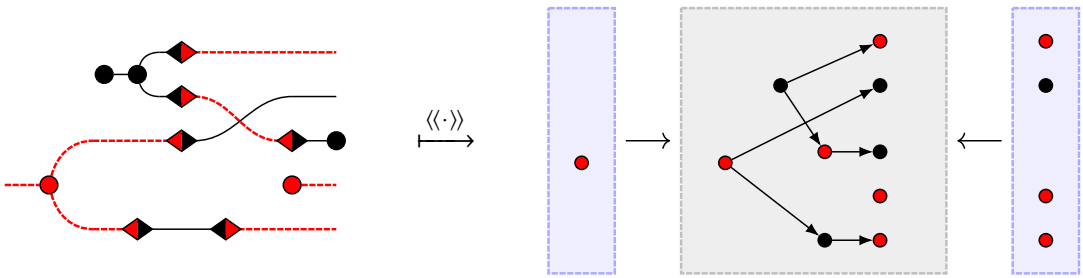
Proposition 4.4. *$\langle\!\langle \cdot \rangle\!\rangle_C \colon \mathbf{S}_{C,\Sigma} +_C \mathbf{Frob}_C \to \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$ is an isomorphism of C-coloured PROPs.*

It is also worth noting that the Frobenius structure identifies the hypergraphs with no hyperedges, i.e. the sets of C-labelled nodes.

Corollary 4.5. *There is an isomorphism of C-coloured PROPS $\mathbf{Frob}_C \cong \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\varnothing})$.*

As arrows of $\mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$ are the same thing as cospans in the slice category $\mathbf{FinSet} \downarrow C$, this corollary is essentially a restatement of Theorem 2.24.

Example 4.6. Consider the free FROP $\mathbf{H}_{C,\Sigma}$ for colours $C = \{\bullet, \bullet\}$ introduced in Example 2.27. By Corollary 4.2, $\mathbf{H}_{C,\Sigma} \cong \mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$. In hypergraphs of $\mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$, hyperedges correspond to switches or . Since these hyperedges are in fact edges (i.e. they have one input and one output node), we can draw them as such. Since they connect any two nodes only when these have a different colour, what we obtain are exactly finite directed bipartite graphs. For example, the combinatorial presentation of the diagram (8) from Example 2.27 is



This shows that Theorem 4.1 not only provides a combinatorial representation for algebraic structures, but conversely it allows us to give a purely algebraic representation for bipartite graphs.

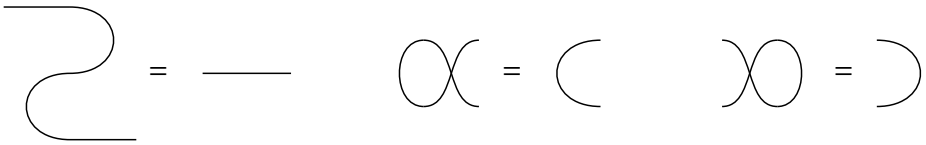
4.4 Characterisation theorem for rewriting

DPOI rewriting deals with hypergraphs $(G \leftarrow I)$ with a *single* interface. These may be seen as a particular class of morphisms in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$, namely those of the form $0 \to G \leftarrow I$, where 0 is the empty hypergraph (which also serves as the initial object of $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$). This means that DPOI rewriting can be meaningfully defined on morphisms of $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ with source 0. However, our semantics $\langle\!\langle \cdot \rangle\!\rangle$ maps diagrams of $\mathbf{S}_\Sigma + \mathbf{Frob}$ to cospans with *any* source. This gap can be overcome: in order to interpret syntactic rewriting as DPOI rewriting, we need an intermediate step in which we 'fold' the two interfaces m, n of a string diagram $a \colon m \to n$ into one $m + n$.

For cospans $m \xrightarrow{i} G \xleftarrow{o} n$, we already noted in Remark 3.20 that we can do this simply by using the copairing $[i, o] : m + n \to G$. We will now show that there is an equivalent syntactic operation to the passage from $m \xrightarrow{i} G \xleftarrow{o} n$ to $0 \to G \xleftarrow{[i,o]} m + n$. First, we note that we can use the Frobenius algebra structure in $\mathbf{H}_\Sigma$ to build 'cup' and 'cap' morphisms $\cup : 0 \to 2$ and $\cap : 2 \to 0$ as follows

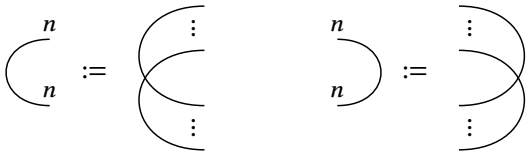
(21)

It follows from the Frobenius equations that the cup and cap maps satisfy the following equations



(22)

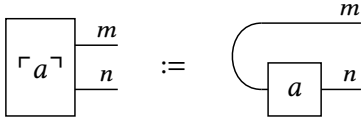
These equations capture the fact that the object 1 is equal to its own *dual*. This structure extends to cups and caps $\cup_n : 0 \to n + n$, $\cap_n : n + n \to 0$ for an arbitrary object n as follows



This collection of cups and caps for every object n endows a FROP with the structure of a compact closed category, which is furthermore coherently self-dual in the sense of [Sel10].

Remark 4.7. The fact that hypergraph categories (i.e. categories where every object is equipped with a chosen Frobenius algebra structure) are always compact closed was noted in [CW87], motivating the original name of *well-supported compact closed categories*.

We can use the 'cup' part of the compact structure to define an operation $\ulcorner \cdot \urcorner$ that bends all of the input wires around to become outputs. That is, for a map $a : m \to n$, we can form $\ulcorner a \urcorner : 0 \to m + n$ as $\cup_m ; (1_m \oplus a)$. Or, as a diagram

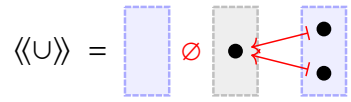


(23)

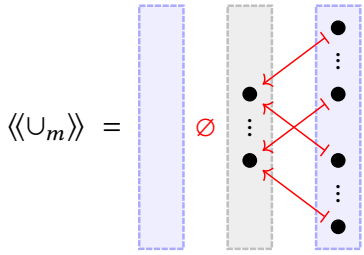
This operation is sometimes called forming the 'name' of a morphism [AC04] and can be inverted by post-composing with $\cap_m \oplus 1_n$ and applying the first equation in (22).

Proposition 4.8. *Suppose $\langle\!\langle a \rangle\!\rangle$ yields a cospan $m \xrightarrow{i} A \xleftarrow{o} n$, then $\langle\!\langle \ulcorner a \urcorner \rangle\!\rangle$ yields a cospan isomorphic to $0 \to A \xleftarrow{[i,o]} m + n$.*

Proof. From the definition of the cup map in terms of Frobenius structure, it follows that



and hence



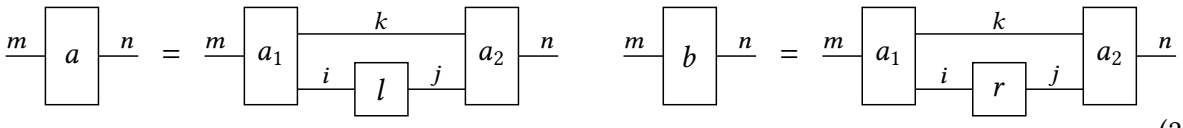
The result then follows from interpreting $\ulcorner a \urcorner := \cup_m ; (1_m \oplus a)$ as a composition of cospans and computing the result by pushout. □

Using $\ulcorner \cdot \urcorner$ will enable us to pass freely between syntactic rewriting of Σ-terms with non-trivial inputs and outputs to rewriting Σ-terms that only have outputs. We can do this both for the Σ-term being rewritten and for the rewriting rule itself. Namely, a rewriting rule $\langle l, r \rangle$ can be replaced by an equivalent (modulo Frobenius structure) rule $\langle \ulcorner l \urcorner, \ulcorner r \urcorner \rangle$. Such terms and rules having only a single interface (the outputs) will in turn correspond directly to DPOI rewriting under the interpretation functor $\langle\!\langle \cdot \rangle\!\rangle$. Putting these ingredients together, we are ready to state our main equivalence theorem for string diagram rewriting.

Theorem 4.9. *Let $\langle l, r \rangle$ be any rewriting rule on $\mathbf{S}_\Sigma + \mathbf{Frob}$. Then*

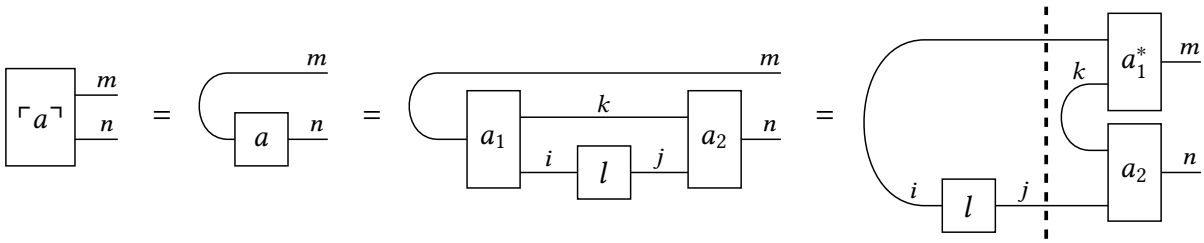
$$a \Rightarrow_{\langle l,r \rangle} b \quad \textit{iff} \quad \langle\!\langle \ulcorner a \urcorner \rangle\!\rangle \rightsquigarrow_{\langle\!\langle \langle \ulcorner l \urcorner, \ulcorner r \urcorner \rangle\!\rangle\!\rangle} \langle\!\langle \ulcorner b \urcorner \rangle\!\rangle \,.$$

Proof. On the direction from left to right, suppose that $a \Rightarrow_{\langle l,r \rangle} b$. Thus, by definition



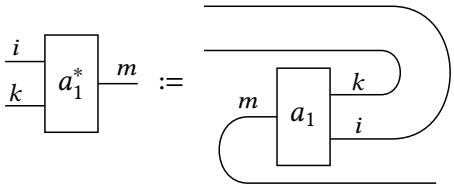
(24)

Using the Frobenius structure of $\mathbf{S}_\Sigma + \mathbf{Frob}$ we can put $\ulcorner a \urcorner$ in the following shape

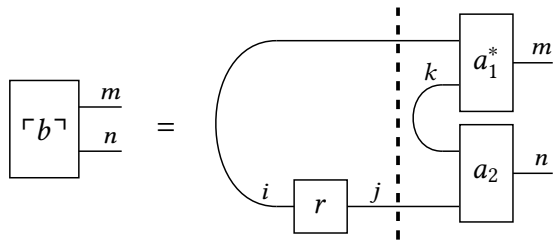


(25)

where



The dashed line in (25) decomposes the rightmost diagram into $\ulcorner l \urcorner \colon 0 \to i + j$ followed by a diagram of type $i + j \to m + n$, which we name $\tilde{a}$. With analogous reasoning



meaning that $\ulcorner a \urcorner = \ulcorner l \urcorner ; \tilde{a}$ and $\ulcorner b \urcorner = \ulcorner r \urcorner ; \tilde{a}$. (26)

Next, we introduce cospans giving semantics to the various diagrams

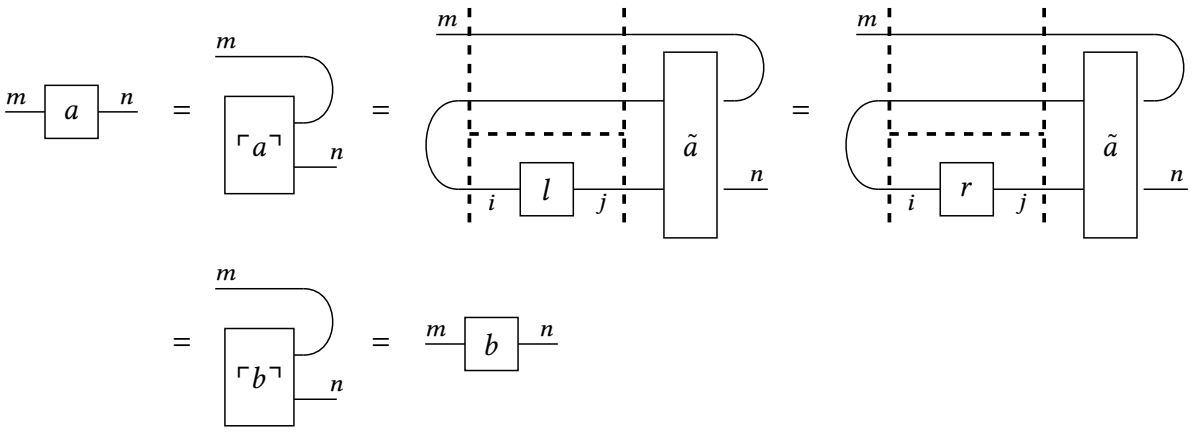
$$\begin{aligned} \langle\!\langle \ulcorner l \urcorner \rangle\!\rangle = 0 \to L \leftarrow i + j \qquad \langle\!\langle \tilde{a} \rangle\!\rangle = i + j \to C \leftarrow m + n \qquad \langle\!\langle \ulcorner r \urcorner \rangle\!\rangle = 0 \to R \leftarrow i + j \\ \langle\!\langle \ulcorner a \urcorner \rangle\!\rangle = 0 \to G \leftarrow m + n \qquad \langle\!\langle \ulcorner b \urcorner \rangle\!\rangle = 0 \to H \leftarrow m + n. \end{aligned} \tag{27}$$

Equation (26) tells that the cospan giving semantics to $\ulcorner a \urcorner$ (respectively, $\ulcorner b \urcorner$) is the composite of cospans giving semantics to $\ulcorner l \urcorner$ (respectively, $\ulcorner r \urcorner$) and $\tilde{a}$. As composition of cospans is by pushout, we obtain a double-pushout diagram as in (13) with $J = i + j$ and $I = m + n$, meaning that $\langle\!\langle \ulcorner a \urcorner \rangle\!\rangle \rightsquigarrow_{\langle\!\langle \ulcorner l \urcorner, \ulcorner r \urcorner \rangle\!\rangle} \langle\!\langle \ulcorner b \urcorner \rangle\!\rangle$.

We now conclude the proof by showing the right to left direction of the statement. Suppose that $\langle\!\langle \ulcorner a \urcorner \rangle\!\rangle \rightsquigarrow_{\langle\!\langle \ulcorner l \urcorner, \ulcorner r \urcorner \rangle\!\rangle} \langle\!\langle \ulcorner b \urcorner \rangle\!\rangle$. Naming cospans $\langle\!\langle \ulcorner a \urcorner \rangle\!\rangle$, $\langle\!\langle \ulcorner b \urcorner \rangle\!\rangle$, $\langle\!\langle \ulcorner l \urcorner \rangle\!\rangle$ and $\langle\!\langle \ulcorner r \urcorner \rangle\!\rangle$ as in (27), this implies by definition the existence of a pushout complement C yielding a DPOI diagram as (13) with $J = v_1 v_2$ and $I = w_1 w_2$. Now, pick $\hat{a} \colon v_1 v_2 \to w_1 w_2$ such that $\langle\!\langle \hat{a} \rangle\!\rangle = v_1 v_2 \to C \leftarrow w_1 w_2$, which exists by fullness of $\langle\!\langle \cdot \rangle\!\rangle$. Because composition in $\mathrm{Csp}_D(\mathbf{Hyp}_\Sigma)$ is by pushout, the existence of such a DPOI diagram yields

$$\begin{aligned} \langle\!\langle \ulcorner a \urcorner \rangle\!\rangle = (0 \to G \leftarrow w_1 w_2) = (0 \to L \leftarrow v_1 v_2) ; (v_1 v_2 \to C \leftarrow w_1 w_2) = \langle\!\langle \ulcorner l \urcorner \rangle\!\rangle ; \langle\!\langle \hat{a} \rangle\!\rangle \\ \langle\!\langle \ulcorner b \urcorner \rangle\!\rangle = (0 \to H \leftarrow w_1 w_2) = (0 \to R \leftarrow v_1 v_2) ; (v_1 v_2 \to C \leftarrow w_1 w_2) = \langle\!\langle \ulcorner r \urcorner \rangle\!\rangle ; \langle\!\langle \hat{a} \rangle\!\rangle. \end{aligned} \tag{28}$$

Because $\langle\!\langle \cdot \rangle\!\rangle$ is faithful, (28) yields decompositions $\ulcorner a \urcorner = \ulcorner l \urcorner ; \hat{a}$ and $\ulcorner b \urcorner = \ulcorner r \urcorner ; \hat{a}$ also on the syntactic side. This allows for a rewriting step $a \Rightarrow_{\langle l, r \rangle} b$ as below, where the dashed lines show how the syntactic matching (*cf.* the shape (6)) is performed



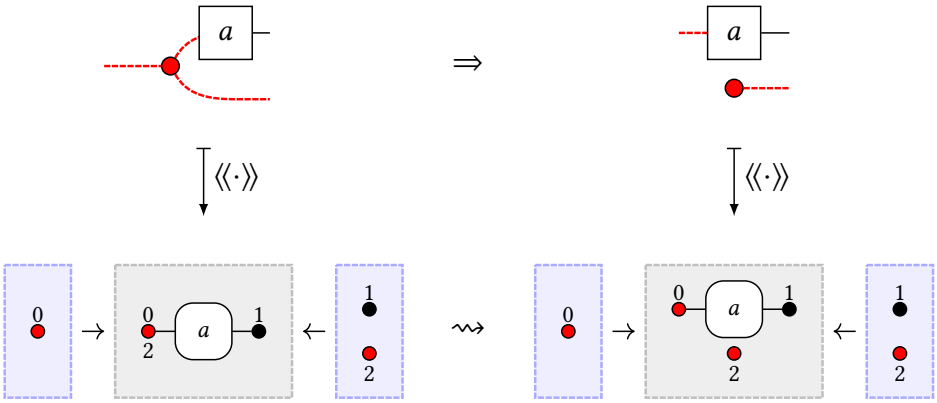
□

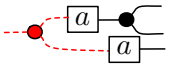
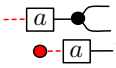
Furthermore, we can prove an analogous result for the rewriting of coloured PROPs, which thus subsumes and generalises the one for PROPs. Now, syntactic rewriting occurs in $\mathbf{S}_{C,\Sigma} +_C \mathbf{Frob}_C$, DPOI rewriting is in $\mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$, and the folding operation $\ulcorner \cdot \urcorner_C$ works as expected in the coloured setting. The result is stated below without proof.

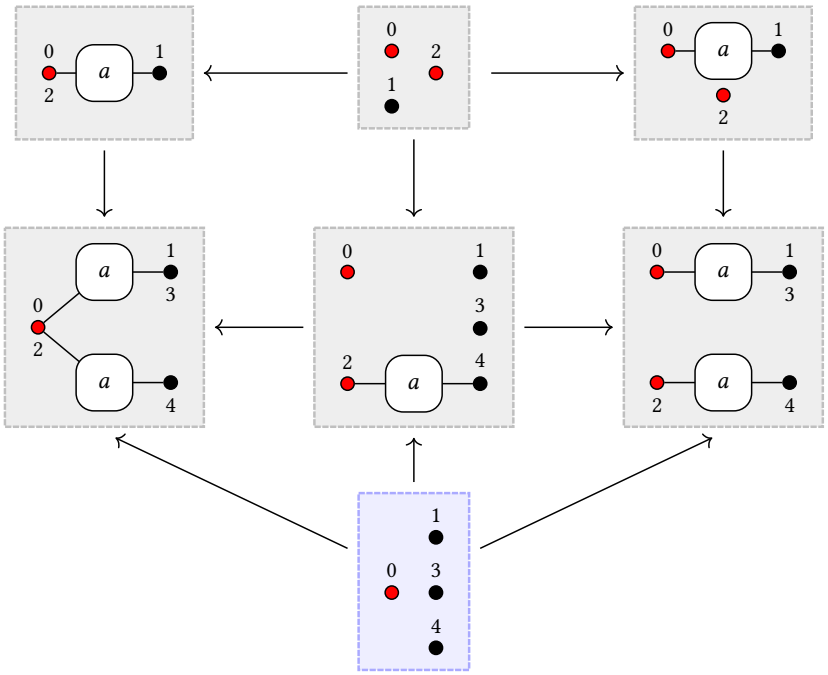
Proposition 4.10. *Let $\langle l, r\rangle$ be any rewriting rule on $\mathbf{S}_{C,\Sigma} +_C \mathbf{Frob}_C$. Then*

$$a \Rightarrow_{\langle l,r\rangle} b \quad \text{iff} \quad \langle\!\langle \ulcorner a \urcorner_C \rangle\!\rangle_C \rightsquigarrow_{\langle\!\langle \langle \ulcorner l \urcorner_C, \ulcorner r \urcorner_C \rangle \rangle\!\rangle_C} \langle\!\langle \ulcorner b \urcorner_C \rangle\!\rangle_C \,.$$

Example 4.11. We give an illustration of the correspondence of Proposition 4.10. Fix $C = \{\bullet, \bullet\}$ and $\Sigma = \{$ $\}$, and take the C-coloured PROP $\mathbf{S}_{\Sigma,C} +_C (\mathbf{Frob}_\bullet + \mathbf{Frob}_\bullet)$. We consider a rule α on such a PROP, together with its interpretation in $\mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$



We use numbers to indicate how the morphisms in the cospan are defined. Notice that these legs may be non-injective. Also, notice how the interpretation "absorbs" the Frobenius component. With the above rule, one can perform the syntactic rewriting step  $\Rightarrow_\alpha$  . It is implemented in $\mathrm{Csp}_{D_C}(\mathbf{Hyp}_{C,\Sigma})$ via the DPOI rewriting step below



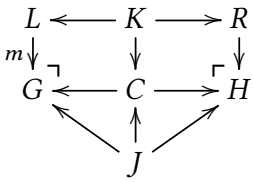
Note that the node labelled 2 in the resulting graph is not in the image of the interface (corresponding to a) and the node labelled by 1 and 3 is in the image of two nodes in the interface (corresponding to a). Hence, the outcomes of syntactic and DPOI rewriting coincide.

4.5 Pushout complements and rewriting modulo Frobenius

We conclude our discussion of the connection between syntactic and combinatoric rewriting by studying rule applications for which more than one pushout complement may exist. We will see

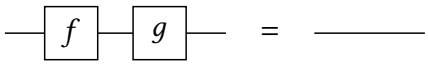
that, rather than a "bad" feature of combinatoric rewriting in this approach, this naturally subsumes the extra freedom one obtains in performing rewrites modulo Frobenius structure.

As discussed in Section 3.3, in adhesive categories such as $\mathbf{Hyp}_\Sigma$, pushout complements for $K \to L \to G$ are guaranteed to be unique only when the morphism $K \to L$ is mono. If we have a look again at a DPOI rewrite from Section 3.4



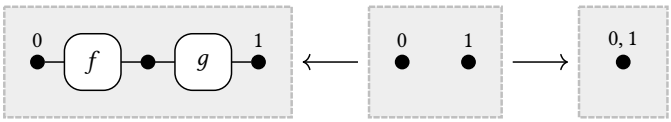
(29)

it follows that the rewritten graph H is totally determined by the map $L \to G$ only in the case where the mapping of K into the LHS L of the rule is mono. However, when rewriting modulo Frobenius structure, there are cases where this is not true. For example, consider the relatively harmless-looking equation that captures the fact that f is a one-sided inverse of g



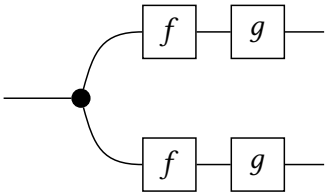
(30)

If we translate this into a hypergraph rewriting rule from the LHS to the RHS, we have no problem, because the embedding of K into L is mono

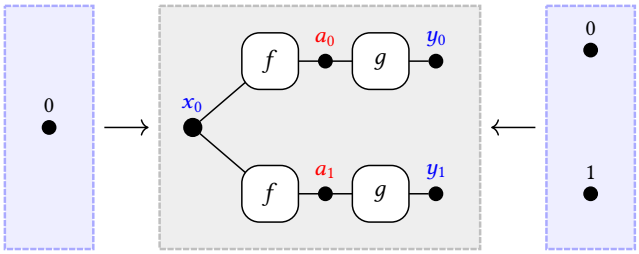


(31)

Hence, a rewriting step involving this rule will be uniquely fixed by the matching m. Consider applying the rule above to this diagram

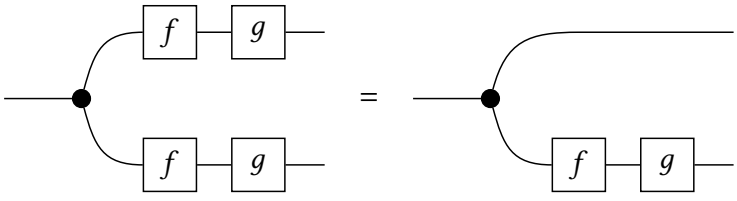


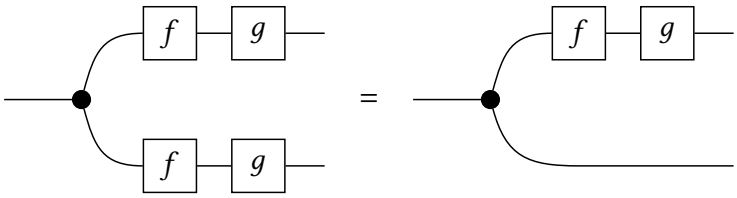
which corresponds to this cospan of hypergraphs



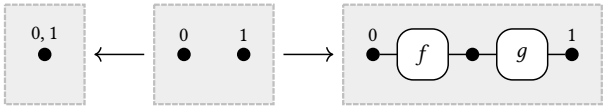
(32)

Then, we see the LHS of the rewriting rule (31) has exactly two matchings, corresponding to the upper and lower copies of f and g, respectively. Since the embedding of the boundary into the LHS of (31) is mono, each of these matchings corresponds to a unique way to rewrite the diagram



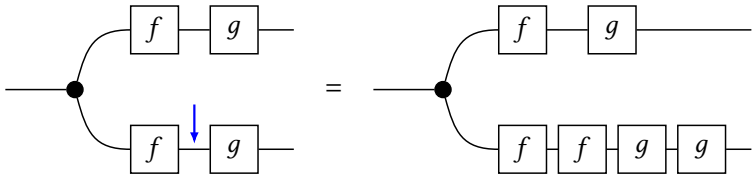


However, the identity wire in equation (30) is captured as a single node in the image of two nodes in the boundary of the rewriting rule: one corresponding to the input of the wire and one corresponding to the output. Hence, if we consider equation (30) instead as a rewriting rule from the RHS to the LHS, we obtain the following span of hypergraphs



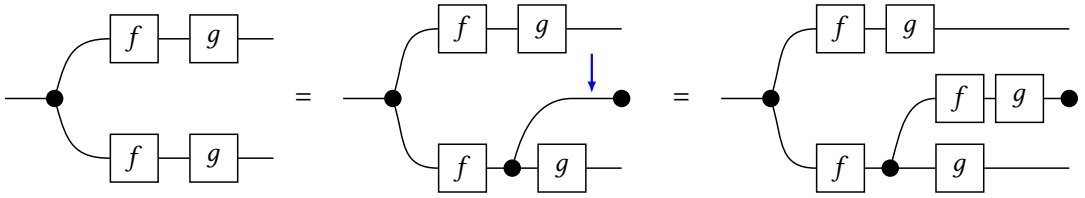
(33)

Now, there is a matching of the LHS of rule (33) for each of the nodes in (32). Unlike before, a single matching corresponds to multiple ways to rewrite the diagram, modulo Frobenius. For the matching m whose image is the node a_1, one possible rewrite corresponds to the obvious one, where we expand the wire between f and g to contain another copy of f and g



(34)

But this is not the only rewrite we can perform at this location. We can also apply some Frobenius algebra equations to create another, distinct rewrite here



(35)

Each of these corresponds to a DPO rewrite with the same matching m, but with a different *context*. That is, a different graph C in the DPO diagram (29) is chosen to complete the left pushout square. It is not clear *a priori* how many such contexts exist, or even if the collection of possible contexts is even finite, up to isomorphism.

Thankfully, this question has already been answered for hypergraphs by [HJKS11], which shows that the set of pushout complements for a given pair of morphisms $K \longrightarrow L \longrightarrow G$ is finite whenever K, L and G are so, and gives a method for enumerating them. We now will briefly sketch how this method works and apply it to the example above, referring the interested reader to [HJKS11] for the relevant proofs.

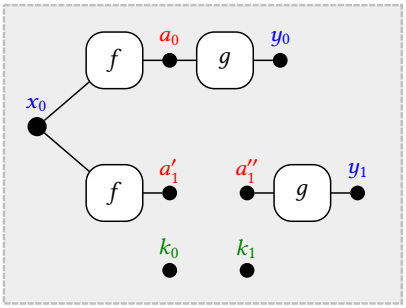
We begin by constructing an "exploded" context $K + \widetilde{G}$, which is the disjoint union of K and a new graph $\widetilde{G}$ defined as follows

(1) add one vertex to $\widetilde{G}$ for every vertex $v \in G$ not in the image of m,
(2) add one hyperedge to $\widetilde{G}$ for every hyperedge $h \in G$ not in the image of m, and
(3) for each hyperedge $h \in \widetilde{G}$, let the i-th source $\widetilde{s}_i(h)$ be $s_i(h)$ if $s_i(h) \in \widetilde{G}$, and otherwise let it be a new, fresh vertex; define the targets similarly.

Then, [HJKS11] showed that any pushout complement C arises as a quotient of this exploded context. That is, we obtain a morphism $n : K \to C$ by composing the embedding of K into $K + \widetilde{G}$ with the quotient map. If pushing out n along $K \to L$ gives our matching $m : L \to G$, we have a valid pushout complement.

In principle, there are exponentially many possible quotients of this graph to consider. However thanks to the fact that the quotient of $K + \widetilde{G}$ must yield a valid pushout complement, we typically only need to consider a small fraction of the possible quotients. By construction of the exploded context, we have an induced map $q : K + \widetilde{G} \to G$ that sends nodes and hyperedges in $\widetilde{G}$ which came from G to themselves and nodes in K to their image under $K \to L \to G$. It was shown in [HJKS11] that all of the pushout complements arise from q by identifying nodes in the fibres of q, i.e. sets of the form $q^{-1}(v)$ for nodes $v \in G$. We can therefore enumerate contexts by looking at all the possible quotients of non-trivial fibres of q and seeing which of them push out to the correct graph. Hence, if q only has a small number of non-trivial fibres, and the fibres themselves are also relatively small, it is practical to enumerate all of the possible contexts.

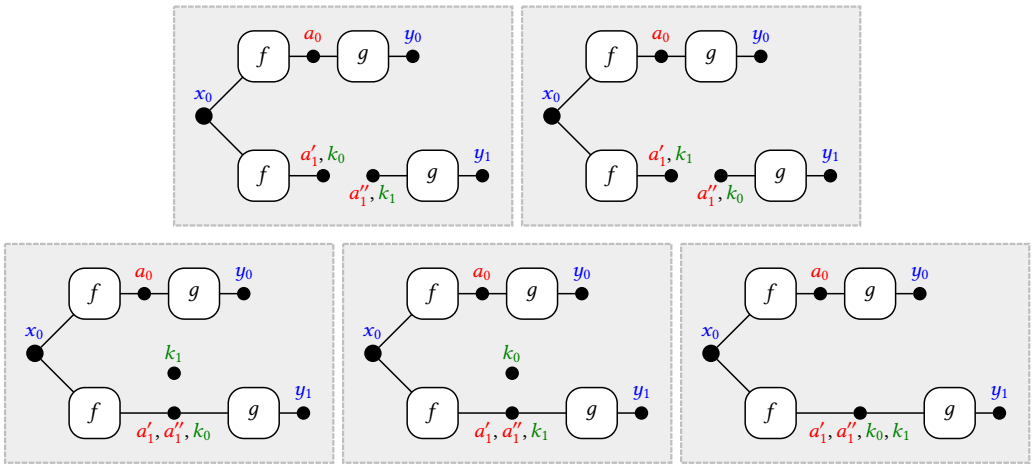
To see how this works on our example, we first compute the exploded context for the matching m whose image is the vertex a_1 in (32)



(36)

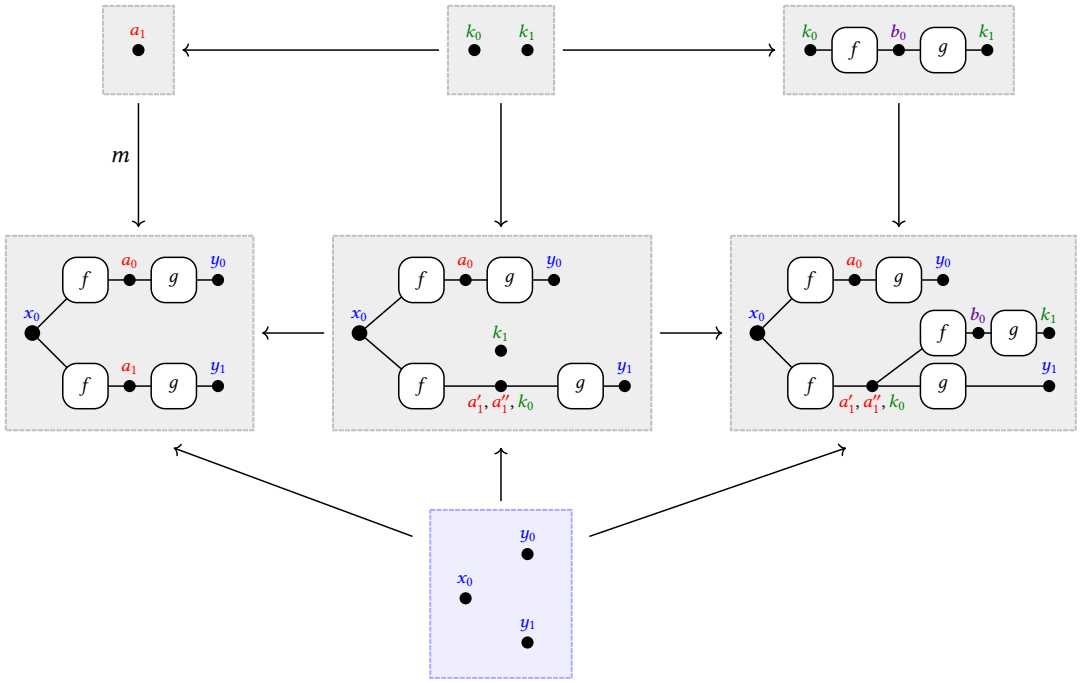
The vertices $\{k_0, k_1\}$ come from K, and the vertices $\{a'_1, a''_1\}$ are new, fresh copies of the vertex a_1, which was in the image of m.

We can form candidate contexts for m as quotients of the graph above. By the discussion above, we can conclude that the only allowed quotients of (36) are the ones that identify nodes in the only non-trivial fibre of $q : K + \widetilde{G} \to G$, which is $q^{-1}(a_1) = \{a'_1, a''_1, k_0, k_1\}$. Of the 15 possible partitions for the set $q^{-1}(a_1)$, five will yield a map $n : K \to C$ that pushes out to give $m : L \to G$, as required. The rest will result in a graph G' with strictly more nodes that G. The five good candidates for C are the following

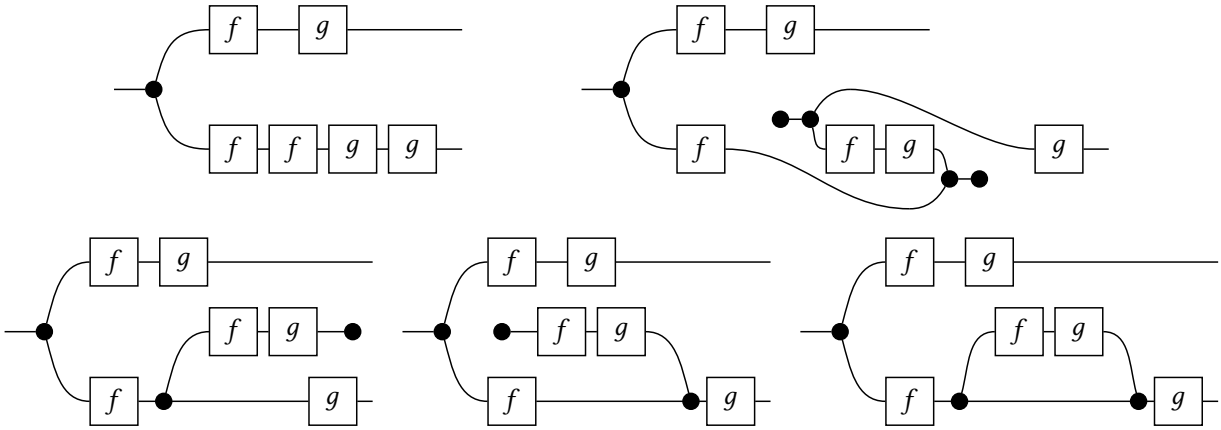


(37)

Note that, even though we see the same context graph multiple times, they differ in the embedding $n : K \to C$, as indicated by the labels. The first context yields the "obvious" rewrite (34), whereas if we compute the DPOI diagram for the third context, we obtain (35)



If we perform this rewriting step for each of the five different contexts in (37), we obtain five different results



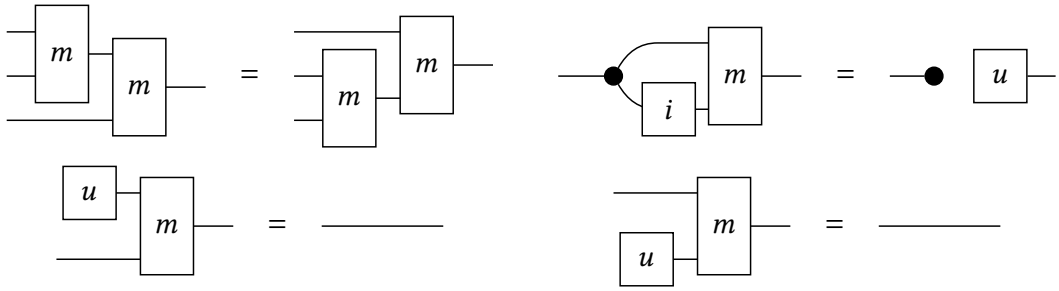
Thanks to Theorem 4.9, we can find all of the distinct ways of rewriting a diagram with a given rule, modulo Frobenius, by enumerating all of the matchings, then enumerating all of the valid contexts.

Finally, note that even though the mapping $K \to L$ in rule (33) is not mono, there is only one context in (37) that yields a sound rewrite for matching m in the absence of Frobenius structure. This is a general feature of string diagram rewriting *without* Frobenius structure, which we will revisit in the second paper in this series [BGK+20].

5 EXAMPLE: GROUP ALGEBRAS

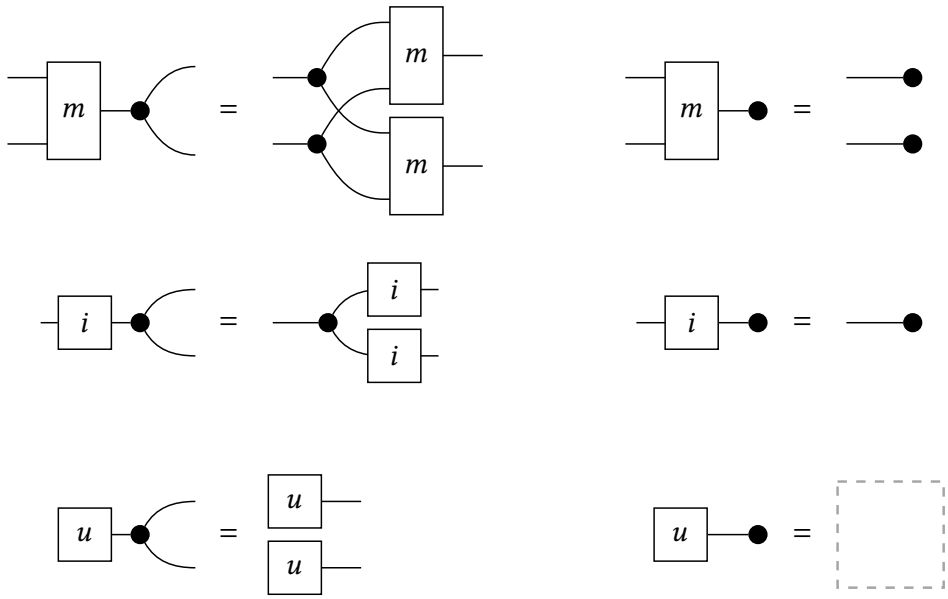
In order to showcase the techniques we have introduced, we now study a string diagram rewriting system relevant to the representation theory of finite groups. We introduce boxes $m : 2 \to 1, i :$

$1 \longrightarrow 1, u : 0 \longrightarrow 1$ satisfying the following associativity, inverse, and unit laws



(38)

which are furthermore "natural" with respect to (,) in the following sense



(39)

Remark 5.1. Note that the Frobenius structure is used to allow expressions that refer to inputs in a non-linear fashion. For instance, if one thinks of the input wire of the top-right equation in (38) as a variable x, this roughly corresponds to the term equation $m(x, i(x)) = u$. Notably, the lefthand-side refers to x twice (indicated by "copying" the input with) and the rightside-side refers to x zero times (indicated by "deleting" the input with).

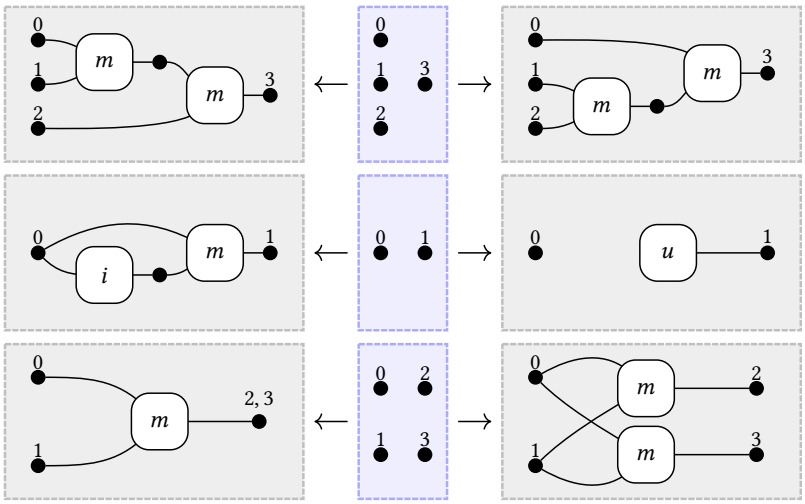
Let $(\mathbf{Vect}_k, \otimes)$ be the symmetric monoidal category of finite-dimensional vector spaces and linear maps with the tensor product. The models of the SMT defined above are an important structure in the study of finite groups.

Proposition 5.2. *For $\Sigma = \{m, i, u\}$ and $\mathcal{E}$ given by the equations (38) and (39), the models of $(\Sigma, \mathcal{E})$ in $(\mathbf{Vect}_k, \otimes)$ are group algebras. That is, a model consists of vector space V spanned by the elements of a finite group G and (bi)linear maps $m : V \otimes V \longrightarrow V$, $i : V \longrightarrow V$, $u : k \longrightarrow V$ defined on the basis of group elements as follows*

$$m(g \otimes h) = gh \qquad i(g) = g^{-1} \qquad u(1) = e \tag{40}$$

Proof. *(Sketch)* This follows from the fact that fixing a (special commutative) Frobenius algebra over a finite-dimensional vector space is equivalent to fixing a basis of vectors "copied" by $\delta :=$, i.e. satisfying $\delta(v) = v \otimes v$ (see [CPV13]). Then, equations (39) imply that m, u, i send basis elements to basis elements, and the equations (38) imply the usual associativity, unit, and inverse laws of a group. For details, see e.g. [Kas12] or [CK17]. □

Now, applying the techniques introduced in the previous section, we can translate the 10 equations (38) and (39) into combinatoric form. For example, the associativity, inverse, and first naturality equation translate as follows

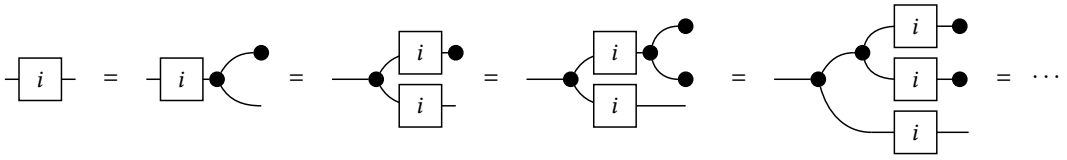


(41)

The remaining 7 equations are similar.

We can use these rules to obtain a terminating rewriting strategy for normalising expressions over $\Sigma = \{m, u, i\}$, at least in the case of acyclic hypergraphs. Intuitively, we eliminate as many u and i boxes as possible, associate the m boxes to the right, and push m, u and i boxes as far toward the outputs as possible.

First, we note that the rules (39) can have bad behaviour if they are applied in a completely naïve fashion modulo Frobenius. For example, applying the naturality rule for i, we can get infinite chains like this



(42)

However, such rewrites are not making any progress in "pushing i towards the outputs". We can fix this problem by constraining the rewriting with using a total order than measures progress and only accept rewrites that are decreasing with respect to that order.

Whereas this is difficult to do in a purely syntactic fashion, it is fairly straightforward to define "progress" using the combinatoric structure. First, we will introduce a number that measures how far a given hyperedge is from an output.

Definition 5.3. For a hyperedge h in an acyclic hypergraph with interface $G \xleftarrow{b} K$, let p be the shortest path connecting h to a node that is not in the source of any hyperedge (which by acyclicity always exists). The *branching depth* of h is the sum of the *branching degree* of every node in p, where the branching degree of node v is $\max(0, \deg(v) + |b^{-1}(v)| - 2)$.

Intuitively, the branching degree of a node captures how many and maps a single node represents, and the branching depth captures a notion of distance from an output (or a), measured in terms of those maps.

The applications of the rules obtained from (39) that "make progress" do so by replacing a single hyperedge h with (possibly) multiple hyperedges whose branching depth is strictly smaller than h. Conversely, the "bad" sequences we want to rule out, such as (42), generate more hyperedges without decreasing the branching depth of the original one.

Let $\mathcal{D}(G) \in \mathbb{N}^\star$ be a word where $\mathcal{D}(G)_k$ is the number of hyperedges with branching depth k. By comparing words of the form $\mathcal{D}(G)$ lexicographically, we obtain a total order on hypergraphs that measures "progress" in the sense we are after.

Definition 5.4. For a totally ordered set A, the reverse lexicographic order $\prec_\ell$ is defined recursively on the set $A^\star$ of words as follows: $u \prec_\ell v$ if

- $|u| < |v|$, or
- $|u| = |v| = k$ and $u_k < v_k$, or
- $|u| = |v| = k$, $u_k = v_k$, and $u_1 ... u_{k-1} \prec_\ell v_1 ... v_{k-1}$.

Reduction Strategy *We begin with an acyclic hypergraph with interface $G \leftarrow J$. The graph is normalised with respect to rules obtained from (38) and (39) as follows*

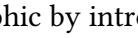
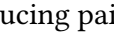
(1) pick a rule from (38) and apply it if possible;
(2) pick a rule from (39) and apply it if possible, subject to the condition that $\mathcal{D}(G)$ is strictly reduced with respect to $\prec_\ell$;
(3) repeat until no rule can be applied in steps 1 and 2.

Theorem 5.5. *The **Reduction Strategy** above terminates.*

Proof. The unit and inverse laws from (38) strictly decrease the number of hypereges in G, which will strictly decrease $\mathcal{D}(G)$. By construction, applying the rules in (39) must strictly decrease $\mathcal{D}(G)$. The associativity law strictly decreases the number of m-labelled hyperedges connected to the left (i.e. 'upper') input of another m-labelled hyperedge, while leaving $\mathcal{D}(G)$ fixed. Since all of these quantities are finite and bounded below, the reduction strategy must terminate. □

6 REWRITING MODULO MULTIPLE FROBENIUS ALGEBRAS

The previous sections have shown that the Frobenius equations are an intrinsic part of the structural rules of diagrammatic theories, distinguished from the domain-specific equations in a rewriting system. However, the hypergraph representation can absorb the structure of a Frobenius algebra, but only one per sort. This is in contrast with many applications of Frobenius algebra rewriting – e.g. in quantum theory [CD08, CK17], graphical linear algebra [Sob], concurrency theory [BHP+19], circuits [BPSZ19], and control theory [BSZ14b, BE15] – which typically deal with multiple, interacting Frobenius algebras on a single sort.

In order to deal elegantly with multiple Frobenius algebras on a single sort, we need an intermediate step, moving from a single-sorted to a coloured setting. Concretely, our starting domain will be a PROP $\mathbb{C}$, with n Frobenius algebras. From this, we build a n-coloured PROP $\mathbb{D}_\Upsilon$, where each colour carries one of the n different Frobenius structures. We then make each of the colours formally isomorphic by introducing pairs of "colour-switch" maps (e.g.  and ) and imposing equations making them inverses to each other.

We shall then prove that $\mathbb{C} \to \mathbb{D}_\Upsilon$ is actually an *equivalence* of coloured PROPs, meaning that $\mathbb{D}_\Upsilon$ is a faithful representation of the information carried by $\mathbb{C}$. By working in the multi-sorted setting provided by $\mathbb{D}_\Upsilon$, we can now exploit the correspondence established in Proposition 4.10. However, we will need a further step: the elimination of redundant colour-switch maps after the application of a rule. This is obtained by normalising graphs with respect to the confluent and terminating rewriting system Υ that removes pairs of inverse colour-switch maps. Thus our implementation establishes a new correspondence stating that "Rewriting PROPs with C Frobenius algebras" corresponds to "DPO rewriting of hypergraphs with C-sorted nodes, in Υ-normal form."

6.1 The Polychromatic Interpretation

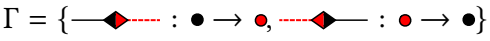
Throughout this and the next section we fix a PROP

$$\mathbb{C} := \mathbf{S}_\Sigma + \mathbf{Frob} + \mathbf{Frob}$$

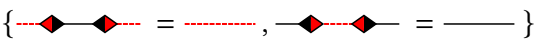
freely generated by a signature Σ and two Frobenius algebras (*cf.* Section 2.6), together with a rewriting system $\mathcal{R}$ on $\mathbb{C}$. Our goal is to provide a DPOI implementation for $\mathcal{R}$-rewriting in $\mathbb{C}$.

Remark 6.1. Even though our exposition deals with rewriting modulo two Frobenius algebras, this is just for simplicity. The theory works for an arbitrary number of Frobenius algebras, via a straightforward generalisation of the developments presented in this section.

Towards this goal, this section provides the intermediate step of representing $\mathbb{C}$ in terms of a coloured PROP $\mathbb{D}_\Upsilon$; this setup will make our diagrammatic theory adapted to DPOI rewriting, via Proposition 4.10. $\mathbb{D}_\Upsilon$ is defined as follows. Consider a signature of "colour conversion" operations Γ (*cf.* Remark 2.26 and Example 2.27), which we denote graphically as

$\Gamma =$ 

together with equations

$\Upsilon =$ 

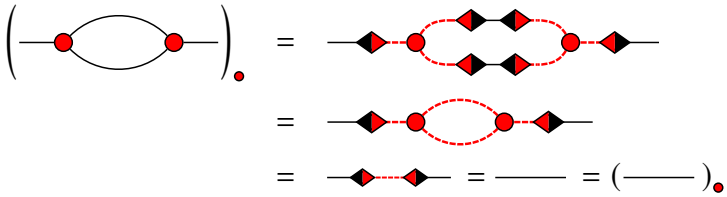
Then $\mathbb{D}$ is defined as the $\{\bullet, \bullet\}$-coloured PROP

$$\mathbb{D} := \mathbf{S}_{\{\bullet,\bullet\},\Sigma\uplus\Gamma} +_{\{\bullet,\bullet\}} (\mathbf{Frob}_\bullet + \mathbf{Frob}_\bullet) \tag{43}$$

and $\mathbb{D}_\Upsilon$ as $\mathbb{D}$ quotiented by Υ. Notice that $\mathbb{D}_\Upsilon$ is generated by the same signature Σ as $\mathbb{C}$, including operations on sort $\bullet$, but also from the colour conversion operations in Γ. Whereas the two Frobenius structures in $\mathbb{C}$ were on the same sort, the two in $\mathbb{D}_\Upsilon$ are on two different sorts: $\bullet$ and $\bullet$. We use $+_{\{\bullet,\bullet\}}$ (*cf.* Example 2.25) to identify the sorts of $\mathbf{Frob}_\bullet + \mathbf{Frob}_\bullet$ with those of $\mathbf{S}_{\{\bullet,\bullet\},\Sigma\uplus\Gamma}$.

We now define the "polychromatic interpretation" $(\cdot)_\bullet \colon \mathbb{C} \to \mathbb{D}_\Upsilon$. Intuitively, $(\cdot)_\bullet$ will "shift" one of the two Frobenius structures of $\mathbb{C}$ from sort $\bullet$ to sort $\bullet$, so that each sort hosts a single Frobenius algebra. Formally, $(\cdot)_\bullet \colon \mathbb{C} \to \mathbb{D}_\Upsilon$ is a morphism of coloured props, where $\mathbb{C}$ is here seen as a $\{\bullet\}$-coloured PROP. It suffices to define $(\cdot)_\bullet$ on the generating objects and arrows of $\mathbb{C}$. For objects, the single sort $\bullet$ of $\mathbb{C}$ is mapped to $\bullet$. For arrows, $(\cdot)_\bullet$ acts as the identity with the exception of the generators of the second Frobenius algebra

Notice that equations Υ are needed in order for this functor to be well-defined. For instance, they ensure preservation of the separability law for the "red" Frobenius algebra



Remark 6.2. As $(\cdot)_\bullet$ has been defined in terms of the generators of $\mathbb{C}$ and $\mathbb{D}$, it will sometimes be useful to regard, by abuse of notation, $(\cdot)_\bullet$ as a mapping from formal string diagrams of the generators of $\mathbb{C}$ to morphisms in $\mathbb{D}$. It is worth noting that this mapping would not extend to a well-defined functor from $\mathbb{C}$ to $\mathbb{D}$, since the latter is missing the equations of Υ.

It is essential for our developments that the polychromatic interpretation is without loss (or gain) of information. This is guaranteed by the following result.

Proposition 6.3. $(\cdot)_\bullet$ *induces an equivalence* $\mathbb{C} \simeq \mathbb{D}_\Upsilon$ *in* CPROP.

Proof. We have already shown that $(\cdot)_\bullet$ gives a strict monoidal functor from $\mathbb{C}$ to $\mathbb{D}_\Upsilon$. We define another functor $K : \mathbb{D}_\Upsilon \to \mathbb{C}$ on objects by letting $K(\circ) = K(\bullet) = \bullet$. Since $\mathbb{D}_\Upsilon$ is presented by generators and equations, it suffices to say what it does on generators of $\mathbb{D}_\Upsilon$. It sends the generators Σ and the two Frobenius algebras to their monochromatic versions, whereas it sends each of the two colour-changers in Γ to $\mathrm{id}_\bullet$. One can straightforwardly check that this gives a well-defined, strict monoidal functor and that $K((\cdot)_\bullet) = Id_{\mathbb{C}}$. So, it remains only to give a natural isomorphism $\kappa : Id_{\mathbb{D}_\Upsilon} \cong (K(\cdot))_\bullet$.

For a word w in $\{\bullet, \circ\}$, $(K(\cdot))_\bullet = \bullet^{|w|}$. So, let $\kappa_w : w \to \bullet^{|w|}$ be the unique monoidal product of $\mathrm{id}_\bullet$ and morphisms of the correct type. This is an isomorphism by construction. Naturality then follows from the definition of $(\cdot)_\bullet$ and the equations Υ. □

Remark 6.4. The construction in this section 'splits the difference' between the two coproducts discussed in Section 2.6. As noted there, the embedding $U : \mathrm{PROP} \to \mathrm{CPROP}$ does not preserve coproducts. However, we can consider the introduction of the colour changers and equations Υ as a weak truncation operation on coloured PROPs $(\cdot)^\bullet$, which forces all of the colours to be isomorphic to $\bullet$. Then, we do indeed have an equivalence of coloured PROPs $U(\mathbb{A} + \mathbb{A}') \simeq (U(\mathbb{A}) + U(\mathbb{A}'))^\bullet$. Proposition 6.3 is then the instantiation of this fact for $\mathbb{A} := \mathbf{S}_\Sigma + \mathbf{Frob}$ and $\mathbb{A}' := \mathbf{Frob}$.

6.2 Interpreting the Rewriting

Now that we have represented $\mathbb{C}$ as a coloured PROP $\mathbb{D}_\Upsilon$, we can pass to hypergraphs by instantiating Proposition 4.4.

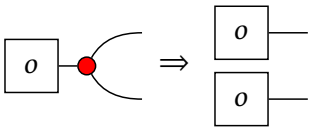
Corollary 6.5. *There is an isomorphism of* $\{\bullet, \circ\}$*-coloured PROPs between* $\mathrm{Csp}_{F_{\{\bullet,\circ\}} I_{\{\bullet,\circ\}}}(\mathbf{Hyp}_{\{\bullet,\circ\},\Sigma \uplus \Gamma})$ *quotiented by* $\langle\langle \Upsilon \rangle\rangle_{\{\bullet,\circ\}}$ *and* $\mathbb{D}_\Upsilon$.

With respect to rewriting, Corollary 6.5 is still unsatisfactory: rewriting in $\mathbb{D}_\Upsilon$ (and thus in $\mathbb{C}$) corresponds to DPOI rewriting in $\mathrm{Csp}_{F_{\{\bullet,\circ\}} I_{\{\bullet,\circ\}}}(\mathbf{Hyp}_{\{\bullet,\circ\},\Sigma \uplus \Gamma})$ only modulo the equations Υ. Clearly, it would be computationally obnoxious to reason about rewriting of $\langle\langle \Upsilon \rangle\rangle_{\{\bullet,\circ\}}$-equivalence classes of graphs. We now proceed in steps towards a solution to the problem. First, henceforth we shall treat the two equations in Υ as rewriting rules on $\mathbb{D}$, with a left-to-right orientation. For simplicity we denote by Υ also the resulting rewriting system. We then observe the result below.

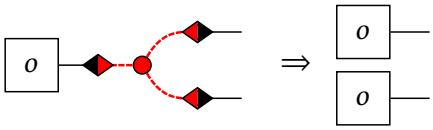
Lemma 6.6. Υ *is terminating and confluent on* $\mathbb{D}$.

Our next goal is then to show that rewriting modulo Υ can be simulated without loss of generality by putting a graph in Υ-normal form and then applying the rewriting rule. However, a naive application of this approach immediately poses problems, as it is shown by the following example.

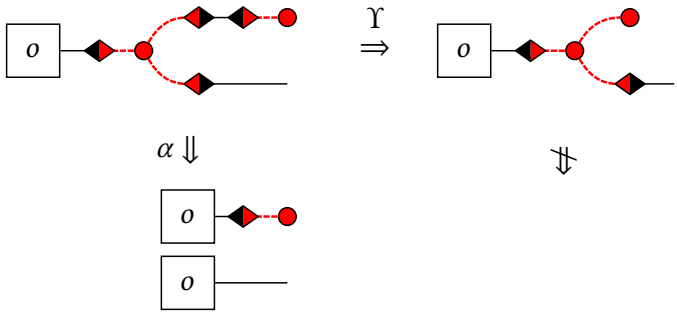
Example 6.7. Suppose Σ contains an operation $o : 0 \to 1$ and consider a rewriting rule α defined as



Under the interpretation $(\cdot)_\bullet$ it yields a rule $\alpha_\bullet$ in $\mathbb{D}$ defined as



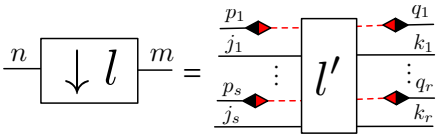
The translated rule conflicts with Υ, meaning that Υ can erase $\alpha_\bullet$-redexes. For instance



(44)

This kind of problematic example motivates the following transformation on the rewriting rules of $\mathbb{D}$. As a preparatory step, we record the following lemma, where $\downarrow c$ is the unique Υ-normal form of a morphism c of $\mathbb{D}$, guaranteed by Lemma 6.6.

Lemma 6.8. *Let l be a morphism of $\mathbb{D}$ in the image of $(\cdot)_\bullet$. Then, there is a morphism l' not containing any Υ-redex such that (in $\mathbb{D}$)*



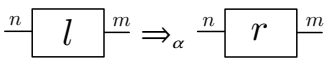
(45)

where $p_1 + j_1 + \cdots + p_s + j_s = n$ and $q_1 + k_1 + \cdots + q_r + k_r = m$, with $p_i, j_i, q_i, k_i \in \mathbb{N}$. Moreover, there is a unique such l' in $\mathbb{D}$ for each l.

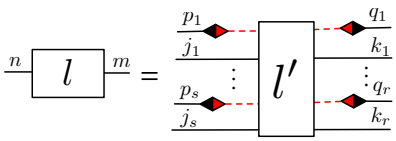
Proof. Given l, by Lemma 6.6 there is a unique $\downarrow l$ in Υ-normal form. Because l is in the image of $(\cdot)_\bullet$, besides —◆··· and ···◆— it can only contain $\bullet$-sorted operators, and external dangling wires are also of sort $\bullet$: thus every wire of sort $\circ$ inside $\downarrow l$ can only be connected to the left boundary via a —◆··· and to the right boundary via a ···◆— . Using the laws of SMCs to "pull out" all such connecting operators towards the corresponding boundary, we obtain the LHS of (45). □

We are now ready to introduce the transformation that will remove the conflicts between a rewriting rule and Υ.

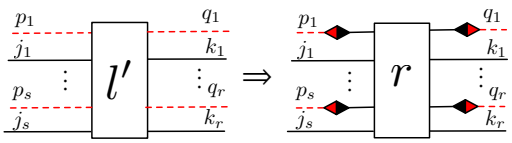
Definition 6.9. Let α be a rewriting rule of type $\bullet^n \to \bullet^m$ on $\mathbb{D}$



We obtain l' through Lemma 6.8

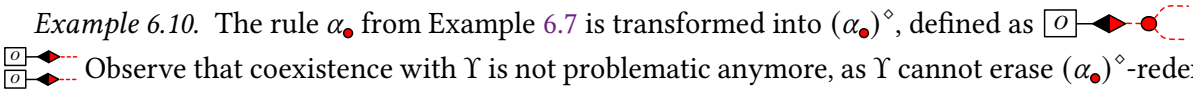


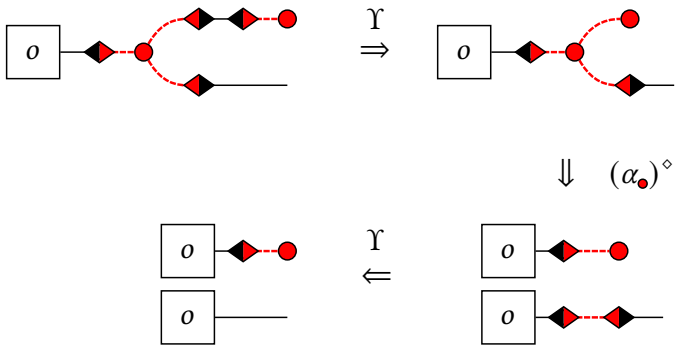
The rule $\alpha^\diamond$ is defined as



Given a rewriting system $\mathcal{R}$, we write $\mathcal{R}^\diamond = \{\alpha^\diamond \mid \alpha \in \mathcal{R}\}$.

It is instructive to show how this transformation neutralises the problem of (44).

Example 6.10. The rule $\alpha_{\bullet}$ from Example 6.7 is transformed into $(\alpha_{\bullet})^{\diamond}$, defined as  Observe that coexistence with Υ is not problematic anymore, as Υ cannot erase $(\alpha_{\bullet})^{\diamond}$-redexes. For instance, in computation (44) we are not stuck anymore



We claim that this "improved" rewriting system removes conflicts with Υ. Or, to put it another way: a single syntactic rewriting step is possible if and only if it can be obtained as a combinatoric step on an Υ-normal form, followed by renormalising using Υ. Symbolically, we can write this as follows

$$a \Rightarrow_{\mathcal{R}} b \quad \text{iff} \quad \downarrow\langle\langle a_{\bullet}\rangle\rangle_{\{\bullet,\bullet\}} \Rrightarrow_{\langle\langle \mathcal{R}^{\diamond}_{\bullet}\rangle\rangle_{\{\bullet,\bullet\}}} \overset{\star}{\Rrightarrow}_{\langle\langle \Upsilon\rangle\rangle_{\{\bullet,\bullet\}}} \downarrow\langle\langle b_{\bullet}\rangle\rangle_{\{\bullet,\bullet\}}$$

Proving the formal statement above will be the main task for the remainder of this section, and it appears as Theorem 6.16.

Contrary to the situation depicted at the beginning of the section, in Theorem 6.16 DPOI rewriting in the combinatorial domain is "on-the-nose": instead of dealing with $\langle\langle\Upsilon\rangle\rangle_{\{\bullet,\bullet\}}$-equivalence classes of hypergraphs, we can now deal exclusively with $\langle\langle\Upsilon\rangle\rangle_{\{\bullet,\bullet\}}$-normal forms, which thanks to Lemma 6.6 are straightforward to compute.

The proof of Theorem 6.16 will go in steps. The theory so far ensures a correspondence between

- rewriting in $\mathbb{C}$ and rewriting in $\mathbb{D}_{\Upsilon}$, thanks to Proposition 6.3;
- rewriting in $\mathbb{D}$ and rewriting in $\mathrm{Csp}_{F_{\{\bullet,\bullet\}} I_{\{\bullet,\bullet\}}}(\mathbf{Hyp}_{\{\bullet,\bullet\},\Sigma\uplus\Gamma})$, thanks to Proposition 4.10.

Thus the only missing link to complete the correspondence in Theorem 6.16 is to adequately represent rewriting in $\mathbb{D}_{\Upsilon}$ as rewriting in $\mathbb{D}$. This is the remit of the next section.

6.3 Adequacy of the Implementation

For the purposes of this section, let $\mathcal{R}$ be a rewriting system on $\mathbb{D}$. We focus on the missing piece of the proof of Theorem 6.16: showing that the rule transformation of Definition 6.9 is an adequate implementation for $\mathcal{R}$-rewriting modulo Υ in $\mathbb{D}$.

Theorem 6.11 (Adequacy). *Let c and d be arrows in $\mathbb{D}$. Then*

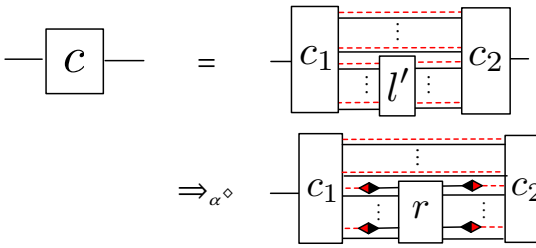
$$c \overset{\star}{\Leftrightarrow}_{\Upsilon} \Rightarrow_{\mathcal{R}} \overset{\star}{\Leftrightarrow}_{\Upsilon} d \;\; \textit{iff} \;\; \downarrow c \;\Rightarrow_{\mathcal{R}^{\diamond}} \overset{\star}{\Rightarrow}_{\Upsilon} \downarrow d.$$

The proof will follow from Propositions 6.12 and 6.15. For the right-to-left direction (completeness), we can prove a stronger statement.

Proposition 6.12. *$c \Rightarrow_{\mathcal{R}^{\diamond}} d$ implies $c \overset{\star}{\Leftrightarrow}_{\Upsilon} \Rightarrow_{\mathcal{R}} d$.*

Proof. For the sake of readability, all the diagrams in the proofs of this section are depicted with unlabelled wires: it is intended that each wire stands for a number of parallel wires of the same type, arbitrary but compatible with its position in the diagram.

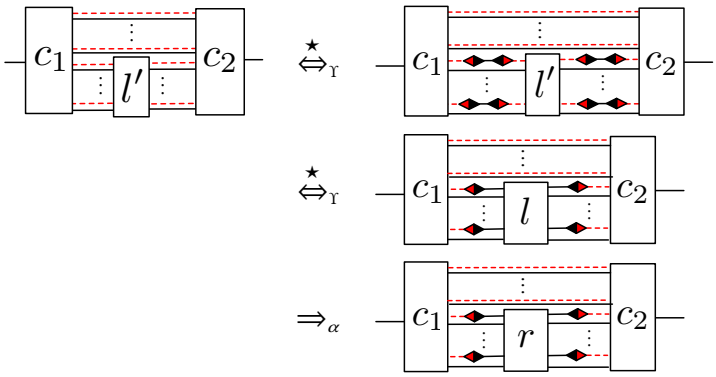
We know c has a redex for $\alpha^\diamond$ for some rule $\alpha \in \mathcal{R}$. If α is given by $l \Rightarrow r$, then $\alpha^\diamond$ rewrites c as



(46)

(47)

In light of (46), c modulo Υ contains a redex for α as well

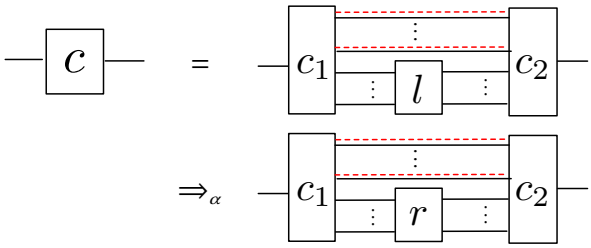


where the second step is justified by the definition of l' as in (45). This proves the statement. □

The left-to-right direction (soundness) of Thorem 6.11 requires more work. First, we have that $\mathcal{R}^\diamond$ is as powerful as $\mathcal{R}$, modulo Υ.

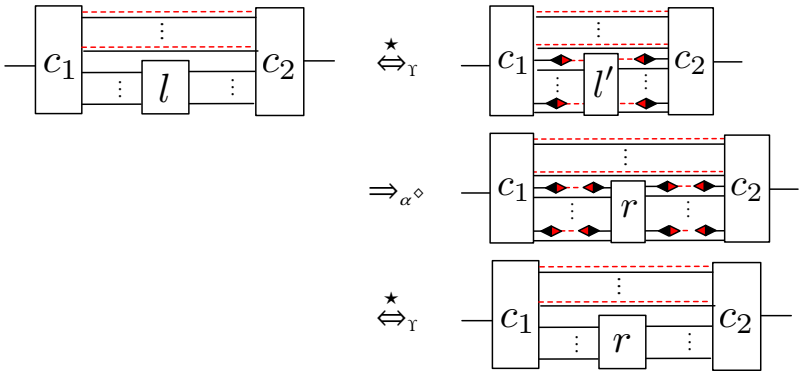
Lemma 6.13. *$c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_\mathcal{R} \overset{\star}{\Leftrightarrow}_\Upsilon d$ iff $c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$.*

Proof. It suffices to show that $c \Rightarrow_\mathcal{R} d$ implies $c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$ and $c \Rightarrow_{\mathcal{R}^\diamond} d$ implies $c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_\mathcal{R} \overset{\star}{\Leftrightarrow}_\Upsilon d$. For the left-to-right implication, the assumption is that c contains a redex for a rule $\alpha \in \mathcal{R}$, say of the form $l \Rightarrow r$



(48)

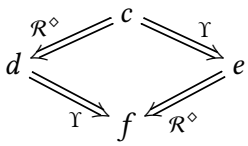
Then, modulo Υ, c also contains a redex for $(l \Rightarrow r)^\diamond$. Applying this rule yields the same outcome, modulo-Υ, as (48)



This proves the left-to-right implication of the statement. The right-to-left implication is given by Proposition 6.12. □

The next step is to show that $\Rightarrow_{\mathcal{R}^\diamond}$ satisfies a "diamond property" with respect to Υ. This property implies that Υ-rewriting does not interfere with $\mathcal{R}^\diamond$-rewriting— whence the latter can be assumed without loss of generality to work on arrows in Υ-normal form, as in the desired implementation (Theorem 6.11). As shown in Example 6.7, the diamond property fails for arbitrary rewriting systems and justifies the introduction of the transformation $(\cdot)^\diamond$.

Lemma 6.14 (Diamond Property). *If $c \Rightarrow_{\mathcal{R}^\diamond} d$ and $c \Rightarrow_\Upsilon e$ then there exists an f such that $d \Rightarrow_\Upsilon f$ and $e \Rightarrow_{\mathcal{R}^\diamond} f$*



Proof. This is immediate from the fact that, by Definition 6.9, the application of $\mathcal{R}^\diamond$ cannot introduce Υ-redexes. Therefore, $\mathcal{R}^\diamond$ and Υ are orthogonal rewriting systems (i.e. they have no critical pairs between each other). □

We are now ready to show soundness.

Proposition 6.15. *$c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_\mathcal{R} \overset{\star}{\Leftrightarrow}_\Upsilon d$ implies $\downarrow c \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Rightarrow}_\Upsilon \downarrow d$.*

Proof. Since Υ is confluent and terminating (Lemma 6.6), the conclusion is equivalent to $\downarrow c \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$, so we focus on this statement.

Assume $c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_\mathcal{R} \overset{\star}{\Leftrightarrow}_\Upsilon d$. By Lemma 6.13, this implies $c \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$. Since Υ is confluent and terminating, this implies $c \overset{\star}{\Rightarrow}_\Upsilon \downarrow c \; {}_\Upsilon\overset{\star}{\Leftarrow} \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$.

We can now drop the first part of the rewrite sequence and focus on $\downarrow c \; {}_\Upsilon\overset{\star}{\Leftarrow} \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$. By repeatedly applying Lemma 6.14, we can commute $\Rightarrow_{\mathcal{R}^\diamond}$ through ${}_\Upsilon\overset{\star}{\Leftarrow}$ to obtain $\downarrow c \Rightarrow_{\mathcal{R}^\diamond} {}_\Upsilon\overset{\star}{\Leftarrow} \overset{\star}{\Leftrightarrow}_\Upsilon d$. Finally, merging ${}_\Upsilon\overset{\star}{\Leftarrow}$ and $\overset{\star}{\Leftrightarrow}_\Upsilon$ yields $\downarrow c \Rightarrow_{\mathcal{R}^\diamond} \overset{\star}{\Leftrightarrow}_\Upsilon d$ as required. □

We now have all the ingredients to prove the main theorem of this section: the DPOI rewriting implementation of rewriting in $\mathbb{C}$.

Theorem 6.16. *Let $\mathcal{R}$ be a rewriting system on $\mathbb{C}$. Then*

$$a \Rightarrow_\mathcal{R} b \quad \text{iff} \quad \downarrow \langle\!\langle a_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}} \Rrightarrow_{\langle\!\langle \mathcal{R}^\diamond_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}}} \overset{\star}{\Rrightarrow}_{\langle\!\langle \Upsilon \rangle\!\rangle_{\{\bullet,\bullet\}}} \downarrow \langle\!\langle b_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}}$$

Proof. First, we have a correspondence at the level of syntactic rewriting (Definition 2.17) in the props $\mathbb{C}$ and $\mathbb{D}_\Upsilon$

$$\boxed{a \Rightarrow_\mathcal{R} b \text{ in } \mathbb{C}} \quad \text{iff} \quad \boxed{a_\bullet \Rightarrow_{\mathcal{R}_\bullet} b_\bullet \text{ in } \mathbb{D}_\Upsilon} \tag{49}$$

This is ensured by the fact that $(\cdot)_\bullet$ is a functorial and full and faithful mapping. Second, we interpret Υ as a set of rewriting rules instead of a set of equations. Then, rewriting in $\mathbb{D}_\Upsilon$ is just the same as rewriting in $\mathbb{D}$ modulo Υ-rewriting. Starting from the right-hand side of (49)

$$\boxed{a_\bullet \Rightarrow_{\mathcal{R}_\bullet} b_\bullet \text{ in } \mathbb{D}_\Upsilon} \quad \text{iff} \quad \boxed{a_\bullet \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_{\mathcal{R}_\bullet} \overset{\star}{\Leftrightarrow}_\Upsilon b_\bullet \text{ in } \mathbb{D}} \tag{50}$$

where $a_\bullet$ and $b_\bullet$ are understood on the right as arrows of $\mathbb{D}$, *cf.* Remark 6.2. Third, we use Theorem 6.11 to give an implementation for rewriting modulo-Υ. Starting from the right-hand side of (50)

$$\boxed{a_\bullet \overset{\star}{\Leftrightarrow}_\Upsilon \Rightarrow_{\mathcal{R}_\bullet} \overset{\star}{\Leftrightarrow}_\Upsilon b_\bullet \quad \text{in } \mathbb{D}} \quad \text{iff} \quad \boxed{\downarrow a_\bullet \Rightarrow_{\mathcal{R}_\bullet^\diamond} \overset{\star}{\Rightarrow}_\Upsilon \downarrow b_\bullet \quad \text{in } \mathbb{D}} \tag{51}$$

Last, Corollary 6.5 and Proposition 4.10 yield the correspondence between rewriting in $\mathbb{D}$ and DPO-rewriting in $\mathrm{Csp}_{F_{\{\bullet,\bullet\}} I_{\{\bullet,\bullet\}}}(\mathbf{Hyp}_{\{\bullet,\bullet\},\Sigma \uplus \Gamma})$. Starting from the right-hand side of (51)

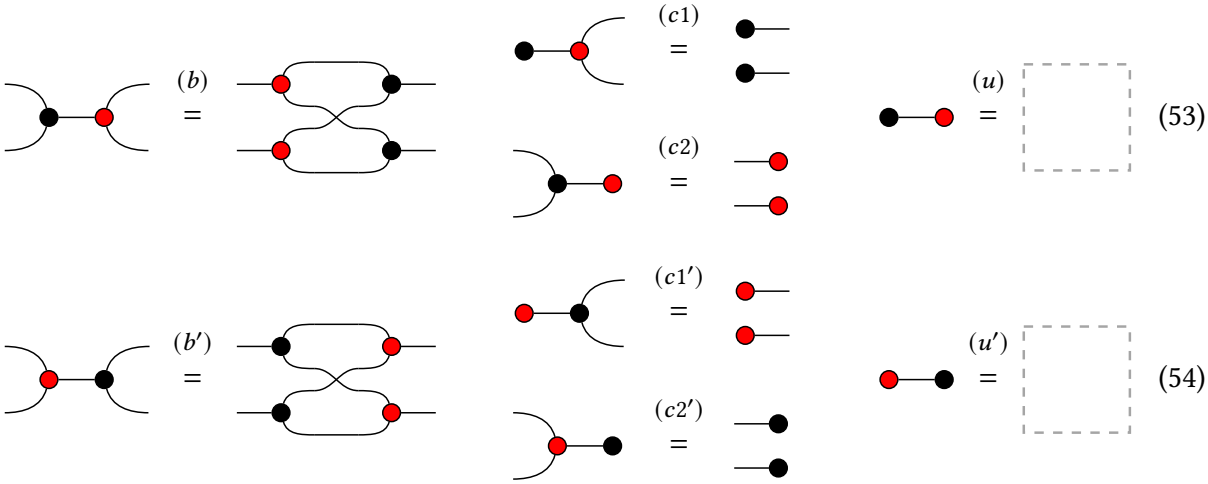
$$\boxed{\downarrow a_\bullet \Rightarrow_{\mathcal{R}_\bullet^\diamond} \overset{\star}{\Rightarrow}_\Upsilon \downarrow b_\bullet \quad \text{in } \mathbb{D}} \quad \text{iff} \quad \boxed{\langle\!\langle \downarrow a_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}} \Rrightarrow_{\langle\!\langle \mathcal{R}_\bullet^\diamond \rangle\!\rangle_{\{\bullet,\bullet\}}} \overset{\star}{\Rrightarrow}_{\langle\!\langle \Upsilon \rangle\!\rangle_{\{\bullet,\bullet\}}} \langle\!\langle \downarrow b_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}} \quad \text{in } \mathrm{Csp}_{F_{\{\bullet,\bullet\}} I_{\{\bullet,\bullet\}}}(\mathbf{Hyp}_{\{\bullet,\bullet\},\Sigma \uplus \Gamma})} \tag{52}$$

Note that $\langle\!\langle \downarrow a_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}} = \downarrow \langle\!\langle a_\bullet \rangle\!\rangle_{\{\bullet,\bullet\}}$, where the normal form on the right is computed in the category $\mathrm{Csp}_{D_{\{\bullet,\bullet\}}}(\mathbf{Hyp}_{\{\bullet,\bullet\},\Sigma \uplus \Gamma})$ according to the rules $\langle\!\langle \Upsilon \rangle\!\rangle_{\{\bullet,\bullet\}}$. To conclude, by chaining (49) to (52) we obtain the statement of the theorem. □

This theorem gives us an effective combinatorial method for rewriting modulo multiple Frobenius algebras.

7 EXAMPLE: INTERACTING BIALGEBRAS

We now turn to one of the main examples of multiple interacting Frobenius algebras: the case of two Frobenius algebras that together interact as a bialgebra. In some sense, this specialises the example from Section 5 that had a single Frobenius algebra $\bullet$ interacting with a group ($m : 2 \to 1, u : 0 \to 1, i : 1 \to 1$). Here, we assume that m and u are themselves part of a second Frobenius algebra. In this case, the associativity and unit equations come for free, so it remains to state analogous equations to(39), which make the two Frobenius structures "natural" with respect to each other



(53)

(54)

For simplicity, we also require that the induced 'cup' and 'cap' maps coincide

(55)

which will entail the "inverse" equation from (38) for $i = id_1$ (see rule (h) in the derived rules (56) below). Hence, it is a strict specialisation of the group algebra structure introduced in Section 5.

This system, referred to as $\mathbb{IB}$ [BSZ14a] ('Interacting Bialgebras'), has appeared ubiquitously in the study of component-based systems across different research areas. It forms the core of the *ZX-calculus* [CD08], which has recently been extended to give sound and complete equational

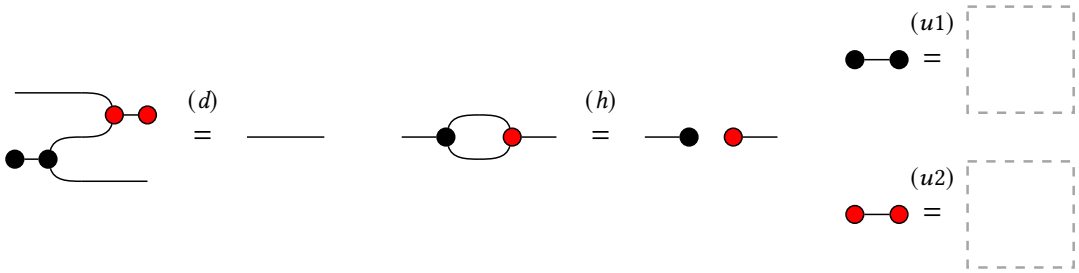
theories for approximately [JPV18] and fully [NW17] universal families of quantum circuits. Also, it has been employed to reason about signal processing circuits in control theory [BSZ14b, BE15], electrical circuits [BPSZ19], and Petri nets [BHP+19].

Remark 7.1. Recall that the models in $(\mathbf{Vect}_k, \otimes)$ of the structure from Section 5 correspond exactly to representations of finite groups (i.e. group algebras). The models of a pair of Frobenius algebras ●, ● satisfying equations (53) and (54) specialise this fact: they correspond exactly to the representations of finite *Abelian* groups. If we additionally impose (55), these are representations of finite Abelian groups whose elements are all self-inverse. See [CK17], Section 9.6.1 for details.

7.1 A Representation Theorem

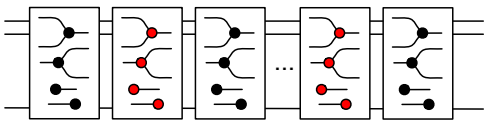
This section will propose a rewriting strategy that exploits the DPO implementation presented in the previous sections and allows for turning any diagram of such an "interacting bialgebra" into a suitable normal form.

The first step is to note that from the rules above one can derive (see e.g. [CD08]) the following two rules, which will soon be useful

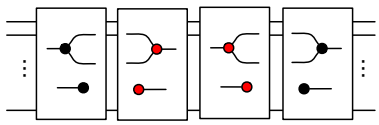


(56)

A generic diagram composed of generators from these two Frobenius algebras consists of arbitrarily many alternating layers of ● and ● generators



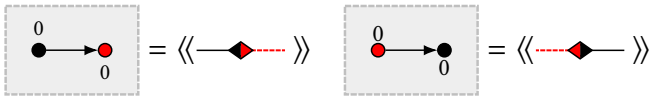
Any such diagram can be rewritten into a ●-*reduced form* that consists of just four layers: an initial layer of ●-comonoid structure, followed by ●-monoid structure, followed by ●-comonoid structure, followed by a final layer of ●-monoid structure



(57)

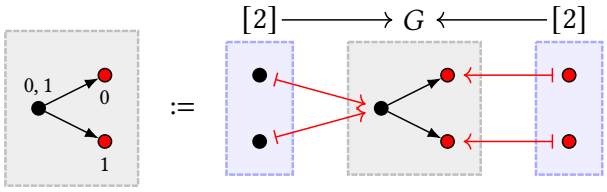
We call this the ●-*reduced form* because there is no internal layer of ● generators. We now characterise these forms in terms of their associated hypergraphs with interfaces. To express hypergraphs with interfaces compactly and unambiguously, we adopt the following notational conventions

(1) As we did in Example 4.6, hyperedges corresponding to and are depicted as unlabelled, directed edges between nodes of appropriate colour, hence



(2) To avoid writing interfaces explicitly, we will indicate these by consistently placing labels above nodes for inputs and below nodes for outputs. For example, the following hypergraph

with interfaces is abbreviated as



Using these conventions, the rules in the system Υ can be written as the following two DPOI rules

Hence, normalising with respect to Υ contracts away any node with precisely one in-edge and one out-edge. We are now ready to characterise $\bullet$-reduced forms.

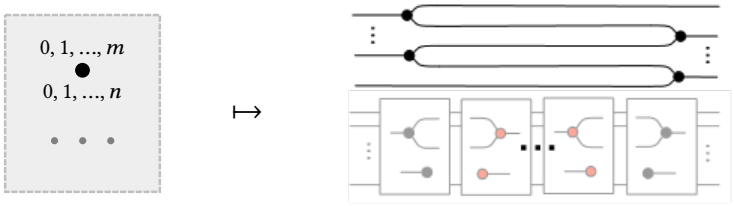
Proposition 7.2. *A string diagram generated by two Frobenius algebras $\bullet$ and $\color{red}\bullet$ is in $\bullet$-reduced form as in (57) (modulo Frobenius equations) if and only if its associated hypergraph with interfaces $I_1 \to G \leftarrow I_2$ satisfies the following conditions. G is directed acyclic and every $\bullet$ node in G is either*

(I) in the image of a single node in I_1 and has no in-edges,
(O) in the image of a single node in I_2 and has no out-edges, or
*(IO) in the image of one or more nodes in **both** I_1 and I_2.*

Proof. When hypergraph nodes representing Frobenius algebra generators are composed, they fuse together. Hence, the nodes in G correspond to maximal connected components of Frobenius algebra generators of the same colour.

First, suppose a string diagram is in the form of (57). Then, each $\bullet$ node in the hypergraph G corresponds to a maximal connected component of $\bullet$ Frobenius generators in (57). We first note that any (co)unit connected to a (co)multiplication can be reduced away. Hence, we need to consider only 5 cases for connected components of $\bullet$ generators: (1) a counit applied to an input wire, (2) a unit applied to an output wire, (3) a tree of comultiplications applied to an input wire, (4) a tree of multiplications connected to an output wire, or (5) a connected component of cases (3) and (4). Cases (1) and (3) yield a $\bullet$ node of type (I). Cases (2) and (4) yield a node of type (O), and case (5) yields a node of type (IO).

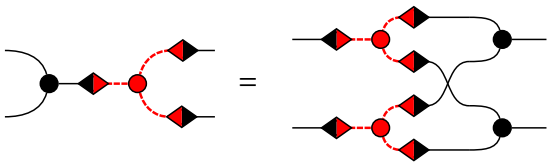
Conversely, we can interpret each of the $\bullet$ nodes of types (I) and (O) as cases (1)-(4) described above. The only difficult case is nodes of type (IO). These can be interpreted as a 'zig-zag' of $\bullet$ comultiplications in the first layer of (57) and multiplications in the last layer, with no $\color{red}\bullet$ generators in between



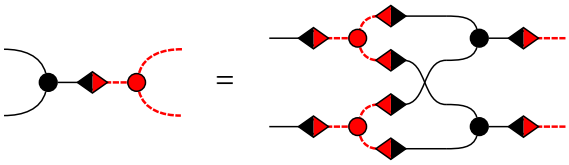
□

Crucially, a hypergraph with interfaces that satisfies the conditions above contains no *interior* $\bullet$ *nodes*, i.e. nodes not in the image of I_1 or I_2. Eliminating these nodes will form the main component of the strategy below. In order to obtain a hypergraph with interfaces satisfying these conditions, we first perform the transformation of the interacting bialgebra rules into a DPOI rewriting system. This is a mechanical procedure, but for clarity, we will show it explicitly for the rule (b). Following the recipe of Theorem 6.16, we first use $(\cdot)_{\color{red}\bullet}$ to get the polychromatic interpretation —(58) below—

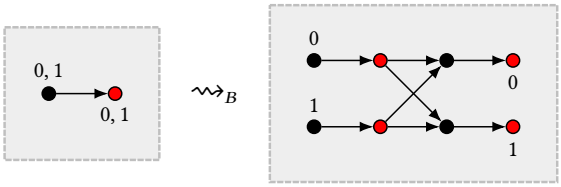
then apply $(\cdot)^\diamond$ to shift the colour change maps on inputs/outputs to the right-hand side —(59)— and finally apply $\langle\!\langle \cdot \rangle\!\rangle_{\{\bullet,\circ\}}$ to interpret



(58)

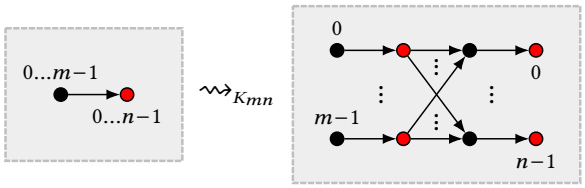


(59)



(60)

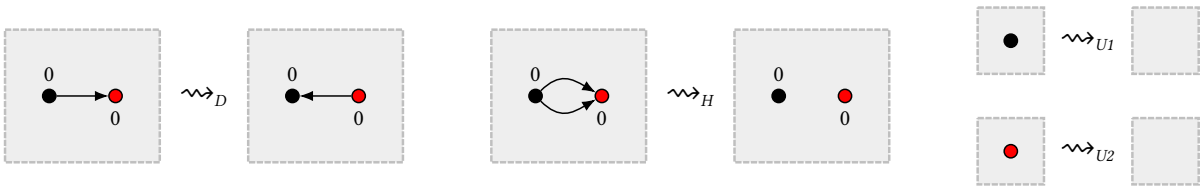
We can give a similar treatment to $(cp1)$, $(cp2)$, and (u) rules. Equivalently, we can introduce a family of rules $K_{m,n}$ for $m, n \geq 0$



(61)

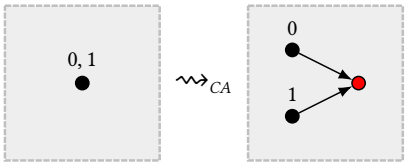
where the righthand-side contains the fully connected bipartite graph from m red nodes to n black nodes. Note when $m = 0$ (resp. $n = 0$), we interpret the range $0...m-1$ (resp. $0...n-1$) as empty. This family of rules is implied by the Frobenius equations and (b), $(cp1)$, $(cp2)$, and (u). Conversely, it implies these 4 rules as special cases (see e.g. [CK17]).

We have mentioned the derived equations in box (56) because, once we translate them into DPOI rules, we see that rule (d) allows to reverse the direction of an arbitrary edge, rule (h) to delete parallel edges, and rules $(u1)$ and $(u2)$ to delete single, isolated nodes



Note the rule (D) (and its converse) allow us to essentially work with undirected graphs, as we can always reverse an edge directions if necessary to create a match. This renders the additional "primed" equations in box (54) redundant, since they are the same as the rules above, but with the directions reversed.

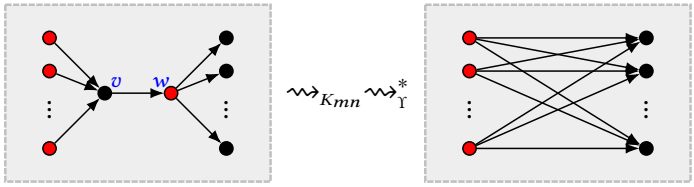
Hence, the only other rule we need is a rule for introducing red caps



Reduction Strategy *We begin with a hypergraph with interfaces* $I_1 \to G \leftarrow I_2$*, whose interfaces* I_1, I_2 *are all of the* ●*-sort. It should be understood that after every rewriting step, the graph is normalised with respect to rules* $(\Upsilon 1)$ *and* $(\Upsilon 2)$*. The strategy proceeds as follows*

(1) Reduce as much as possible using rules $(U1)$, $(U2)$, and (H), using the rule (D) to reverse edge directions as necessary.

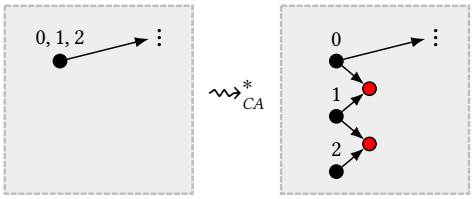
(2) If there are no interior ● nodes, go to step 5. Otherwise, apply the rule K_{mn} to an interior ● node v and one neighbouring ● w node to remove it as follows:



where we again use the (D) rule to reverse edge directions as necessary.

(3) If there are remaining interior ● nodes, go to step 1.

(4) If a ● node is in the image of multiple nodes in I_1 and of no nodes in I_2, apply the converse of rule (CA) to split it into multiple ● nodes connected by ● nodes. For example



We split nodes only in the image of I_2 similarly.

(5) Apply (D) or its converse to direct the remaining edges from the image of I_1, to the ● nodes, then to the image of I_2.

The "essential trick" in this strategy is step (2), which removes pairs of adjacent nodes (v, w) at the expense of introducing some additional edges. Since we are always removing nodes and parallel edges in the "main loop" (i.e. steps 1-3), the number of nodes in the graph always goes down, and the number of edges is bounded above by the number of nodes. Steps (4) and (5) are then just a finite amount of post-processing in order to get the exact form in Proposition 7.2.

THEOREM 7.3. *The* ***Reduction Strategy*** *above terminates and yields a graph in reduced form.*

PROOF. Each iteration of steps 1-3 reduces the number of interior ● nodes by 1. Hence it terminates after n iterations for n interior ● nodes, with no interior ● nodes. Step 4 guarantees all remaining, non-interior ● nodes are of the form (I), (O), or (IO) as in Proposition 7.2, and step 5 guarantees the directed acyclicity conditions. □

Remark 7.4. The quantum circuit optimisation tool PyZX [KvdW20] uses a version of the **Reduction Strategy** above to simplify phase-free diagrams using the ZX-calculus.

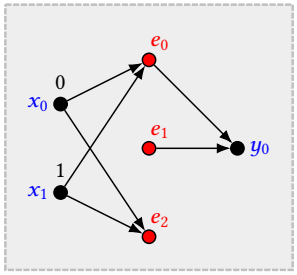
7.2 Rewriting as quantifier elimination

We close this section with a brief discussion about the ●-reduced form, and its relationship to the semantics of $\mathbb{IB}$. It was shown in [BSZ14a] that the PROP for $\mathbb{IB}$ is isomorphic to the PROP

LinRel($\mathbb{Z}_2$) of $\mathbb{Z}_2$-linear relations. That is, morphisms $S : m \to n$ are linear sub-spaces $S \subseteq Z_2^m \times Z_2^n \cong Z_2^{m+n}$, $\oplus$ is given by direct product, and composition is done relation-style

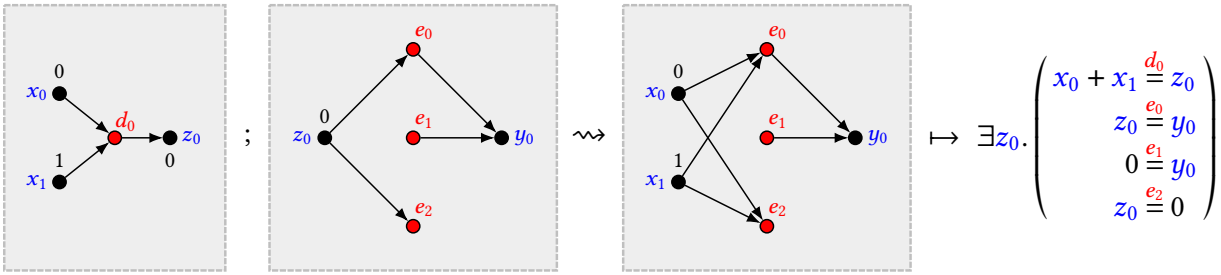
$$(v, w) \in (S; T) \iff \exists u.(v, u) \in S, (u, w) \in T \tag{62}$$

As explained in [Zan15], the •-reduced form (called the *cospan form* therein) enables us to 'read off' S as a homogeneous system of equations (or equivalently, as a basis for $S^\perp$). In this form, • nodes correspond to variables, and • nodes to equations, whose LHS and RHS consist of those variables connected by in-edges and out-edges, respectively. For example, the diagram below represents the space of solutions to the following system of equations

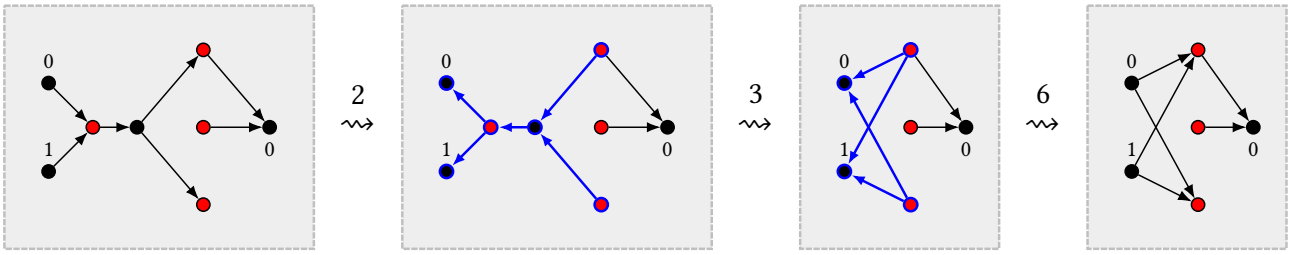


$$\mapsto \quad \begin{pmatrix} x_0 + x_1 \overset{e_0}{=} y_0 \\ 0 \overset{e_1}{=} y_0 \\ x_0 + x_1 \overset{e_2}{=} 0 \end{pmatrix} \tag{63}$$

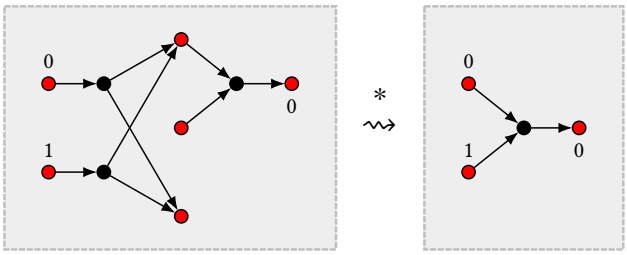
This interpretation gives a semantical view of the **Reduction Strategy** as a quantifier elimination procedure. The main purpose of the procedure is to eliminate interior • nodes. Since these nodes arise from sequential compositions in **LinRel**($\mathbb{Z}_2$), equation (62) tells that they correspond to existentially quantified variables



The core of the **Reduction Strategy** are steps 2 and 3. The former isolates an existentially quantified variable z on the LHS of an equation e, and step 3 substitutes any occurrence for that variable with its RHS, simultaneously eliminating z and e. Applying this procedure to the diagram above yields the •-reduced form in (63)



Since everything in $\mathbb{IB}$ and the **Reduction Strategy** is colour-symmetric, we can use the same strategy to compute the analogous • reduced forms. To do so, we first pre- and post-compose with colour changers to obtain a graph with an interface consisting entirely of • nodes, then apply the **Reduction Strategy** with the colours reversed. Applying this to example (63) yields



This again gives a canonical representation of a sub-space S (called the *span form*), but this time as a basis for S itself, rather than $S^\perp$ [Zan15]. ● nodes correspond to basis vectors, where the presence of an edge indicates a 1 in the corresponding position. The final diagram in the rewrite sequence above represents S as $\mathbf{span}\{((1,1),(0))\} \subseteq \mathbb{Z}_2^2 \times \mathbb{Z}_2$, which is indeed the space of solutions to the system given in (63).

Note that we have focused on the case of $\mathbb{IB}$ and $\mathbb{Z}_2$-linear equations because it is the simplest. It was shown in [BSZ17b] that this system generalises straightforwardly to a system $\mathbb{IB}_F$ that has as its PROP $\mathbf{LinRel}(F)$ for an arbitrary field F. In that case, we introduce a family of generators for each $a \in F\backslash\{0,1\}$ that give weights to edges. By modifying the **Reduction Strategy** to account for these weights, we can still obtain a (slightly more elaborate) procedure for removing internal nodes. This then gives the graphical analogue to quantifier elimination over an arbitrary field F.

Interestingly, this graphical version of quantifier elimination is *inherently compositional*. It is possible to introduce generators and relations, breaking the semantic connection with $\mathbf{LinRel}(F)$, while using **Reduction Strategy** on sub-diagrams in the $\mathbb{IB}_F$ fragment. This technique can exploit the fact that the ZX-calculus contains $\mathbb{IB}$ to perform peephole optimisations on quantum circuits, even though the latter have a more complex semantics than $\mathbf{LinRel}(\mathbb{Z}_2)$.

For yet another perspective, recall that $\mathbb{IB}$ enjoys a modular characterisation in terms of distributive laws of PROPs [BSZ14a], which prescribes that each diagram can be turned into *cospan form* and *span form*. As observed, these correspond to ●-reduced and ●-reduced forms respectively: thus our result provides algorithmic means to reach them, which were lacking in the abstract picture. It also fills the main gap in formulating the realisability procedure for signal flow graphs [BSZ15, BSZ17a] entirely as a diagram rewriting procedure.

8 CONCLUSIONS

Increasingly, string diagrams are establishing themselves as the yardstick formalism to reason compositionally about graphical models of computation across different fields. These developments demand a mathematical foundation of how to *compute* with equational theories of string diagrams, seen as rewriting systems. In this work, we laid out such foundations, in the form of a sound and complete interpretation of string diagram rewriting as double-pushout rewriting on a suitable domain of hypergraphs. One fundamental aspect of this modelling is the presence of *interfaces*: the compositional nature that is intrinsic to string diagrams has to be adequately mirrored in the hypergraph interpretation, which we achieved by resorting to the theory of cospans, graphs with interfaces and the associated notion of rewriting.

From the viewpoint of string diagrams, the key advantage of working under the hypergraph interpretation is that the structural laws usually imposed on the diagrammatic syntax get absorbed in the combinatorial representation. In this way, instead of having to deal with the subtleties of rewriting modulo those structural laws, one may use double-pushout rewriting "on-the-nose" on the corresponding hypergraphs.

Furthermore, we saw how absorbing all the structural laws becomes way subtler once there are more than one Frobenius structure at stake. In that situation, we were able to come up with an

adequate interpretation by imposing extra structure to distinguish the different Frobenius algebras, which was suitably normalised during the rewriting procedure.

We concluded the paper with an extended example, in which we proposed a terminating rewriting strategy for the theory of Interacting Bialgebras. This is a well-studied diagrammatic calculus with cross-disciplinary applications. The richness of the calculus, featuring two bialgebras and two Frobenius algebras, had previously made it very difficult and unwieldy to study its rewriting properties. Using our interpretation, the two Frobenius algebra structures get absorbed, leaving only the bialgebra rules as non-trivial rewrites, which drastically simplifies its study. This led us to a representation theorem, a reduction strategy, and a semantical view of such strategy as quantifier elimination.

The work presented in this paper is only the first part of the story: we have presented a characterisation of string diagram rewriting modulo Frobenius algebras, and showed that it matches naturally double-pushout rewriting of hypergraphs with interfaces. But what about equational theories that do not feature any Frobenius algebra? Building on the framework developed so far, in the sequel of this paper we show how also these theories, in which rewriting just happens modulo the laws of symmetric monoidal categories, without any additional structure, can be characterised in terms of double-pushout rewriting. Next, we complete our framework by investigating confluence in the context of string diagram rewriting.

Acknowledgements. We are thankful to the anonymous referees for their helpful comments; Fabio Gadducci acknowledges support from Italian MIUR PRIN 2017FTXR7S IT-MATTERS (Methods and Tools for Trustworthy Smart Systems); Fabio Zanasi acknowledges support from EPSRC grant EP/V002376/1. Paweł Sobociński was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001) and the Estonian Research Council grant PRG1210.

REFERENCES

[AC04] Samson Abramsky and Bob Coecke. A categorical semantics of quantum protocols. In *LICS 2004*, pages 415–425. IEEE Computer Society, 2004.

[ASW09] Luisa Albasini, Nicoletta Sabadini, and Robert F. C. Walters. Cospans and spans of graphs: A categorical algebra for the sequential and parallel composition of discrete systems. *Preprint arXiv:0909.4136*, 2009.

[Bé67] Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, volume 47 of *LNM*, pages 1–77. Springer, 1967.

[BCR17] John Baez, Brandon Coya, and Franciscus Rebro. PROPs in network theory. *Preprint arXiv:1707.08321*, 2017.

[BD89] Leo Bachmair and Nachum Dershowitz. Completion for rewriting modulo a congruence. *Theoretical Computer Science*, 67(2-3):173–201, 1989.

[BE15] John Baez and Jason Erbele. Categories in control. *Theory and Application of Categories*, 30:836–881, 2015.

[BG01] Roberto Bruni and Fabio Gadducci. Some algebraic laws for spans (and their connections with multirelations). In Wolfram Kahl, David Lorge Parnas, and Gunther Schmidt, editors, *RELMIS 2001*, volume 44(3) of *ENTCS*, pages 175–193. Elsevier, 2001.

[BGK09] Filippo Bonchi, Fabio Gadducci, and Barbara König. Synthesising CCS bisimulation using graph rewriting. *Information and Computation*, 207(1):14–40, 2009.

[BGK+16] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *LICS 2016*, pages 710–719. ACM, 2016.

[BGK+17] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Pawel Sobociński, and Fabio Zanasi. Confluence of graph rewriting with interfaces. In Hongseok Yang, editor, *ESOP 2017*, volume 10201 of *LNCS*, pages 141–169. Springer, 2017.

[BGK+18] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. Rewriting with Frobenius. In Anuj Dawar and Erich Grädel, editors, *LICS 2018*, pages 165–174. ACM, 2018.

[BGK+20] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. String diagram rewrite theory II: Rewriting with symmetric monoidal structure. *Preprint available at arXiv:2104.14686*, 2020.

[BGK+21] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobociński, and Fabio Zanasi. String diagram rewrite theory III: Confluence with and without Frobenius. *Preprint available at arXiv:2109.06049*, 2021.

[BHP+19] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. In *POPL 209*, volume 3 of *PAPL*, pages 25:1–25:28. ACM, 2019.

[BHPS17] Filippo Bonchi, Joshua Holland, Dusko Pavlovic, and Pawel Sobociński. Refinement for signal flow graphs. In Roland Meyer and Uwe Nestmann, editors, *CONCUR 2017*, volume 85 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[BPSZ19] Filippo Bonchi, Robin Piedeleu, Pawel Sobociński, and Fabio Zanasi. Graphical affine algebra. In *LICS 2019*, pages 1–12. IEEE, 2019.

[BSZ14a] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. Interacting bialgebras are Frobenius. In Anca Muscholl, editor, *FOSSACS 2014*, volume 8412 of *LNCS*, pages 351–365. Springer, 2014.

[BSZ14b] Filippo Bonchi, Pawel Pawel Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In Paolo Baldan and Daniele Gorla, editors, *CONCUR 2014*, volume 8704 of *LNCS*, pages 435–450. Springer, 2014.

[BSZ15] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. Full abstraction for signal flow graphs. In *POPL 2015*, pages 515–526. ACM, 2015.

[BSZ17a] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. The calculus of signal flow diagrams I: Linear relations on streams. *Information and Computation*, 252:2–29, 2017.

[BSZ17b] Filippo Bonchi, Pawel Sobociński, and Fabio Zanasi. Interacting Hopf algebras. *Pure and Applied Algebra*, 221(1):144 – 184, 2017.

[Bur93] Albert Burroni. Higher dimensional word problems with applications to equational logic. *Theoretical Computer Science*, 115(1):43–62, 1993.

[BvEG+87] Hendrik Pieter Barendregt, Marko C. J. D. van Eekelen, John R. W. Glauert, Richard Kennaway, Marinus J. Plasmeijer, and M. Ronan Sleep. Term graph rewriting. In Jacopus W. de Bakker, A. J. Nijman, and Philip C. Treleaven, editors, *PARLE 1987*, pages 141–158, Berlin, Heidelberg, 1987. Springer.

[CD08] Bob Coecke and Ross Duncan. Interacting quantum observables. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008*, volume 5126 of *LNCS*, pages 298–310. Springer, 2008.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

[CMR+97] Andrea Corradini, Ugo Montanari, Francesca Rossi, Hartmut Ehrig, Reiko Heckel, and Michael Löwe. Algebraic approaches to graph transformation - Part I: Basic concepts and double pushout approach. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*, pages 163–246. World Scientific, 1997.

[CPV13] Bob Coecke, Dusko Pavlovic, and Jamie Vicary. A new description of orthogonal bases. *Mathematical Structures in Computer Science*, 23(3):555–567, 2013.

[CW87] Aurelio Carboni and Robert F. C. Walters. Cartesian bicategories I. *Pure and Applied Algebra*, 49(1-2):11–32, 1987.

[DDK10] Lucas Dixon, Ross Duncan, and Aleks Kissinger. Open graphs and computational reasoning. In S. Barry Cooper, Prakash Panangaden, and Elham Kashefi, editors, *DCM 2010*, volume 26 of *EPTCS*, pages 169–180, 2010.

[DJ90] Nachum Dershowitz and Jean-Pierre Jouannaud. Rewrite systems. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier and MIT Press, 1990.

[DK13] Lucas Dixon and Aleks Kissinger. Open-graphs and monoidal theories. *Mathematical Structures in Computer Science*, 23(2):308–359, 2013.

[EK04] Hartmut Ehrig and Barbara König. Deriving bisimulation congruences in the DPO approach to graph rewriting. In Igor Walukiewicz, editor, *FOSSACS 2004*, volume 2987 of *LNCS*, pages 151–166. Springer, 2004.

[FS19] Brendan Fong and David I. Spivak. Hypergraph categories. *Pure and Applied Algebra*, 223:4746–4777, 2019.

[Gad07] Fabio Gadducci. Graph rewriting for the π-calculus. *Mathematical Structures in Computer Science*, 17(3):407–437, 2007.

[GH98] Fabio Gadducci and Reiko Heckel. An inductive view of graph transformation. In Francesco Parisi-Presicce, editor, *WADT 1997*, volume 1376, pages 223–237. Springer, 1998.

[HJKS11] Marvin Heumüller, Salil Joshi, Barbara König, and Jan Stückrath. Construction of pushout complements in the category of hypergraphs. In Rachid Echahed, Annegret Habel, and Mohamed Mosbah, editors, *GCM 2010*, volume 39 of *ECEASST*. EASST, 2011.

[HP07] Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. In Luca Cardelli, Marcelo P. Fiore, and Glynn Winskel, editors, *Computation, Meaning, and Logic*, volume 172 of *ENTCS*, pages 437–458. Elsevier, 2007.

[Hue80] Gérard Huet. Confluent reductions: Abstract properties and applications to term rewriting systems: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[JPV18] Emmanuel Jeandel, Simon Perdrix, and Renaud Vilmart. A complete axiomatisation of the ZX-calculus for Clifford+T quantum mechanics. In Anuj Dawar and Erich Grädel, editors, *LICS 2018*, pages 569–578. ACM, 2018.

[JS91] Andre Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991.

[JSV96] Andre Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119(3):447–468, 1996.

[Kas12] Christian Kassel. *Quantum Groups*. Springer, 2012.

[KB70] Donald E. Knuth and Peter B. Bendix. Simple word problems in universal algebras. In *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, 1970.

[Kis14] Aleks Kissinger. Finite matrices are complete for (dagger-)hypergraph categories. *Preprint available at arXiv1406.5942*, 2014.

[KvdW20] Aleks Kissinger and John van de Wetering. Pyzx: Large scale automated diagrammatic reasoning. In Bob Coecke and Matthew Leifer, editors, *QPL 2019*, volume 318 of *EPTCS*, pages 229–241. Open Publishing Association, 2020.

[KZ15] Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In Amy P. Felty and Aart Middeldorp, editors, *CADE 2015*, volume 9195 of *LNCS*, pages 326–336. Springer, 2015.

[Lac04] Steve Lack. Composing PROPs. *Theory and Application of Categories*, 13(9):147–163, 2004.

[Laf03] Yves Lafont. Towards an algebraic theory of Boolean circuits. *Pure and Applied Algebra*, 184(2–3):257–310, 2003.

[LS05] Steve Lack and Pawel Sobociński. Adhesive and quasiadhesive categories. *Theoretical Informatics and Applications*, 39(3):511–546, 2005.

[Mac65] Saunders Mac Lane. Categorical algebra. *Bulletin of the American Mathematical Society*, 71(1):40–106, 1965.

[Mim14] Samuel Mimram. Towards 3-dimensional rewriting theory. *Logical Methods in Computer Science*, 10(2), 2014.

[ML98] Saunders Mac Lane. *Categories for the Working Mathematician*. Springer, 1998.

[MS09] John MacDonald and Laura Scull. Amalgamations of categories. *Canadian Mathematical Bulletin*, 52:273–284, 2009.

[NW17] Kang Feng Ng and Quanlong Wang. A universal completion of the ZX-calculus. *Preprint available at abs/1706.09877*, 2017.

[Plu99] Detlef Plump. Term graph rewriting. In Hartmut Ehrig, Gregor Engels, Hans-Jorg Kreowski, and Grzegorz Rozenberg, editors, *Handbook of Graph Grammars and Computing by Graph Transformations, Vol. 2: Applications, Languages, and Tools*, pages 3–61. World Scientific, 1999.

[Plu10] Detlef Plump. Checking graph-transformation systems for confluence. In Frank Drewes, Annegret Habel, Berthold Hoffmann, and Detlef Plump, editors, *Manipulation of Graphs, Algebras and Pictures*, volume 26 of *ECEASST*. EASST, 2010.

[PS81] Gerald E Peterson and Mark E Stickel. Complete sets of reductions for some equational theories. *Journal of the ACM*, 28(2):233–264, 1981.

[RSW05] Robert Rosebrugh, Nicoletta Sabadini, and R. F. C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and Application of Categories*, 17(6):164–177, 2005.

[Sel10] Peter Selinger. Autonomous categories in which $A = A^*$. In *QPL 2010*, pages 151–160, 2010.

[Sel11] Peter Selinger. A survey of graphical languages for monoidal categories. *Springer Lecture Notes in Physics*, 13(813):289–355, 2011.

[Sob] Pawel Sobocinski. Graphical Linear Algebra, https://graphicallinearalgebra.net/.

[SS05] Vladimiro Sassone and Pawel Sobociński. Reactive systems over cospans. In *LICS 2005*, pages 311–320. IEEE Computer Society, 2005.

[Vis01] Eelco Visser. A survey of rewriting strategies in program transformation systems. In Bernhard Gramlich and Salvador Lucas, editors, *WRS 2001*, volume 57 of *ENTCS*, pages 109–143. Elsevier, 2001.

[Zan15] Fabio Zanasi. *Interacting Hopf Algebras: The Theory of Linear Systems*. PhD thesis, École Normale Supérieure de Lyon, 2015.