

Rewriting Graphically with Symmetric Traced Monoidal Categories

George Kaye

School of Computer Science, University of Birmingham, UK

March 19, 2021

Abstract

We examine a variant of hypergraphs that we call *interfaced linear hypergraphs*, with the aim of creating a sound and complete graphical language for symmetric traced monoidal categories (STMCs) suitable for graph rewriting. In particular, we are interested in rewriting for categorical settings with a *Cartesian* structure, such as digital circuits. These are incompatible with previous languages where the trace is constructed using a compact closed or Frobenius structure, as combining these with Cartesian product can lead to degenerate diagrams. Instead we must consider an approach where the trace is constructed as an atomic operation. Interfaced linear hypergraphs are defined as regular hypergraphs in which each vertex is the source and target of exactly one edge each, equipped with an additional interface edge. The morphisms of a freely generated STMC are interpreted as interfaced linear hypergraphs, up to isomorphism (soundness). Moreover, any linear hypergraph is the representation of a unique STMC morphism, up to the equational theory of the category (completeness). This establishes interfaced linear hypergraphs as a suitable combinatorial language for STMCs. We then show how we can apply the theory of adhesive categories to our graphical language, meaning that a broad range of equational properties of STMCs can be specified as a graph rewriting system. The graphical language of digital circuits is presented as a case study.

1 Introduction

Constructors, architects, and engineers have always enjoyed using blueprints, diagrams, floorplans and other kinds of graphical representations of their designs. These are often more than simply illustrations aiding the understanding of a formal specification, they are the specification itself. By contrast, in mathematics, diagrams have not been traditionally considered first-class citizens, although they are often used to help the reader *visualise* a construction or a proof. However, the development of new *formal* diagrammatic languages for a variety of systems such as quantum communication and computation [12], computational linguistics [13] and signal-flow graphs [3, 4], proved that diagrams can be used not just to aid understanding of proofs, but also to formulate proofs. This formulation has multifaceted advantages, from enabling the use of graph-theoretical techniques to aid reasoning [20] to making the teaching of algebraic concepts to younger students less intimidating [18].

These graphical languages build on a mathematical infrastructure of (usually symmetric and strict) monoidal categories [27], and in particular compact closed categories [29]. Systems modelled by morphisms in a compact closed category have a general notion of *interface port*, so that any two ports can be connected, provided the types match. This allows compact closed categories to describe systems with a flexible and refined notion of *causality*, such as quantum systems [32] or games [9]. In contrast, systems such as digital circuits have a stricter notion of causality, enforcing that connections may only happen between ports with the same type but opposite input-output polarities. This requires a different kind

of categorical setting, that of a *symmetric traced monoidal category* [22], or STMC. These categories come equipped with an explicit construct (the trace) to model causal feedback loops.

String diagrams [39] are becoming the established mathematical language of diagrammatic reasoning, whereby equal terms are usually interpreted as isomorphic (or isotopic) diagrams. While this is enough for reasoning about *structural* properties, properties which have computational content require a *rewriting* of the diagram. To make this possible, diagrams must be represented as combinatorial objects, such as graphs or hypergraphs, which have enough structure. The framework of *adhesive categories* is of particular interest to us [35], as it implies that graph rewriting is always well-defined.

Our main motivation is to fully formalise prior work on diagrammatic reasoning for digital circuits [19, 20], for which we need a string diagram language of STMCs along with adhesive categorical infrastructure for rewriting. It might seem that this is a solved problem, as combinatorial languages for graph rewriting have already been studied as *open graphs* [14, 30] and *hypergraphs* [5, 44, 7], which satisfy soundness and completeness. However, the completeness theorem raises for us insurmountable technical problems, which we set to overcome in this paper. In *loc.cit.* STMCs are constructed by embedding them into the more expressive setting of a SMC equipped with a *Frobenius structure*, which induces a compact closed structure into which an STMC can be embedded. To reason about digital circuits we require the framework of *dataflow categories*, which are STMCs in which the monoidal tensor is a Cartesian product [11, 21]. It is the interaction between the diagonal morphism of the Cartesian product and the Frobenius structure which is problematic.

In general it is well known that in compact closed categories finite products automatically become *biproducts* [26]. This is enough to compromise the construction as a setting for modelling digital circuits, which do not physically satisfy the equational properties of a biproduct. But the problem runs deeper, as the Frobenius structure itself is not compatible with Cartesian product, as seen in the diagram below:



On the left, the Frobenius structure equates the splitting and joining of the wires with a feedback loop, implementing a trace structure. On the right, the splitting of the wires copies the co-unit of the Frobenius co-monoid, resulting in a degenerate circuit. To solve this problem we need to define the trace structure directly, and prove soundness and definability for these direct definitions.

Besides the major problem above, there are some small technical issues with hypergraphs that we solve by reintroducing the concept of homeomorphism similar to that used in framed point graphs [30]. This allows us to represent the trace of the identity, which is not well-formed in vanilla hypergraphs as it is a closed loop of wires. It also means we can identify a matching of a subgraph F in a graph $\text{Tr}^x(F)$ by using a monomorphism, which is essential for performing double pushout (DPO) graph rewriting.

The primary contributions of this paper are therefore as follows: we refine the definition of hypergraphs in [5] in order to define a *sound* and *complete* graphical language for STMCs that does not become degenerate in the presence of Cartesian structure. We show that this language can be used with the framework of adhesive categories, so any additional axioms can be expressed as graph rewrite rules without any ambiguity. This allows us to make the proofs in [20] rigorous.

1.1 Structure of the report

The structure of the report is as follows. In §2 we recap the required background on monoidal categories, and in particular Cartesian and symmetric traced monoidal categories, the primary focus of our work. In §3 we introduce a standard definition of hypergraphs, and then refine this to obtain *linear hypergraphs*, which are motivated by our study of string diagrams. §4 details several hypergraph constructs and operations that will be of use to us. We then use these ingredients in §5 to show that we can represent morphisms in a free PROP (a category of PROducts and Permutations, where objects are natural numbers) as hypergraphs. We take the opposite perspective in §6, to show that we can also recover categorical

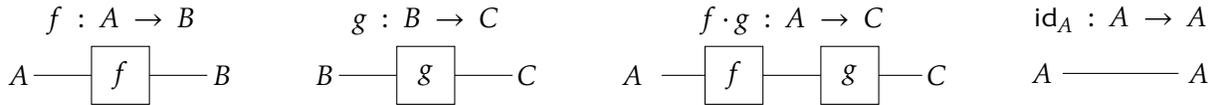
terms from hypergraphs, enabling us to conclude both soundness and completeness. In §7 we study graph rewriting, a useful application of our graphical language, and in §8 follow with a case study into the axioms related to digital circuits. Finally in §9 we generalise our approach to consider terms from any STMC, not just PROPs. The finer details of some of the more bureaucratic proofs can be found in the appendices.

1.2 Notation

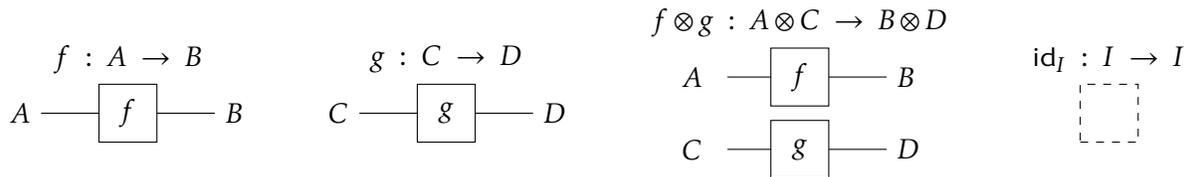
Let $|X|$ be the cardinality of a set X . We write $[n] \subset \mathbb{N}$ as the subset of the natural numbers containing $0, 1, \dots, n-1$. For two sets X and Y , let $X+Y$ be their disjoint union and $X-Y$ be the relative complement of Y in X . Let (X, \leq^X) be a totally ordered set, where usually we will just write the carrier X . We take the convention that the order is defined by the order the elements are written, i.e. in $\{x, y, z\}$, $x < y < z$ and for $\{x, y\} \cup \{z, w\}$, $x < y < z < w$. We use π_i as a ‘projection’ function to denote the i th element of a totally ordered set. For two functions $f : X \rightarrow Y$ and $g : U \rightarrow V$, we denote as $f + g : X + U \rightarrow Y + V$ their disjoint union that acts as f on elements of X and as g on elements of U .

2 Monoidal categories

We begin by recapping the concepts of monoidal categories. A category \mathcal{C} is a collection of objects A, B, C, \dots with morphisms f, g, h, \dots between them. A morphism f between objects A and B is denoted $f : A \rightarrow B$. The morphisms between each pair of objects $A \rightarrow B$ form a *hom-set*, denoted $\mathcal{C}(A, B)$. Each object is equipped with an identity morphism $\text{id}_A : A \rightarrow A$. Morphisms can be composed sequentially: if we have morphisms $f : A \rightarrow B$ and $g : B \rightarrow C$ we also have the morphism $f \cdot g : A \rightarrow C$. Composition is associative ($f \cdot (g \cdot h) = (f \cdot g) \cdot h$) and unital ($\text{id} \cdot f = f = f \cdot \text{id}$). In the language of string diagrams, we represent morphisms as boxes, and composition by horizontal juxtaposition. The identity is drawn as an empty wire. Equal morphisms in the category correspond to isomorphic diagrams – ‘only connectivity matters’.



A *monoidal* category [27] introduces a new binary operation known as the *monoidal tensor*, denoted $- \otimes -$. The unit object of the monoid is denoted I . Much like sequential composition, the tensor is associative ($(f \otimes g) \otimes h = f \otimes (g \otimes h)$) and unital with respect to the identity of the unit object ($\text{id}_I \otimes f = f = f \otimes \text{id}_I$). When we write categorical terms, \otimes binds tighter than \cdot , so $f \otimes g \cdot h \otimes k$ should be read as $(f \otimes g) \cdot (h \otimes k)$. Graphically, the tensor is drawn as vertical juxtaposition and the unit object is drawn as ‘empty space’.



The addition of tensor means that there are multiple ways in which we can compose morphisms in sequence or in parallel that lead to equal terms. This is known as *functoriality*, and can be expressed as the following axiom: $(f \cdot g) \otimes (h \cdot k) = f \otimes h \cdot g \otimes k$. Functoriality means that using the one dimensional algebraic notation can obfuscate the true nature of the inherently two dimensional structure. This is especially important computationally, as numerous extra operations must be performed to manipulate a term appropriately. Fortunately, the graphical notation eliminates this overhead, as both terms correspond to the same diagram:

$$(f \otimes h) \cdot (g \otimes k) : A \otimes D \rightarrow C \otimes F$$

To acquire a framework suitable for modelling systems, we need a way of crossing over the wires in our diagrams. This is achieved by equipping each pair of objects A, B in our category with a *symmetry* $\sigma_{A,B} : A \otimes B \rightarrow B \otimes A$. A category in *symmetric monoidal category* (or SMC), and braidings are called *symmetries*. The symmetry satisfies the axioms of *naturality* $f \cdot g \cdot \sigma_{B,D} = \sigma_{A,C} \cdot f \otimes g$, *hexagon* $\sigma_{A,B} \otimes \text{id}_C \cdot \text{id}_A \otimes \sigma_{B,C} = \sigma_{A,B \otimes C}$ and *self-inverse* $\sigma_{A,B} \cdot \sigma_{B,A} = \text{id}_A \otimes \text{id}_B$, illustrated below.

We are particularly interested in *free monoidal categories*, where morphisms, or ‘terms’, are generated over a *monoidal signature* $\Sigma = (\Sigma_O, \Sigma_M)$: a set of object variables and morphism variables (*generators*), equipped with functions $\text{dom}, \text{cod} : \Sigma_M \rightarrow \Sigma_O^*$, where Σ_O^* is a list of object variables, denoting the domain and codomain of each generator. Effectively, generators are the building blocks from which we can form categorical terms, by composing generators in sequence or parallel with each other, identity morphisms and symmetries. For example, the free monoidal category generated over the signature $\Sigma = \{f : X \rightarrow B \otimes C, g : B \otimes A \rightarrow X\}$ contains the following term:

$$f \otimes \text{id}_A \cdot \text{id}_B \otimes \sigma_{C,A} \cdot g \otimes \text{id}_C$$

Lemma 1 (Staging). *Any morphism $f \in \mathbf{Term}_\Sigma$ can be written as in the form $f = f_0 \cdot f_1 \cdot \dots \cdot f_n$, where f_i is a tensor containing only one non-identity morphism, $f_i = \text{id}_p \otimes k \otimes \text{id}_q$.*

Proof. By functoriality and unitality. □

A useful class of symmetric monoidal categories are called *PROPs* (PROduct and Permutation categories), categories with natural numbers as objects and addition as tensor product. These are especially natural with regards to graphical notation as an object $n \in \mathbb{N}$ can be drawn as n wires.

Lemma 2 (Composite symmetry). *Any symmetry $\sigma_{m,n}$ in a free PROP can be expressed as a combination of multiple symmetries $\sigma_{1,1}$ and identities.*

Proof. By the hexagon axiom. □

2.1 Symmetric traced monoidal categories

So far, the wires in our string diagrams have only travelled in one direction across the page: from left to right. However to model some systems we may want to ‘bend’ these wires, such as to model feedback. A common way of doing this is to use a *compact closed category* [29], in which every object A has a *dual* A^* , drawn as a wire travelling from right to left. Each object is also equipped with additional structural morphisms known as the *cup* $: A^* \otimes A \rightarrow I$ and the *cap* $: I \rightarrow A \otimes A^*$ for ‘bending’ wires. However, this setting is not suitable for all applications. In a compact closed category there is a flexible notion of causality, where morphisms do not have so much a notion of input and output but rather a bidirectional

interface port. Instead, we may wish to enforce a strict notion of causality, where only outputs of morphisms can connect to inputs. To do this, we must look at a flavour of monoidal categories known as *symmetric traced monoidal categories* (or STMCs for short), which were introduced by Joyal et al. [28] and refined by Hasegawa [22].

An STMC is an SMC with an extra family of operations known as *trace operators*. For a morphism $f : X \otimes A \rightarrow X \otimes B$, we can *trace* it to form the morphism $\text{Tr}_{A,B}^X(f) : A \rightarrow B$. We will often drop the subscript for clarity when there is no ambiguity. A trace is represented graphically by ‘bending around’ one of the output wires to join up with one of the input wires. This enables wires to travel in the opposite direction for a period, but all wires must still be oriented left-to-right when interacting with morphisms, as shown below:

$$\begin{array}{c} X \\ A \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} X \\ B \end{array} \xrightarrow{\text{Tr}_{A,B}^X(f)} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} B \\ \end{array}$$

There are several (equivalent) formulations of the axioms of STMCs, but here we present the four detailed by Hasegawa in [22].

Tightening

$$\text{Tr}_{A,D}^X(\text{id}_X \otimes g \cdot f \cdot \text{id}_X \otimes h) = g \cdot \text{Tr}_{B,C}^X(f) \cdot h$$

$$\begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{g} \\ \boxed{g} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{h} \\ \boxed{h} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} D \quad = \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{g} \\ \boxed{g} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{h} \\ \boxed{h} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} D$$

Yanking

$$\text{Tr}_{X,X}^X(\sigma_{X,X}) = \text{id}_X$$

$$\begin{array}{c} \text{---} \\ \text{---} \end{array} X \begin{array}{c} \text{---} \\ \text{---} \end{array} X \quad = \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} X \text{---} X$$

Superposing

$$\text{Tr}_{A \otimes C, B \otimes C}^X(f \otimes \text{id}_C) = \text{Tr}_{A,B}^X(f) \otimes \text{id}_C$$

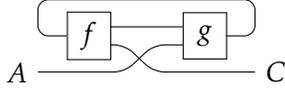
$$\begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} B \\ C \end{array} \quad = \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} B \\ C \end{array}$$

Exchange

$$\text{Tr}_{A,B}^Y(\text{Tr}_{Y \otimes A, Y \otimes B}^X(f)) = \text{Tr}_{A,B}^X(\text{Tr}_{X \otimes A, X \otimes B}^Y(\sigma_{Y,X} \otimes \text{id}_A \cdot f \cdot \sigma_{X,Y} \otimes \text{id}_B))$$

$$\begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} B \quad = \quad \begin{array}{c} \text{---} \\ \text{---} \end{array} \begin{array}{c} \boxed{f} \\ \boxed{f} \end{array} \begin{array}{c} \text{---} \\ \text{---} \end{array} B$$

As with regular symmetric monoidal categories, we can generate *free* STMCs over a given signature with the addition of the trace operator. For example, the free STMC defined over Σ contains the following term:

$$\text{Tr}^X(\langle \otimes \text{id}_A \cdot \text{id}_B \otimes \sigma_{D,A} \cdot \rangle \otimes \text{id}_D)$$


We can also derive one other important lemma that holds in any free STMC.

Lemma 3 (Global trace). *For any morphism $f \in \mathbf{Term}_\Sigma$, we can represent it as $\text{Tr}^x(\hat{f})$, where \hat{f} is a morphism containing no trace.*

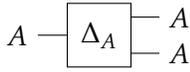
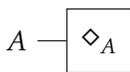
Proof. By superposing and tightening. □

2.2 Monoidal theories

On their own, the axioms of symmetric traced monoidal categories are not particularly interesting. To model systems we need to impose additional structure on our categories, which can be done with the introduction of new axioms. A *monoidal theory* is a monoidal signature equipped with a set of equations: pairs of terms with equal domain and codomain, e.g. for generators $f, g : A \rightarrow A$, a suitable equation could be $f \cdot g = g \cdot f$. Well-known monoidal theories include those of commutative monoids, Frobenius monoids and non-commutative monoids, which contain various combinations of generators for forking and splitting wires (see [5, Example 2.1] for details). Monoidal theories can also be used to model the operational semantics of compositional systems: the generators are the building blocks of that system and the axioms represent the operational semantics that we can use to reduce complex systems into simpler ones. In §8 we will examine a theory at the centre of our research, that of *digital circuits*. For now, we will present an example that motivate our work.

2.2.1 Example: Cartesian categories

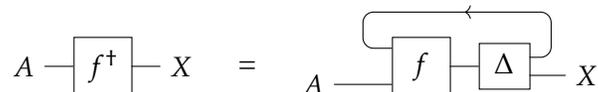
A *Cartesian category* is a symmetric monoidal category where each object is equipped with a *diagonal* morphism $\Delta_A : A \rightarrow A \otimes A$, and where the unit object is *terminal*: for every object A , there is a unique morphism $\diamond_A : A \rightarrow I$.

$$\Delta_A : A \rightarrow A \otimes A \qquad \diamond_A : A \rightarrow I$$



In essence, a Cartesian category is a monoidal category in which the tensor product is the Cartesian product. In the Cartesian monoidal theory, the families of diagonals and terminal morphisms are the generators; the accompanying axioms can be seen in Table 1.

Cartesian categories that are also traced are known as *dataflow categories* [39, §6.4]. The interaction of the trace with the Cartesian product is especially interesting, as it admits a *fixpoint operator*, as noticed by Hasegawa [21] and Martin Hyland independently. Equivalent observations had also been made before the introduction of traced monoidal categories, such as by Bloom and Ésik [2] and Ştefănescu [43].

Theorem 4 (Trace-fixpoint correspondence [21]). *A Cartesian category \mathcal{C} is traced if and only if it has a family of functions $(-)^{\dagger} : \mathcal{C}(A \otimes X, X) \rightarrow \mathcal{C}(A, X)$*

$$A \text{ --- } \boxed{f^{\dagger}} \text{ --- } X \quad = \quad A \text{ --- } \boxed{f} \text{ --- } \boxed{\Delta} \text{ --- } X$$


such that the following axioms are satisfied:

Naturality axioms

$$\begin{aligned} f \cdot \Delta_B &= \Delta_A \cdot f \otimes f & A &\rightarrow B \otimes B \\ f \cdot \diamond_B &= \diamond_A & A &\rightarrow 0 \end{aligned}$$

Commutative comonoid axioms

$$\begin{aligned} \Delta_A \cdot \Delta_A \otimes A &= \Delta_A \cdot (A \otimes \Delta_A) & A &\rightarrow A \otimes A \otimes A \\ \Delta_A \cdot (\diamond_A \otimes A) &= A & A &\rightarrow A \\ \Delta_A \cdot (A \otimes \diamond_A) &= A & A &\rightarrow A \\ \Delta_A \cdot \sigma_{A,A} &= \Delta_A & A &\rightarrow A \otimes A \end{aligned}$$

Coherence axioms

$$\begin{aligned} \Delta_0 &= 0 & 0 &\rightarrow 0 \\ \Delta_{A \otimes B} \cdot (A \otimes \sigma_{A,B} \otimes B) &= \Delta_A \otimes \Delta_B & A \otimes B &\rightarrow A \otimes B \otimes A \otimes B \\ \diamond_A &= 0 & 0 &\rightarrow 0 \\ \diamond_{A \otimes B} &= \diamond_A \otimes \diamond_B & A \otimes B &\rightarrow 0 \end{aligned}$$

Table 1: The axioms for a Cartesian monoidal category [39]

Naturality

$$(\text{id}_X \otimes g \cdot f)^\dagger = g \cdot (f)^\dagger$$

Dinaturality

$$(\Delta_{X \otimes A} \cdot g \otimes \diamond_X \otimes \text{id}_A \cdot f)^\dagger = \Delta_A \cdot (\Delta_{X \otimes A} \cdot g \otimes \diamond_X \cdot f)^\dagger \otimes \text{id}_A \cdot f$$

Diagonal

$$(\Delta_X \otimes \text{id}_A \cdot f)^\dagger = ((f)^\dagger)^\dagger$$

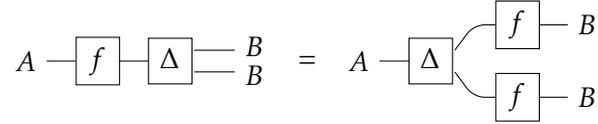
We can use the fixpoint operator to model *feedback* in our systems. In particular, we can derive the slightly simpler *fixed-point equation* from the dinaturality axiom [22] that allows us to ‘unfold’ the fixpoint.

$$f^\dagger = \Delta_A \cdot f^\dagger \otimes \text{id}_A \cdot f$$

2.2.2 Graphical reasoning with monoidal theories

The reason that graphical languages are so useful when dealing with monoidal categories is that the axioms are absorbed into the notation, and the tedious bureaucracy is eliminated. Unfortunately, once we start adding extra structure this starts to fall apart. For example, take the example of the naturality of the Cartesian diagonal.

$$f \cdot \Delta_B = \Delta_A \cdot f \otimes f$$



Clearly, this axiom cannot be absorbed by the graphical notation: even the number of boxes differs! To tackle these axioms, we must consider diagrams not just up to isomorphism, but up to *rewriting*. To do this, we must move away from the topological string diagrams and towards a more combinatorial diagram, where vertices and edges are explicitly defined. With these diagrams we can perform *graph rewriting*, of which numerous formalisms and frameworks exist [15, 16, 35].

As we have already observed in the introduction, this is not a new endeavour: previously this has been studied with string graphs [14, 30] and hypergraphs [5, 44, 7, 8]. However, these are rooted in compact closed categories, which are incompatible with the Cartesian product. This is because the Cartesian product automatically becomes a *biproduct* in a compact closed setting [26], which is not always suitable (e.g. in the category of digital circuits detailed in §8). Therefore, we will need to define a slightly different combinatorial structure.

3 Hypergraphs

We begin by recalling a standard notion of hypergraphs in which edges have ordered sources and targets, as in [5]. Let \mathbb{A} be a countably infinite set of *atoms* (or *names*, in the sense of [37]).

Definition 5 (Hypergraph). *A hypergraph is a tuple $H = (V, E, s, t)$ where*

- $V \subset \mathbb{A}$ is a finite set of vertices.
- E is a set containing, for each $k, l \in \mathbb{N}$, finite sets $E[k, l]$ of hyperedges with k sources and l targets.
- s, t are families of functions denoting sources and targets of edges, i.e. for each $E[k, l]$:
 - for each $i < k$, there exists the i th source map $s[i] : E[k, l] \rightarrow V$
 - for each $i < l$, there exists the j th target map $t[i] : E_{k,l} \rightarrow V$

We call the *in-degree*, written $\text{in}(v)$ (resp. *out-degree*, written $\text{out}(v)$) of a vertex the number of edges it is the target (resp. source) of. We call a hypergraph *discrete* if it has no edges. To reduce our use of space, for a hypergraph $H = (V, E, s, t)$ we will often use V_H, E_H etc. to access members of the tuple.

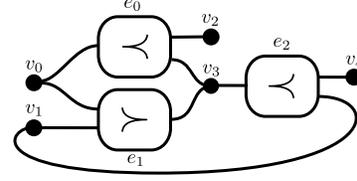
A *hypergraph signature* is a set of labels Σ equipped with functions $\text{dom}, \text{cod} : \Sigma \rightarrow \mathbb{N}$. A *labelled hypergraph* over signature Σ is a hypergraph $H = (V, E, s, t)$ and a labelling function $\Lambda : E \rightarrow \Sigma$, such that for any $e \in E$, if $\Lambda(e) = l$ then $\text{dom}(l) = |s(e)|$ and $\text{cod}(l) = |t(e)|$.

Example 6. *Below there is an informal drawing of a hypergraph over the hypergraph signature*

$$\Sigma = \{< : 1 \rightarrow 2, > : 2 \rightarrow 1\}.$$

Vertices are drawn as black dots. Edges are drawn as boxes, with ordered sources and targets connected on the left and right respectively.

$$\begin{aligned}
 V &= \{v_0, v_1, v_2, v_3, v_4\} & E[1, 2] &= \{e_0, e_2\} & E[2, 1] &= \{e_1\} \\
 s[0](e_0) &= v_0 & t[0](e_0) &= v_2 & t[1](e_0) &= v_3 \\
 s[0](e_1) &= v_0 & s[1](e_1) &= v_1 & t[0](e_1) &= v_3 \\
 s[0](e_2) &= v_3 & t[0](e_2) &= v_4 & t[1](e_2) &= v_1 \\
 \Lambda &= \{e_0 \mapsto \langle, e_1 \mapsto \rangle, e_2 \mapsto \langle\}
 \end{aligned}$$



Category. A labelled hypergraph homomorphism $h : F \rightarrow G$ consists of functions $h_V : V_F \rightarrow V_G$ and, for each $k, l \in \mathbb{N}$, $h_E : E_F[k, l] \rightarrow E_G[k, l]$ such that sources, targets and labels are preserved.

$$\begin{array}{ccccc}
 E_F[k, l] & \xrightarrow{s_F[i]} & V_F & E_F[k, l] & \xrightarrow{t_F[i]} & V_F & E_F & \xrightarrow{\Lambda_F} & \Sigma \\
 \downarrow h_E & & \downarrow h_V & \downarrow h_E & & \downarrow h_V & \downarrow h_E & & \downarrow \text{id} \\
 E_G[k, l] & \xrightarrow{s_G[i]} & V_G & E_G[k, l] & \xrightarrow{t_G[i]} & V_G & E_G & \xrightarrow{\Lambda_G} & \Sigma
 \end{array}$$

If h_V and h_E are bijective then F and G are isomorphic $F \equiv G$. It is immediate that \equiv is an equivalence relation, and we quotient hypergraphs by it.

Hypergraph homomorphisms are the morphisms in the category of hypergraphs **Hyp**, a functor category [5]. Hypergraph signatures Σ can be seen as hypergraphs, with a vertex v and edges for each label $m \rightarrow n$ in the signature, with v appearing m (resp. n) times in its sources (resp. targets). Thus labelled hypergraphs are defined as a slice category.

Definition 7 (Category of hypergraphs [5]). *Let **Hyp** be the functor category $[X, \mathbf{Set}]$, where X has as objects pairs of natural numbers (m, n) and an extra object \star . For each object $x = (m, n)$, there are $m + n$ arrows from x to \star . Let $\mathbf{Hyp}_\Sigma = \mathbf{Hyp}/\Sigma$ be the slice category over a hypergraph signature Σ .*

We call a hypergraph homomorphism an *embedding* if its components are injective.

Lemma 8. *A morphism in \mathbf{Hyp}_Σ is a monomorphism if and only if is an embedding.*

Proof. For a morphism $m : F \rightarrow G$ to be mono, for any two morphisms $p, q : H \rightarrow F$ (for any other hypergraph H), if $m \circ p = m \circ q$ then $p = q$. First we show that if m is an embedding it must be mono. If we consider each equivalence map of m separately, this means that we must show that $m_V \circ p_V = m_V \circ q_V$ implies $p_V = q_V$ (and the same for m_E). But we have assumed that m_T is injective, so the antecedent reduces to $g_T = h_T$. So m is mono.

Conversely, if m is not an embedding, it cannot be a monomorphism. A morphism that is not an embedding maps multiple vertices or edges into one. Therefore for a morphism $m : F \rightarrow G$ that maps vertices v_1 and v_2 in F to v in G , there exist two morphisms $p, q : G \rightarrow F$ (in $p, v \mapsto v_1$ and in $q, v \mapsto v_2$), and similar for morphisms that map multiple edges to one. Therefore there exist p, q such that $p \neq q$, so m is not mono. \square

3.1 Linear hypergraphs

In hypergraphs, vertices can connect to an arbitrary number of edges. However, to make wires in string diagrams split or join, an additional Frobenius structure must be imposed. This structure works particularly well in the framework of compact closed categories, but this is a structure which we aim to avoid. Therefore we must restrict hypergraphs so that the in-degree and out-degree of each vertex is at most one: vertices with in-degree 0 represent the inputs of the term and vertices with out-degree 0 represent the outputs of the term. We call a hypergraph *linear* if this condition is satisfied.

While we identified the input and output vertices of the hypergraph above, they are not ordered, and thus we do not have a true ‘interface’. One option is to identify the interfaces by means of certain

(ordered) cospans, as in [5], but we take an alternative approach in which we build our interfaces directly into our hypergraphs by means of an additional interface edge \bullet . We write the set of edges and this interface as $E + 1$.

We could simply add this edge to our existing definition of hypergraphs. However, we wish to define a *sound and complete* graphical language: we want *every* diagram to correspond to a term in our category. Therefore we take this opportunity to reformulate our definition of hypergraphs, yielding *interfaced linear hypergraphs*. In Section 7 we shall see how our definition can be related the more traditional definition.

Definition 9 (Interfaced linear hypergraph). *An interfaced linear hypergraph is a tuple $H = (E, S, T, \kappa)$ where*

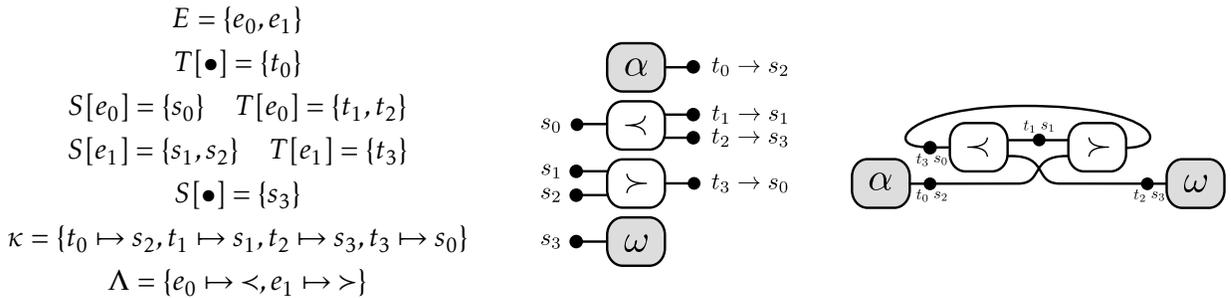
- A finite set $E \subset \mathbb{A}$ of edges
- S, T are finite sets containing, for each edge $e \in E + 1$, finite totally ordered sets of source and target vertices $S[e], T[e] \subset \mathbb{A}$, such that for any $V_1, V_2 \in S \cup T$, $V_1 \cap V_2 = \emptyset$.
- $\kappa : \bigcup_{e \in E} T[e] \rightarrow \bigcup_{e \in E} S[e]$ is a connections bijection between targets and sources.

We split vertices into sets of sources S and targets T , with a connections bijection κ between them. The ordering of sources and targets of edges is determined by the order of the sets. Splitting the vertices in this way allows us to enforce that each vertex is the source and target of only one edge while still retaining the order of sources and targets for each edge. It also simplifies the operations defined below: for example, when we compose hypergraphs we ‘coalesce’ the outputs of F and inputs of G together, as explained in Section 4.2. With one set of vertices, we could delete the outputs of F and the inputs of G , and then define ‘fresh’ vertices as the bridge between the two hypergraphs. However, we would have to redefine the orders on the vertices such that the ordering on the non-interface edges was preserved. Keeping sources and targets separate eliminates this problem. To simplify notation when talking about members of the source and target sets, we will use lower case variables to denote a single vertex, i.e. $s \in S$ means $s \in \bigcup_{e \in E} S[e]$. $T[\bullet]$ is the set of *inputs*, and $S[\bullet]$ is the set of *outputs*.

For an interfaced linear hypergraph H with m inputs and n outputs, we write it as $H : m \rightarrow n$, where $m \rightarrow n$ is the *type* of the hypergraph. As with simple hypergraphs, we can define *labelled linear hypergraphs* over a signature Σ with labelling function Λ , where $\Lambda(e) = \phi$ is valid only if $|S[e]| = \text{dom}(\phi)$ and $|T[e]| = \text{cod}(\phi)$.

Example 10. A linear hypergraph over Σ can be drawn in two ways, illustrated below. In a more formal notation (left), edges are stacked with their ordered source (resp. target) vertex sets on the left (resp. right) of the diagram. Connections are represented by the arrow on the far right. We represent the inputs (resp. outputs) of the term as incident to a grey edge labelled α (resp. ω).

A more intuitive representation (right) is similar to how we drew hypergraphs earlier, where we draw connected target and source vertices as a single black dot. The orders on the vertices dictate the position of each vertex’s connection to an edge. The more formal representation can be unambiguously recovered from the more intuitive one.



Category. A (labelled) linear hypergraph homomorphism $h : F \rightarrow G$ consists of functions

$$h_S : \bigcup_{e \in E_F} S_F[e] \rightarrow \bigcup_{e \in E_G} S_G[e] \quad h_T : \bigcup_{e \in E_F} T_F[e] \rightarrow \bigcup_{e \in E_G} T_G[e] \quad h_E : E_F \rightarrow E_G$$

between sources, targets and edges, such that the first four diagrams below commute. If h_T , h_S and h_E are bijective and the latter two diagrams below also commute, then F and G are *isomorphic* written $F \equiv G$. It is immediate that \equiv is an equivalence relation, and we quotient labelled interfaced linear hypergraphs by it.

$$\begin{array}{c}
 \text{homomorphism} \\
 \hline
 \begin{array}{ccccccc}
 E_F \xrightarrow{S_F[-]} S_F & E_F \xrightarrow{T_F[-]} T_F & \cup T_F \xrightarrow{\kappa_F} \cup S_F & E_F \xrightarrow{\Lambda_F} \Sigma & 1 \xrightarrow{T_F[-]} T_F & 1 \xrightarrow{S_F[-]} S_F \\
 \downarrow h_E & \downarrow h_E & \downarrow h_T & \downarrow h_E & \downarrow \text{id} & \downarrow \text{id} \\
 E_G \xrightarrow{S_G[-]} S_G & E_G \xrightarrow{T_G[-]} T_G & \cup T_G \xrightarrow{\kappa_G} \cup S_G & E_G \xrightarrow{\Lambda_G} \Sigma & 1 \xrightarrow{T_G[-]} T_G & 1 \xrightarrow{S_G[-]} S_G \\
 \downarrow h_E & \downarrow h_E & \downarrow h_T & \downarrow h_E & \downarrow \text{id} & \downarrow \text{id} \\
 E_G \xrightarrow{S_G[-]} S_G & E_G \xrightarrow{T_G[-]} T_G & \cup T_G \xrightarrow{\kappa_G} \cup S_G & E_G \xrightarrow{\Lambda_G} \Sigma & 1 \xrightarrow{T_G[-]} T_G & 1 \xrightarrow{S_G[-]} S_G
 \end{array} \\
 \hline
 \text{equivalence}
 \end{array}$$

Labelled interfaced linear hypergraphs form a category $\mathbf{LHyp}_\Sigma^\bullet$ with objects the labelled interfaced linear hypergraphs over signature Σ and morphisms the labelled interfaced linear hypergraph homomorphisms.

Lemma 11. *A morphism in $\mathbf{LHyp}_\Sigma^\bullet$ is mono if and only if it is an embedding.*

Proof. As with simple hypergraphs (Lemma 8). □

We will now use the term ‘hypergraph’ to mean ‘interfaced linear hypergraph’ unless specified.

4 Operations and constructs

We can create hypergraphs compositionally using the operations of an STMC: composition, monoidal tensor, symmetry and trace. In this section we will detail their definitions, in addition to some other important components of our hypergraph framework.

When performing operations, it is imperative that our hypergraphs do not become degenerate. We call a hypergraph *well-formed* if for any $V_1, V_2 \in S \cup T$, $V_1 \cap V_2 = \emptyset$, κ is bijective, and the labelling condition is satisfied. Some of the more bureaucratic proofs in this section have been omitted: to find them the interested reader can turn to Appendix A.

4.1 Equivariance

When performing operations on hypergraphs, the vertices and edges of the hypergraphs involved must be disjoint so that we do not create degenerate hypergraphs. However, this is not always the case, such as when composing a hypergraph with itself. Fortunately, since the sets of vertices and edges are subsets of the countably infinite set of atoms \mathbb{A} , we can simply *rename* the problematic edges or vertices [37].

Definition 12 (Action). *For any permutation $\tau : \mathbb{A} \rightarrow \mathbb{A}$, an action $\tau \odot -$ acts as follows:*

Element *For any elements $x \in \mathbb{A}$, $y \notin \mathbb{A}$, $\tau \odot x = \tau(x)$ and $\tau \odot y = y$.*

Set *For any set X , $\tau \odot X = \{\tau \odot x \mid x \in X\}$.*

Totally ordered sets *As with regular sets, preserving the order i.e. if $x < y$ then $\tau \odot x < \tau \odot y$*

Function *For any function $f : X \rightarrow Y$, $(\tau \odot f)(v) = \tau \odot f(\tau^{-1} \odot v)$.*

Definition 13 (Renaming). *For any labelled interfaced linear hypergraph*

$$F = (E, S, T, \kappa, \Lambda)$$

and for any permutation $\tau : \mathbb{A} \rightarrow \mathbb{A}$ we can apply τ to F to rename it:

$$\tau \odot F = (\tau \odot E, \tau \odot S, \tau \odot T, \tau \odot \kappa, \tau \odot \Lambda).$$

Proposition 14 (Equivariance of hypergraphs). *For any labelled interfaced linear hypergraph $F : m \rightarrow n$ and permutation $\tau : \mathbb{A} \rightarrow \mathbb{A}$, $\tau \odot F \equiv F$.*

Proof.

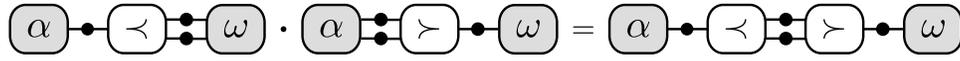
$$h_T(v) = \tau^{-1} \odot v \quad h_S(v) = \tau^{-1} \odot v \quad h_E(e) = \tau^{-1} \odot e$$

□

Therefore the definition of F is equivariant under name permutations, so we are justified in renaming vertices and edges ‘on the fly’. Since we use graphs up to isomorphism, this will implicitly also quotient by equivariance.

4.2 Composition

To compose hypergraphs sequentially, we ‘redirect’ any vertices that connected to the output of the first hypergraph to those originally connected to the input of the second hypergraph. Graphically, we juxtapose the hypergraphs horizontally:



Definition 15 (Composition). *For any two labelled interfaced linear hypergraphs $F : m \rightarrow n$ and $G : n \rightarrow p$ over signature Σ :*

$$F : m \rightarrow n = (E_F, S_F, T_F, \kappa_F, \Lambda_F) \quad G : n \rightarrow p = (E_G, S_G, T_G, \kappa_G, \Lambda_G)$$

we define their composition as follows.

$$H : m \rightarrow p = F \cdot G = (E_H, S_H, T_H, \kappa_H, \Lambda_H)$$

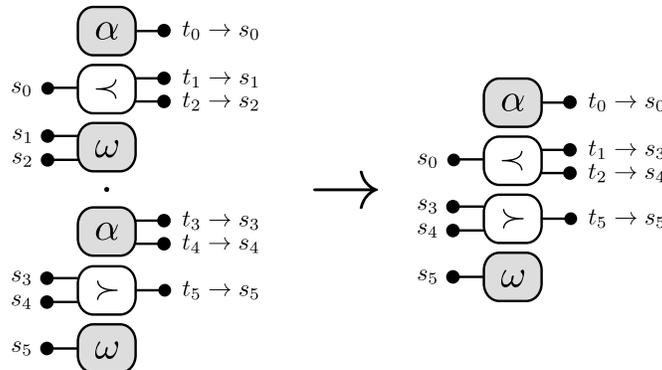
The new set of edges is simply the disjoint union of the edges in F and G , and we do the same for the labelling function. To obtain the new sets of vertices, we delete the outputs of F and the inputs of G .

$$\begin{aligned} E_H &= E_F + E_G & \Lambda_H &= \Lambda_F + \Lambda_G \\ S_H[\bullet] &= S_G[\bullet] & S_H[e \in E_F] &= S_F[e] & S_H[e \in E_G] &= S_G[e] \\ T_H[\bullet] &= T_F[\bullet] & T_H[e \in E_F] &= T_F[e] & T_H[e \in E_G] &= T_G[e] \end{aligned}$$

The connections function maps a vertex connected to the i th output of F to the vertex connected to the i th input of G .

$$\kappa_H(v) = \begin{cases} \kappa_G(\pi_i(T_G[\bullet])) & \text{if } \kappa_F(v) = \pi_i(S_F[\bullet]) \\ \kappa_F + \kappa_G & \text{otherwise} \end{cases}$$

This can be drawn formally as follows:



Proposition 16 (Well-formedness of composition). *For two labelled interfaced linear hypergraphs $F : m \rightarrow n$ and $G : n \rightarrow p$, $F \cdot G$ is a well-formed labelled interfaced linear hypergraph.*

Proof. We have only removed vertices so the sources and targets must still be disjoint. The only change in the connections function means that the vertices that originally connected to the output vertices of F (which have been deleted) now connect to the sources originally connected to the input vertices of G (which have also been deleted), so κ is bijective. The incidence of vertices on regular edges is also unaffected, so the labelling condition is satisfied. \square

The unit of composition is the identity hypergraph, a hypergraph where all vertices are the sources and the targets of the interface. Below are examples for $n = 1$ and $n = 2$.



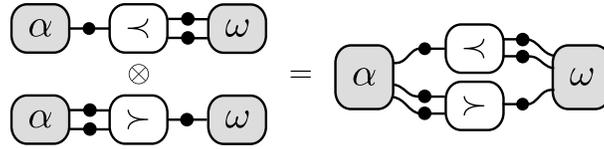
Definition 17 (Identity hypergraph). *An identity hypergraph $\text{id}_n : n \rightarrow n$ over signature Σ is defined as*

$$\text{id}_n = (\{S[\bullet]\}, \{T[\bullet]\}, \emptyset, \kappa, \emptyset)$$

where $S[\bullet], T[\bullet] \subset \mathbb{A}$ are finite disjoint totally ordered sets, $|S[\bullet]| = |T[\bullet]| = n$, and $\kappa(\pi_i(T[\bullet])) = \pi_i(S[\bullet])$.

4.3 Monoidal tensor

We can also compose hypergraphs in parallel, which is known as their *monoidal tensor*. We simply combine their input and outputs, and leave everything else untouched. Graphically, we can represent this by juxtaposing them vertically.



Definition 18 (Monoidal tensor). *For any two labelled interfaced linear hypergraphs $F : m \rightarrow n$ and $G : p \rightarrow q$ over a signature Σ :*

$$F = (E_F, S_F, T_F, \kappa_F, \Lambda_F) \quad G = (E_G, S_G, T_G, \kappa_G, \Lambda_G)$$

we define their monoidal tensor as follows.

$$H : m + p \rightarrow n + q = F \otimes G = \mathbf{LHyp}_{\Sigma}^{\bullet}[H]$$

Once again, the edges and labels are the union of those in F and G .

$$E_H = E_F + E_G \quad \Lambda_H = \Lambda_F + \Lambda_G$$

We do not need to delete any vertices, only combine the interfaces.

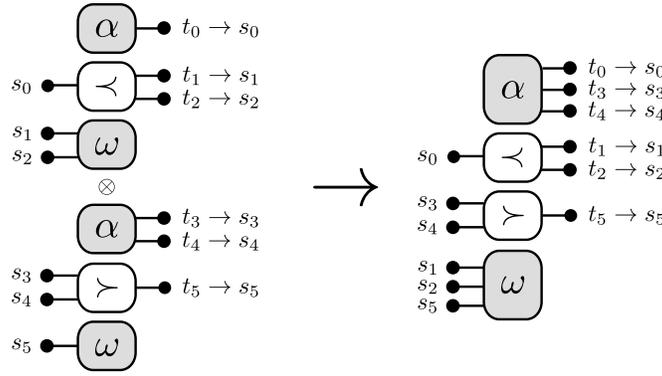
$$S_H[e \in E_F] = S_F[e] \quad S_H[e \in E_G] = S_G[e] \quad S_H[\bullet] = S_F[\bullet] + S_G[\bullet]$$

$$T_H[e \in E_F] = T_F[e] \quad T_H[e \in E_G] = T_G[e] \quad T_H[\bullet] = T_F[\bullet] + T_G[\bullet]$$

Subsequently the connections function is just the union of those in F and G .

$$\kappa_H = \kappa_F + \kappa_G$$

This can be drawn formally as follows:



Proposition 19 (Well-formedness of tensor). *For any two labelled interfaced linear hypergraphs $F : m \rightarrow n$ and $G : p \rightarrow q$, $F \otimes G$ is a well-formed labelled interfaced linear hypergraph.*

Proof. We have not added any new vertices, so the sources and targets are still disjoint. The connections of each hypergraph are unaffected, so κ is a bijection. Likewise, we have not interfered with the sources and targets of regular edges, so the labelling condition is satisfied. \square

The unit of monoidal tensor is the empty hypergraph (an identity hypergraph on 0). This is simply a hypergraph with no edges or vertices. Graphically this is represented as two empty interfaces.

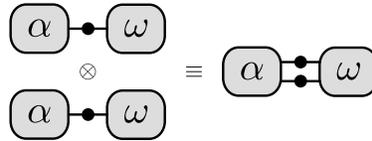


Definition 20 (Empty hypergraph). *The empty hypergraph $\text{id}_0 : 0 \rightarrow 0$ over signature Σ is defined as*

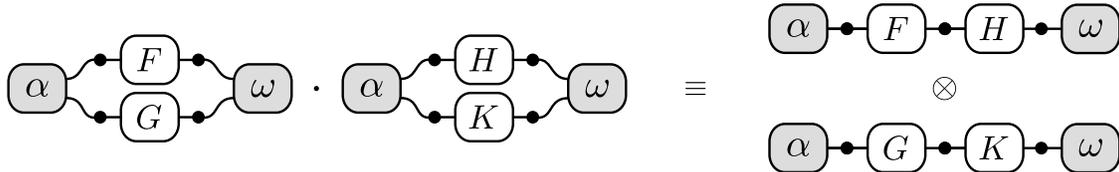
$$\text{id}_0 = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$$

$-\otimes-$ is a bifunctor, so there may be multiple orders in which we can perform sequential composition or monoidal tensor that still result in the same hypergraph.

Proposition 21 (Bifunctoriality I). *For any $m, n \in \mathbb{N}$, $\text{id}_m \otimes \text{id}_n \equiv \text{id}_{m+n}$*

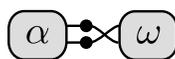


Proposition 22 (Bifunctoriality II). *For any labelled interfaced linear hypergraphs $F : m \rightarrow n$, $G : r \rightarrow s$, $H : n \rightarrow p$, $K : s \rightarrow t$, $F \otimes G \cdot H \otimes K \equiv (F \cdot H) \otimes (G \cdot K)$.*



4.4 Symmetry

To swap the orders of vertices in the interfaces, we require a new construct, named the *swap* hypergraph. This hypergraph swaps over two wires.



Definition 23 (Swap hypergraph). *The swap hypergraph for two wires $\sigma_{1,1}$ is defined as*

$$\sigma_{1,1} = (\emptyset, \{S[\bullet]\}, \{T[\bullet]\}, \kappa, \emptyset)$$

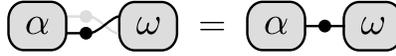
where $S[\bullet] = \{c, d\}$, $T[\bullet] = \{a, b\}$, $a, b, c, d \in \mathbb{A}$, $\kappa(a) = d$, $\kappa(b) = c$.

By composing multiple copies of the swap hypergraph in sequence and parallel we can build up constructs in which we swap many wires.

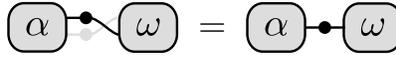
Definition 24 (Composite swap). *For any $m, n \in \mathbb{N}$, we can define a composite swap hypergraph as follows.*

$$\sigma_{m,n} : m + n \rightarrow n + m$$

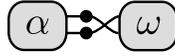
$$\sigma_{0,n} = \text{id}_n$$



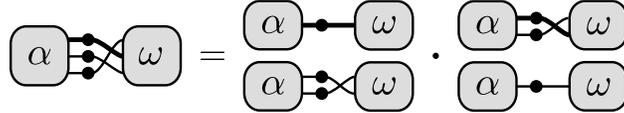
$$\sigma_{m,0} = \text{id}_m$$



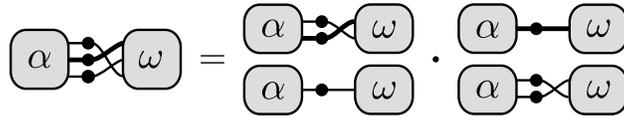
$$\sigma_{1,1} = \sigma_{1,1}$$



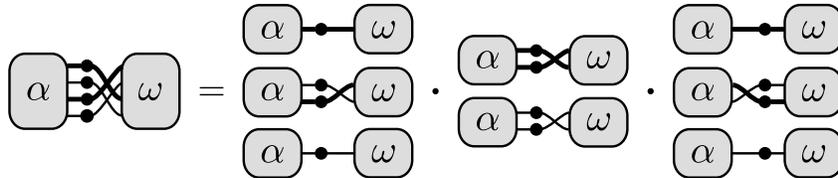
$$\sigma_{m+1,1} = m \otimes \sigma_{1,1} \cdot \sigma_{m,1} \otimes 1$$



$$\sigma_{1,n+1} = \sigma_{1,n} \otimes 1 \cdot n \otimes \sigma_{1,1}$$



$$\sigma_{m+1,n+1} = 1 \otimes \sigma_{1,n} \otimes 1 \cdot \sigma_{m,n} \otimes \sigma_{1,1} \cdot n \otimes \sigma_{m,1} \otimes 1$$



For some proofs, it may be preferential to represent composite swaps in a non-inductive way, and instead think in terms of swapping the sets of the input and output vertices.

Lemma 25 (Alternate swap). For any $m, n \in \mathbb{N}$, a composite swap hypergraph $\sigma_{m,n}$ can be written in the form

$$\sigma_{m,n} = (\emptyset, \{S[\bullet]\}, \{T[\bullet]\}, \kappa, \emptyset)$$

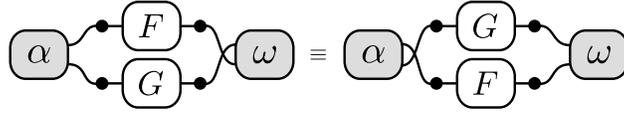
where $A, B, C, D \subset \mathbb{A}$ are disjoint sets such that $|A| = |D| = m$, $|B| = |C| = n$, $S[\bullet] = C + D$, $T[\bullet] = A + B$, and

$$\kappa(\pi_i(A)) = \pi_i(D) \quad \kappa(\pi_i(B)) = \pi_i(C)$$

Composite swap hypergraphs are *natural*: we can ‘push through’ hypergraphs composed on either side.

Proposition 26 (Naturality of swap). For $m, n, p, q \in \mathbb{N}$ and labelled interfaced linear hypergraphs $F : m \rightarrow n$ and $G : p \rightarrow q$,

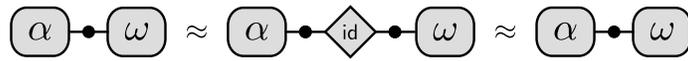
$$F \otimes G \cdot \sigma_{n,q} \equiv \sigma_{m,p} \cdot G \otimes F$$



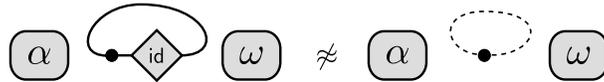
4.5 Homeomorphism

The operations so far have been fairly straightforward. However, a subtlety arises when we consider the trace. A naive approach to performing $\text{Tr}^x(F)$ would be to take the first x inputs and outputs and join them together. Now consider the trace of the identity: one might assume that $\text{Tr}^x(\text{id}_x) = \text{id}_0$ as it is simply a closed loop and does not ‘affect’ the term per se, but this is not always the case [23, §6.1]. So we cannot discard these loops, but we cannot represent closed loops in vanilla hypergraphs as vertices can only connect to edges.

This issue arises because we have ‘absorbed’ the identity morphisms, so to solve this problem we introduce the notion of *homeomorphism* to create *identity edges* $1 \rightarrow 1$, drawn as grey diamonds. We write the set of identity edges in a hypergraph as $E[\text{id}]$, and as such our hypergraph tuple becomes $H = (E, E[\text{id}], S, T, \kappa, \Lambda)$. The sources and targets of these identity edges must be preserved by homomorphism. In general, we can introduce or remove identity edges at will by performing an *expansion* or *smoothing* respectively.



The only exception is when the source and target vertex of the identity edge are connected. Performing a smoothing here would create an invalid hypergraph.



Definition 27 (Expansion). For a labelled interfaced linear hypergraph $F = (E_F, E_F[\text{id}], S_F, T_F, \kappa_F, \Lambda_F)$, and $s \in S_F, t \in T_F$ such that $\kappa_F(t) = s$, we can perform an *expansion* on (t, s) to yield hypergraph

$$G = (E_F, E_F[\text{id}] + \{e_{\text{id}}\}, S_F + S[e_{\text{id}}], T_F + T[e_{\text{id}}], \kappa_H, \Lambda_F)$$

with e_{id} fresh in \mathbb{A} , $S[e_{\text{id}}] = \{s'\}$, $T[e_{\text{id}}] = \{t'\}$, s', t' fresh in \mathbb{A} and

$$\kappa_H(v) = \begin{cases} s' & \text{if } v = t \\ s & \text{if } v = t' \\ \kappa_F(v) & \text{otherwise} \end{cases}$$

Proposition 28 (Well-formedness of expansion). *For any labelled interfaced linear hypergraph F containing target vertex v and source vertex s , where $\kappa(t) = s$, the result of performing an expansion on (t, s) is a well formed labelled interfaced linear hypergraph.*

Proof. All introduced vertices are fresh in \mathbb{A} , so the sources and targets are disjoint. One target vertex v connects to the fresh source vertex, and the fresh target vertex connects to its original connection $\kappa(v)$, so κ is bijective. The regular edges are unaffected, so the labelling condition is satisfied. \square

Definition 29 (Smoothing). *For a labelled interfaced linear hypergraph $F = (E_F, E_F[\text{id}] + \{e_{\text{id}}\}, S_F, T_F, \kappa_F, \Lambda_F)$, where $S[e_{\text{id}}] = \{s\}$ and $T[e_{\text{id}}] = \{t\}$, $\kappa_F(t) \neq s$, we can perform a smoothing on e_{id} to yield hypergraph*

$$G = (E_F, E_F[\text{id}], S_F - S_F[e_{\text{id}}], T_F - T_F[e_{\text{id}}], \kappa_H, \Lambda_F)$$

$$\kappa_H(v) = \begin{cases} \kappa_F(t) & \text{if } \kappa(v) = t \\ \kappa_F(v) & \text{otherwise} \end{cases}$$

Proposition 30 (Well-formedness of smoothing). *For any labelled interfaced linear hypergraph containing an identity edge e_{id} with source s and target t , $\kappa(t) \neq s$, the result of performing a smoothing on e_{id} is a well formed labelled interfaced linear hypergraph.*

Proof. We only remove vertices, so the sources and targets are disjoint. The change in the connections is that the target vertex that original connected to the source of the identity edge now redirects to the source vertex connected to by the target of the identity edge, so κ is bijective. The regular edges are unaffected, so the labelling condition is satisfied. \square

We call an interfaced linear hypergraph *minimal* if no smoothings can be performed. We quotient hypergraphs by homeomorphism and always draw the minimal version.

4.6 Trace

Now equipped with homeomorphism, we can define a suitable trace operation. To trace a hypergraph with one wire, we create a new identity edge with $\pi_0(S_F[\bullet])$ and $\pi_0(T_F[\bullet])$ as its source and target respectively.

$$\text{Tr}^1\left(\left(\alpha \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega\right)\right) = \alpha \bullet \left(\text{id} \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega \right)$$

The use of the identity edge ensures that we can represent the trace of the identity as a valid hypergraph.

$$\text{Tr}^1\left(\left(\alpha \bullet \omega\right)\right) = \alpha \bullet \left(\text{id} \bullet \omega \right)$$

Tracing multiple wires is performed inductively. The trace of no wires is equal to the original hypergraph.

$$\text{Tr}^1\left(\left(\alpha \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \phi \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega\right)\right) = \alpha \bullet \phi \bullet \omega$$

To trace multiple wires, we simply trace one at a time.

$$\text{Tr}^2\left(\left(\alpha \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \phi \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega\right)\right) = \text{Tr}^1\left(\left(\alpha \bullet \left(\text{id} \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \phi \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega \right)\right)\right) = \alpha \bullet \left(\text{id} \bullet \left(\text{id} \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \phi \bullet \left\langle \begin{array}{c} \bullet \\ \bullet \end{array} \right\rangle \bullet \omega \right) \right)$$

Definition 31 (Trace). For a labelled interfaced linear hypergraph

$$F : x + m \rightarrow x + n = (E_F, E_F[\text{id}], S_F, T_F, \kappa_F, \Lambda_F)$$

we can recursively define its trace of x wires $\text{Tr}^x(F) : m \rightarrow n$ as

$$\text{Tr}^0(F) = F$$

$$\text{Tr}^{x+1}(F) = \text{Tr}^1(\text{Tr}^x(F)) \text{ for } x > 0$$

with the base case defined as follows.

$$H = \text{Tr}^1(F) = (E_H, E_H[\text{id}], S_H, T_H, \kappa_H, \Lambda_H)$$

To perform a trace, we must introduce one identity edge to join the first input and output together, and ensure this does not create a closed loop of wires. Otherwise, the edges and labels remain the same.

$$E_H = E_F \quad E_H[\text{id}] = E_F[\text{id}] + \{e_{\text{id}}\} \quad e_{\text{id}} \text{ fresh in } \mathbb{A} \quad \Lambda_H = \Lambda_F$$

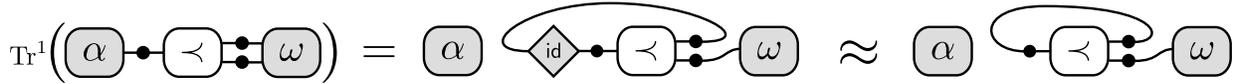
We take the first input and output vertex, and set them to be the target and source of the identity edge respectively.

$$\begin{aligned} S_H[\bullet] &= S_F[\bullet] - \pi_0(S_F[\bullet]) & S[e_{\text{id}}] &= \{\pi_0(S_F[\bullet])\} & S_H[e \in E_F] &= S_F[e] \\ T_H[\bullet] &= T_F[\bullet] - \pi_0(T_F[\bullet]) & T[e_{\text{id}}] &= \{\pi_0(T_F[\bullet])\} & T_H[e \in E_F] &= T_F[e] \end{aligned}$$

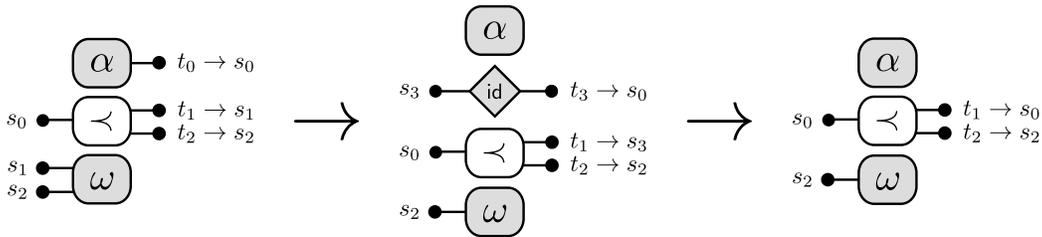
Since we have not deleted any vertices, the connections bijection remains the same.

$$\kappa_H = \kappa_F$$

After this operation, we can smooth the term as much as possible to remove any redundant identity edges, while still preserving the loops that do not connect to any regular edges.



The whole procedure can be drawn formally as follows:



Proposition 32 (Well-formedness of trace). For any labelled interfaced linear hypergraphs $F : x + m \rightarrow x + n$, $\text{Tr}^x(F)$ is a well-formed labelled interfaced linear hypergraph.

Proof. We have only moved a source and target from one edge to the identity edge, so the sets are still disjoint. The connections bijection is unchanged, so κ is bijective. The regular edges are unaffected, so the labelling condition is satisfied. \square

With trace defined, this means that we have all the operations of a STMC defined in terms of interfaced linear hypergraphs. Now we must show that hypergraphs equipped with these operations form a *sound and complete* graphical language.

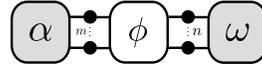
5 Soundness

We propose hypergraphs as a graphical language for STMCs. In particular we will focus on traced PROPs [34], categories with natural numbers as objects and addition as tensor product. First we consider *soundness*.

We fix a traced PROP \mathbf{Term}_Σ of morphisms freely generated over a signature Σ , and assemble labelled interfaced linear hypergraphs into the traced PROP $\mathbf{HypTerm}_\Sigma$, in which the morphisms $m \rightarrow n$ are hypergraphs of type $m \rightarrow n$, with composition, tensor, symmetry and trace defined as above.

Definition 33 (Interpretation functor). *We define the interpretation functor from terms to labelled interfaced linear hypergraphs as the identity-on-objects traced monoidal functor $\llbracket - \rrbracket_\Sigma : \mathbf{Term}_\Sigma \rightarrow \mathbf{HypTerm}_\Sigma$.*

We omit the subscript if unambiguous. $\llbracket - \rrbracket_\Sigma$ is defined recursively over the syntax of the term. For a generator $\phi : m \rightarrow n$, we interpret it as an edge with m sources and n targets.



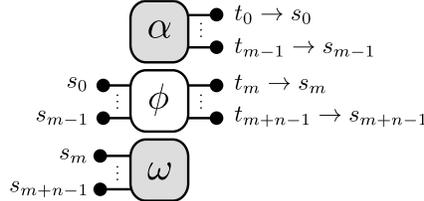
Formally, for a generator $\phi : m \rightarrow n$, this is defined as

$$\llbracket \phi \rrbracket_\Sigma = (\{e \text{ fresh in } \mathbb{A}\}, S, T, \kappa, \Lambda)$$

where

$$\begin{aligned} S[\bullet] &= \{s_i \text{ fresh in } \mathbb{A} \mid i < m\} & S[e] &= \{s_{i+m} \text{ fresh in } \mathbb{A} \mid i < n\} & \Lambda(e) &= \phi \\ T[\bullet] &= \{t_i \text{ fresh in } \mathbb{A} \mid i < m\} & T[e] &= \{s_{i+m} \text{ fresh in } \mathbb{A} \mid i < n\} & \kappa(t_i) &= s_i \end{aligned}$$

This can be drawn formally as follows.



Identity and symmetry morphisms translate into their hypergraph versions (Definitions 17 and 23)

$$\llbracket \text{id}_n \rrbracket = \text{id}_n \quad \llbracket \sigma_{m,n} \rrbracket = \sigma_{m,n}$$

To generate hypergraphs of larger terms, we can combine the morphism, identity and swap hypergraphs using composition and monoidal tensor, or by using the trace operator.

$$\llbracket f \cdot g \rrbracket = \llbracket f \rrbracket \cdot \llbracket g \rrbracket \quad \llbracket f \rrbracket \otimes \llbracket g \rrbracket \quad \llbracket \text{Tr}^x(f) \rrbracket = \text{Tr}^x(\llbracket f \rrbracket)$$

Proposition 34 (Well-formedness). *For any term f in a traced PROP \mathbf{Term}_Σ , $\llbracket f \rrbracket_\Sigma$ is a well-formed labelled interfaced linear hypergraph.*

Proof. Generator, identity and swap hypergraphs are well-formed, and all other operations involved create well-formed interfaced linear hypergraphs (Propositions 16, 19, 32). \square

To show soundness, we must examine that the axioms of STMCs are satisfied in the language of labelled interfaced linear hypergraphs. as illustrated in Figure 1.

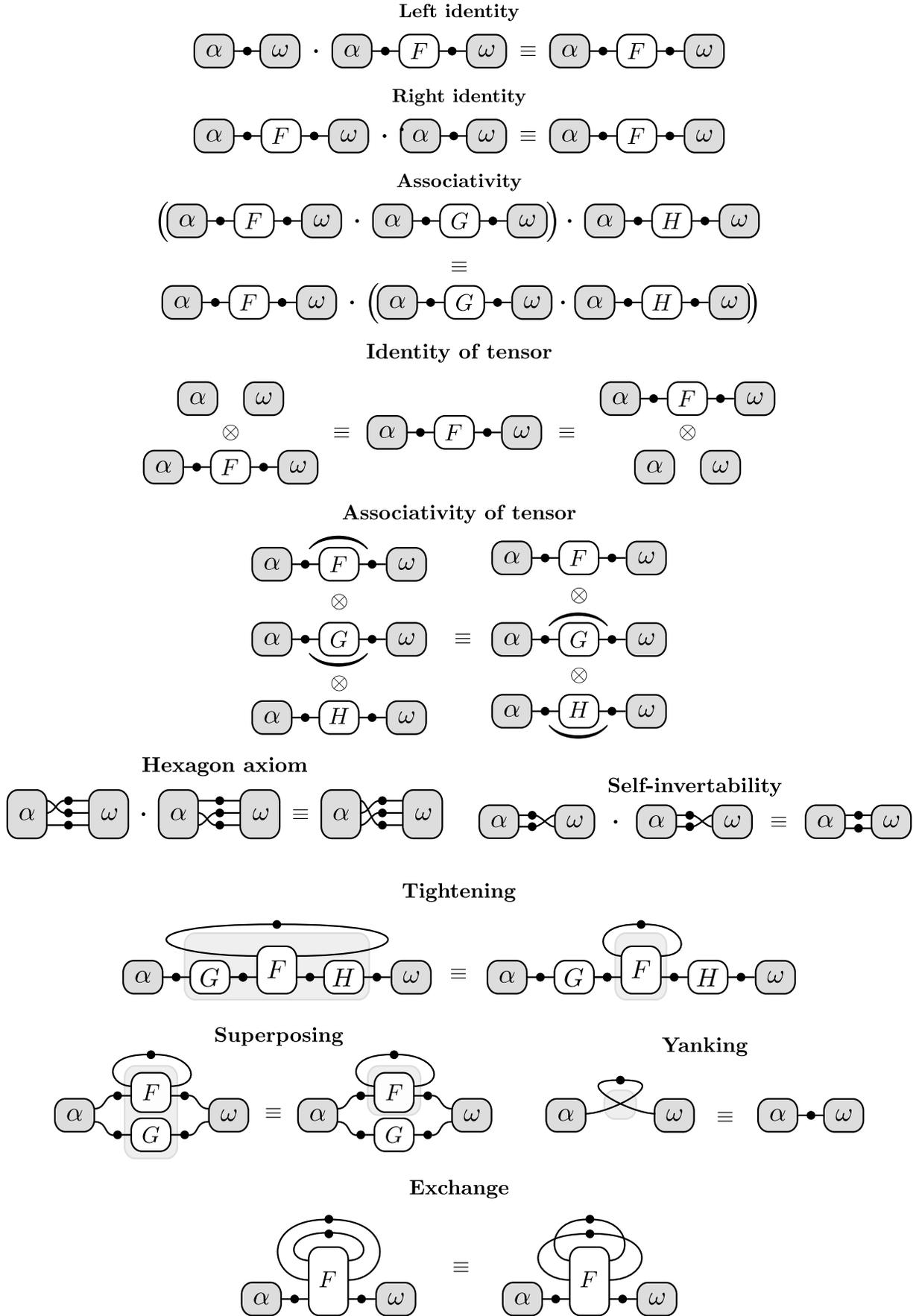


Figure 1: The axioms of STMCs, represented using labelled interfaced linear hypergraphs.

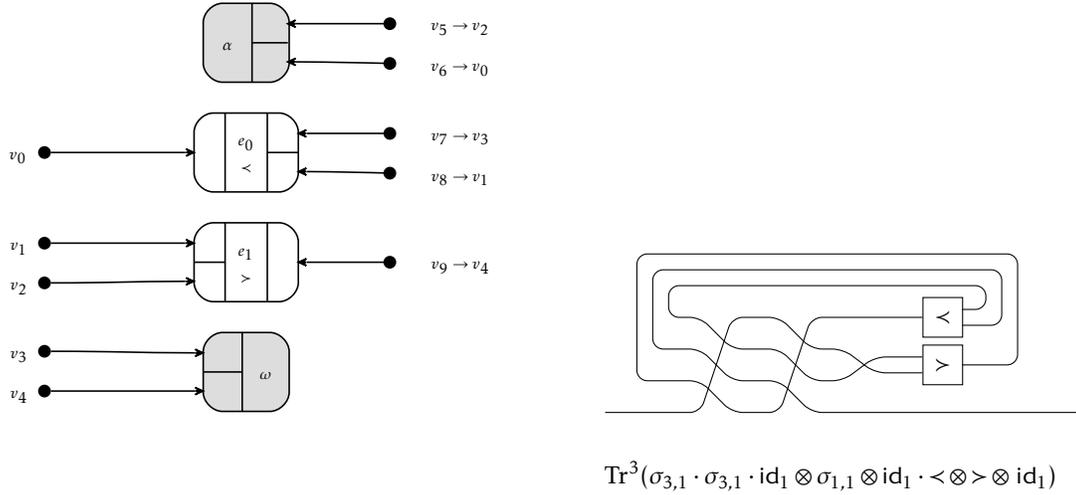


Figure 2: A hypergraph and its corresponding categorical term.

Theorem 35 (Soundness). *For any morphisms $f, g \in \mathbf{Term}_\Sigma$, if $f = g$ under the equational theory of the category, then their interpretations as labelled interfaced linear hypergraphs are isomorphic $\llbracket f \rrbracket \equiv \llbracket g \rrbracket$.*

Proof. Composition produces well-formed interfaced linear hypergraphs (Proposition 16) and satisfies the axioms of categories with the identity hypergraph $\text{id}_n : n \rightarrow n$ as the unit of composition for n . Monoidal tensor produces well-formed interfaced linear hypergraphs (Proposition 19), is a bifunctor (Propositions 21 and 22) and satisfies the axioms of (strict) monoidal categories with the empty hypergraph $0 : 0 \rightarrow 0$ as the monoidal unit. The swap hypergraph is natural (Proposition 26) and satisfies the axioms of symmetric monoidal categories. The trace operator produces well-formed interfaced linear hypergraphs (Proposition 32) and satisfies the axioms of symmetric traced monoidal categories specified in Section 2. \square

Remark 36. *One may wonder if the axioms also hold arbitrary STMCs where the objects are not just natural numbers. The answer is yes – the generalisation can be found in Section 9.*

6 Completeness

We are also able to recover categorical terms in an STMC from labelled interfaced linear hypergraphs. This is a two stage process: first we show that any well-formed labelled interfaced linear hypergraph has at least one corresponding categorical term (*definability*); then we show that all of these terms are equal in the category (*coherence*).

6.1 Definability

The strategy to retrieve a categorical term from a hypergraph is to exploit the formal graphical representation, in which all edges are ‘stacked’. From this representation we can read off a tensor of generators, then connect wires of opposite polarities by linking them with trace and symmetries. An example is shown in Figure 2.

Definition 37 (Definability). *Labelled interfaced linear hypergraphs are definable if for every $F \in \mathbf{LHyp}_\Sigma^\bullet$, we can retrieve a well-formed categorical equation for which the hypergraph interpretation of that term is equivalent to the original graph, i.e. for a candidate $\langle\langle - \rangle\rangle : \mathbf{HypTerm}_\Sigma \rightarrow \mathbf{Term}_\Sigma$, then $\langle\langle F \rangle\rangle \equiv F$.*

The first step is to fix a total order on the edges e_1, \dots, e_n , including any identity edges. We fix this order \leq globally. The stack operation creates a tensor of the corresponding generators in \mathbf{Term}_Σ for each edge

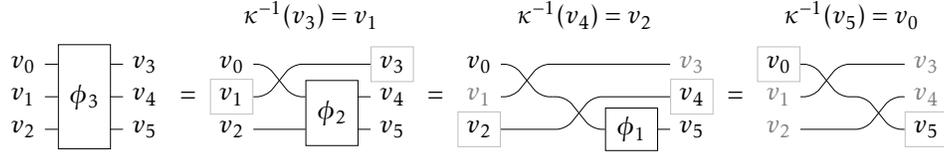


Figure 3: Performing the shuffle algorithm, where ϕ_n denotes the shuffle construct for n vertices.

in the hypergraph. Identity edges are represented by identity morphisms id_1 in the stack.

$$\text{stack}(-)_{\Sigma, \leq} : \mathbf{HypTerm}_{\Sigma} \rightarrow \mathbf{Term}_{\Sigma}$$

$$\text{stack}(F)_{\Sigma, \leq} = \bigotimes_{e \in (E_F + E_F[\text{id}, \leq])} \phi \text{ where } \phi = \begin{cases} \Lambda(e) & \text{if } e \in E_F \\ \text{id}_1 & \text{if } e \in E_F[\text{id}] \end{cases}$$

Most of the outputs from our stack of generators will need to connect to the inputs of other generators in the stack, so we must trace them around. Then the only remaining step is to then connect the traced wires to the corresponding inputs in the stack. Here it will be useful to consider the all the target and source vertices as two totally ordered sets, respecting our new edge order \leq . We write $S[\leq]$, $T[\leq]$ for the ordered set of all vertices which respects the original order on the vertices and the new order on the edges. Since the interface is not contained within the order, we set input vertices to be the lowest elements of $T[\leq]$ and the output vertices to be the greatest elements of $S[\leq]$. For example, if we have sets $S[e_1] = \{s_1, s_2\}$, $S[e_2] = \{s_3, s_4, s_5\}$ and $S[\bullet] = \{s_6\}$, then if we define \leq as $e_1 < e_2$, then $S[\leq] = \{s_1, s_2, s_3, s_4, s_5, s_6\}$. To simplify notation, we also introduce the notion of a *connections permutation*.

Definition 38 (Connections permutation). *For an interfaced linear hypergraph H equipped with edge order \leq , where $|S[\leq]| = x$, we call its connections permutation $p : [x] \rightarrow [x]$ the permutation such that every $i < x$, $\kappa_H(\pi_i(T[\leq])) = \pi_{p(i)}(S[\leq])$.*

Lemma 39 (Discrete composition). *For any two discrete interfaced linear hypergraphs $F : m \rightarrow n$ and $G : n \rightarrow k$, with connections permutations p and q respectively, then the connections permutation of $F \cdot G$ is $q \circ p$.*

Proof. By definition of connections permutations, for each target $t = \pi_i(T_F[\bullet])$, $\kappa_F(t) = \pi_{p(i)}(S_F[\bullet])$, and for each target $t = \pi_i(T_G[\bullet])$, $\kappa_G(t) = \pi_{q(i)}(S_G[\bullet])$. Since composition deletes the input vertices of G , we are only concerned with the input vertices of F . By the first connections permutation, in F the i th input vertex originally connected to the $p(i)$ th output vertex, so by definition of composition, in $F \cdot G$ it will be connected to $\kappa_G(\pi_{p(i)}(T_G[\bullet]))$. By the second connections permutation this is equal to $\pi_{q(p(i))}(S_G[\bullet])$. Therefore the connections permutation of $F \cdot G$ is $q \circ p$. \square

We use this permutation to define a ‘shuffle’ construct comprised of symmetries and identities, defined recursively over the set $S[\leq]$. The target that connects to the lowest source is determined, and a symmetry pulling this wire up to the ‘top’ is then defined: this wire is now in the correct position and is of no further concern to us. We recursively perform shuffle on the remaining source and target vertices until none remain, as demonstrated in Figure 3. Before proceeding, we show that the ‘input-output’ connectivity of the shuffle construct reflects the connectivity of the original hypergraph.

Lemma 40 (Correctness of shuffle). *For any interfaced linear hypergraph F with connections permutation p and some $\bar{v} : 0 \rightarrow n \in \mathbf{Term}_{\Sigma} = v_0 \otimes v_1 \otimes \dots \otimes v_{n-1}$, $\bar{v} \cdot \text{shuffle}(F)_{\Sigma, \leq} = v_{p^{-1}(0)} \otimes v_{p^{-1}(1)} \otimes \dots \otimes v_{p^{-1}(n-1)}$.*

Proof. This is by induction on n . For $n < 2$ the statement is trivially correct. For $n = 2$, there are two cases: $\{0 \mapsto 0, 1 \mapsto 1\}$ and $\{0 \mapsto 1, 1 \mapsto 0\}$. By naturality of symmetry, $\bar{v} \cdot \sigma_2$ is equal to $v_0 \otimes (v_1 \cdot \phi_1) = v_0 \otimes v_1$ in the former and $v_1 \otimes (v_0 \cdot \sigma_1) = v_1 \otimes v_0$, so for both cases the statement holds. For $n > 2$, $\bar{v} \cdot \phi_n = v_x \otimes (v_0 \otimes \dots \otimes v_{x-1} \otimes v_{x+1} \otimes \dots \otimes v_{n-1} \cdot \sigma_{n-1})$ where $x = p^{-1}(0)$ by naturality of symmetry. Therefore the first element of the tensor is correct, and the remaining elements follow by inductive hypothesis. \square

```

Function shuffle'_{\Sigma, \leq}(S, T, \kappa)
  if |S| = 0 then
    | return id_0;
  end
  s \leftarrow \pi_0(S); t \leftarrow \kappa^{-1}(s);
  i \leftarrow i \text{ where } \pi_i(T) = t; j \leftarrow |T| - i - 1;
  f \leftarrow \sigma_{i,1} \otimes j;
  return f \cdot id_1 \otimes \text{shuffle}_{\Sigma, \leq}(S - s, T - t, \kappa);
end
Function shuffle_{\Sigma, \leq}(F = (E, S, T, \kappa, \Lambda))
  | shuffle'_{\Sigma, \leq}(S[\leq], T[\leq], \kappa);
end

```

Algorithm 1: Defining the shuffle construct.

Since there is no input ‘box’, we also need to precompose the shuffle construct with another symmetry to pull the input wires to the ‘top’ of the term. To retrieve a term from a hypergraph H with edge order \leq , we simply trace the composition of the corresponding shuffle construct and edge stack.

Definition 41 (Definability functor). *We define the definability functor as the identity-on-objects traced monoidal functor $\llbracket - \rrbracket_{\Sigma, \leq} : \mathbf{HypTerm}_{\Sigma} \rightarrow \mathbf{Term}_{\Sigma}$ with its action defined for a given edge order \leq and interfaced linear hypergraph $F : m \rightarrow n$ with $|S[\leq]|$ as*

$$\llbracket F \rrbracket_{\Sigma, \leq} = \text{Tr}^{x-m}(\sigma_{x-m, m} \cdot \text{shuffle}(F)_{\Sigma, \leq} \cdot \text{stack}(F)_{\Sigma, \leq} \otimes \text{id}_n)$$

To conclude definability we must be able to return to the original hypergraph. The shuffle construct is our main obstacle to showing this, so we tackle it separately.

Lemma 42 (Definability of shuffle). *For any shuffle construct $\phi_n : n \rightarrow n$, interpretation $F = \llbracket \phi_n \rrbracket$, and permutation $p : [n] \rightarrow [n]$ such that for some $\vec{v} : 0 \rightarrow n = v_0 \otimes v_1 \otimes \dots \otimes v_{n-1}$, $\vec{v} \cdot \phi = v_{p(0)} \otimes v_{p(1)} \otimes \dots \otimes v_{p(n-1)}$, then $\kappa(\pi_i(T_F[\bullet])) = \pi_{p(i)}(S_F[\bullet])$.*

Proof. We first use staging and composite symmetry to arrange the shuffle construct into ‘slices’ containing exactly one symmetry $\sigma_{1,1}$. We then perform induction on n . For $n < 2$ the statement holds trivially. For $n = 2$ we examine the two cases $\{0 \mapsto 0, 1 \mapsto 1\}$ and $\{0 \mapsto 1, 1 \mapsto 0\}$ as in correctness of shuffle. They correspond to the interfaced linear hypergraphs id_2 and $\sigma_{1,1}$ respectively. For both cases the statement holds. For $n > 2$, we split the definition of σ_n into two parts: $\sigma'_n = \sigma_{x,1} \otimes \text{id}_{n-1-x}$ where $x = p^{-1}(0)$, and $\sigma''_n = \text{id}_1 \otimes \sigma_{n-1}$. In $\llbracket \sigma'_n \rrbracket$, $\kappa(\pi_x(T[\bullet])) = \pi_0(S[\bullet])$ by definition of the swap hypergraph. In $\llbracket \sigma''_n \rrbracket$, $\kappa(\pi_0(T[\bullet])) = \pi_0(S[\bullet])$ by definition of identity and monoidal tensor. Therefore in $\llbracket \sigma_n \rrbracket$, $\kappa(\pi_i(T)) = \pi_0(S)$ by discrete composition. So for the first vertex in T the statement holds. For the remaining vertices we apply the inductive hypothesis to ϕ_{n-1} and add one to the indices of each vertex, since we tensor the construct with an identity wire. \square

Finally we can take on the entire term.

Proposition 43 (Definability). *For any interfaced linear hypergraph $F : m \rightarrow n$ equipped with edge order \leq , where $G = \llbracket \llbracket F \rrbracket_{\Sigma, \leq} \rrbracket$, then $F \equiv G$.*

Proof. We map the sources and targets of edges in F to the corresponding vertices in $\llbracket \llbracket F \rrbracket_{\Sigma, \leq} \rrbracket$. We do the same for the edges. If there is an id_1 in the stack of generators, we perform an expansion to introduce an identity edge. The labelling condition is immediate, so we only need to consider the connections condition. Using the edge order \leq , we consider the connections permutation of F and G , which we name p and q respectively. If $F \equiv G$, then the two permutations must be the same.

We examine the permutation q . There are two classes of vertices to consider: those that are inputs and those that are not. In the first case, the i th input (corresponding to $\pi_i(T[\leq])$) will connect to the $p(i)$ th source vertex by definability of shuffle, so $q = p$. In the second case, the i th target (which corresponds to $\pi_{i+m}(T[\leq])$), will connect to the $p(i+m)$ th source vertex by definition of trace, discrete composition and definability of shuffle. So $q = p$ in this case also. Therefore the connection permutations are equal, so $F \equiv G$. \square

6.2 Coherence

We cannot immediately conclude completeness. There are multiple orders we can choose when stacking the edges, resulting in multiple different shuffle constructs and recovered terms. For *coherence* these recovered terms must all be equal by the equations of the STMC.

Definition 44 (Coherence). *Interfaced linear hypergraphs are coherent if, for any well-formed interfaced linear hypergraph F and any two orders on its edges \leq_1, \leq_2 , $\langle\langle F \rangle\rangle_{\Sigma, \leq_1} = \langle\langle F \rangle\rangle_{\Sigma, \leq_2}$ by the equations of the STMC.*

Fortunately, we only need to consider switching two consecutive edges while retaining the others, e.g. $x < f < g < y$ becomes $x < g < f < y$. We can then swap any two edges in the set by propagating these swaps throughout the entire set. To enable us to do this, we can combine a tensor of edge boxes into one single box:

Lemma 45 (Combination). *For any $t = \bigotimes_{i=0}^m a_i \otimes \bigotimes_{i=0}^n b_i \otimes \bigotimes_{i=0}^p c_i$, we can rewrite it as $a \otimes \bigotimes_{i=0}^n b_i \otimes c$ where $a : \Sigma_{i=0}^m \text{dom}(a_i) \rightarrow \Sigma_{i=0}^m \text{cod}(a_i)$ and $c : \Sigma_{i=0}^p \text{dom}(c_i) \rightarrow \Sigma_{i=0}^p \text{cod}(c_i)$.*

We need a lemma to show that we can transform the shuffle construct for a given order into one for a different order by adding appropriate symmetries on either side, reflecting that the edge boxes have now swapped over.

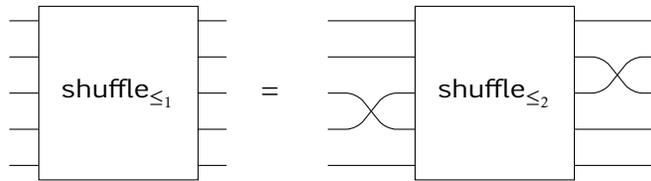
Lemma 46 (Coherence of shuffle). *For any interfaced linear hypergraph $F : m \rightarrow n$, two orders \leq_1, \leq_2 on its edges in which only two consecutive elements $e_1 : n' \rightarrow n$ and $e_2 : p' \rightarrow p$ have been swapped, and shuffle constructs*

$$\text{shuffle}(F)_{\Sigma, \leq_1} : m + p + q + r + s \rightarrow p' + q' + r' + s' + n$$

$$\text{shuffle}(F)_{\Sigma, \leq_2} : m + p + r + q + s \rightarrow p' + r' + q' + s' + n$$

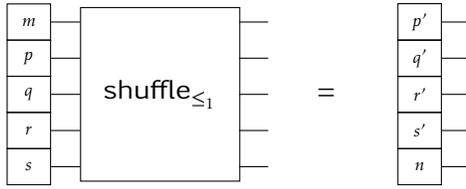
then

$$\text{shuffle}(F)_{\Sigma, \leq_1} = \text{id}_m \otimes \text{id}_p \otimes \sigma_{q,r} \otimes \text{id}_s \cdot \text{shuffle}(F)_{\Sigma, \leq_2} \cdot \text{id}_{p'} \otimes \sigma_{r',q'} \otimes \text{id}_{s'} \otimes \text{id}_n$$

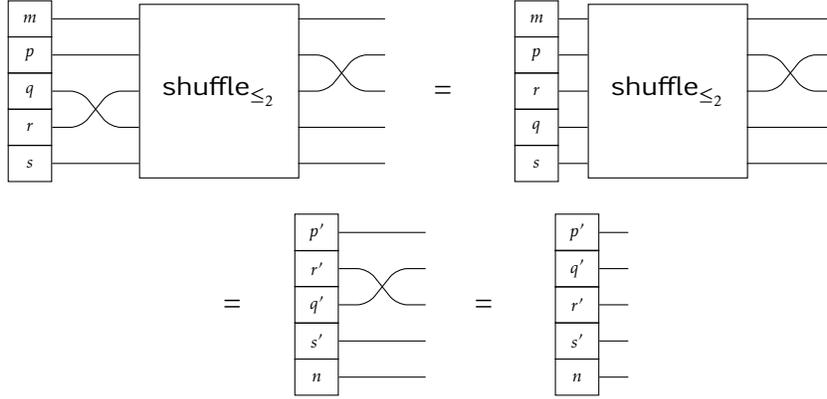


Proof. We show equality by asserting for every ‘input’ to the term, it always leads to the same ‘output’. In the diagram below, each box represents a ‘bundle’ of wires. By correctness of shuffle (Lemma 40), we know that the each shuffle construct respects the connections permutation of the H with order \leq_1 and \leq_2 respectively. Effectively, this means that it will shuffle the vertices into new bundles for each generator in the stack, only differing by the swapped edges (labelled r' and q' on the diagram below). We can therefore use naturality of symmetry to show both expressions are equal.

LHS



RHS



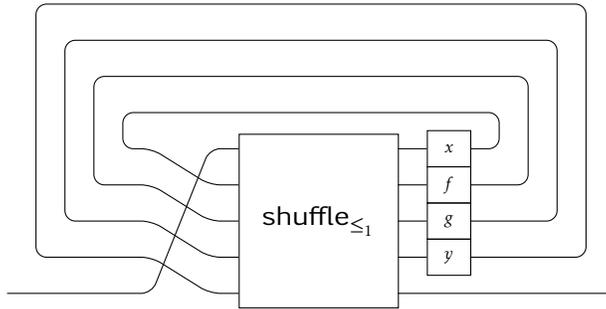
□

Lemma 47 (Coherence for two edges). *For any interfaced linear hypergraph $F : m \rightarrow n$ and two orders \leq_1, \leq_2 on its edges which differ only by the swapping of two consecutive elements, $\langle\langle H \rangle\rangle_{\Sigma, \leq_1} = \langle\langle H \rangle\rangle_{\Sigma, \leq_2}$.*

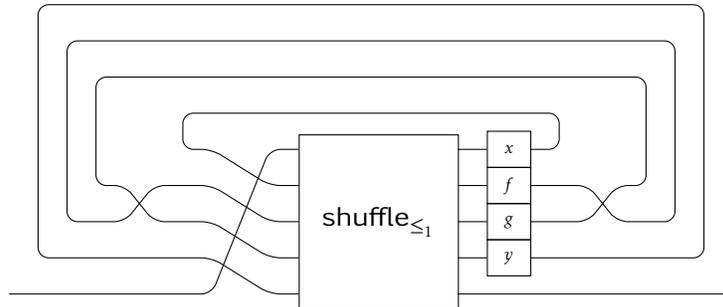
Proof. Any term generated by $\langle\langle F \rangle\rangle_{\Sigma, \leq}$ can be rewritten in the form

$$\text{Tr}^{|T|-m}(\sigma_{|T|-m, m} \cdot \text{shuffle}(F)_{\Sigma, \leq} \cdot x \otimes f \otimes g \otimes y \otimes \text{id}_n)$$

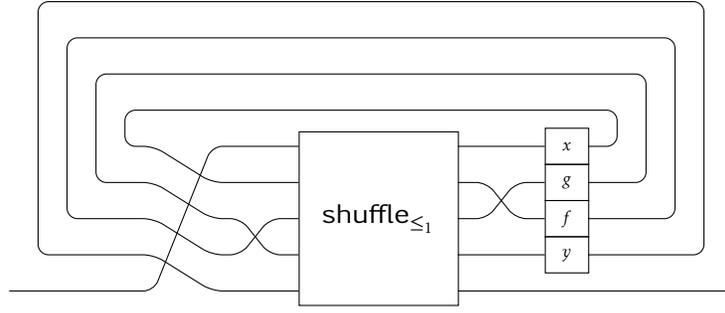
by using combination (Lemma 45). So we have a term of the form:



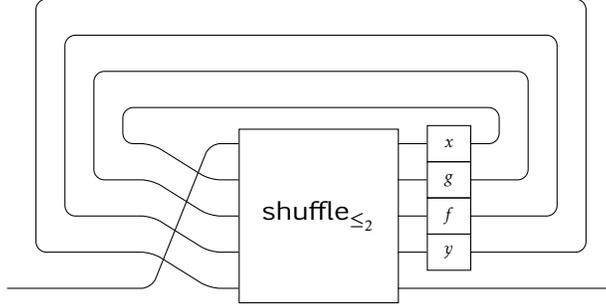
By exchange:



By naturality of symmetry and functoriality:



By coherence of shuffle (Lemma 46):



□

We can then extend this lemma to obtain our final coherence result.

Proposition 48 (Coherence). *For all orderings of edges \leq_x on an interfaced linear hypergraph F ,*

$$\langle\langle F \rangle\rangle_{\Sigma, \leq_1} = \langle\langle F \rangle\rangle_{\Sigma, \leq_2} = \cdots = \langle\langle F \rangle\rangle_{\Sigma, \leq_x}$$

Proof. By repeatedly applying Lemma 47 until the desired order is obtained. □

Since the edge order chosen is irrelevant, we are justified in dropping all subscripts from $\langle\langle - \rangle\rangle$ and not referring to ‘the chosen’ order. We can now conclude completeness.

Theorem 49 (Completeness I). *For any interfaced linear hypergraph $F \in \mathbf{LHyp}_{\Sigma}^{\bullet}$ there exists a unique morphism $f \in \mathbf{Term}_{\Sigma}$, up to the equations of the STMC, such that $\llbracket f \rrbracket = H$.*

Proof. By definability (Proposition 43) and coherence (Proposition 48). □

We can also operate in the opposite direction and return to the original term after translating it into a hypergraph. We first state a property of the definability functor.

Lemma 50 (Compositionality of definability). *For any $F, G \in \mathbf{LHyp}_{\Sigma}^{\bullet}$ and $x \in \mathbb{N}$,*

$$\langle\langle F \cdot G \rangle\rangle = \langle\langle F \rangle\rangle \cdot \langle\langle G \rangle\rangle \quad \langle\langle F \otimes G \rangle\rangle = \langle\langle F \rangle\rangle \otimes \langle\langle G \rangle\rangle \quad \langle\langle \text{Tr}^x(F) \rangle\rangle = \text{Tr}^x(\langle\langle F \rangle\rangle)$$

Proof. By sliding and yanking. □

Theorem 51 (Completeness II). *For any morphism $f \in \mathbf{Term}_{\Sigma}$, $\langle\langle \llbracket f \rrbracket \rangle\rangle = f$.*

Proof. As f is freely generated, $f = \text{Tr}^x(f_0 \cdot f_1 \cdots f_{n-1})$ by staging and global trace, where each $f_i = \text{id}_p \otimes k \otimes \text{id}_q$. Therefore $\llbracket f \rrbracket = \text{Tr}^x(\llbracket f_0 \rrbracket \cdot \llbracket f_1 \rrbracket \cdots \llbracket f_{n-1} \rrbracket)$ by definition of the definability functor, and $\langle\langle \llbracket f \rrbracket \rangle\rangle = \text{Tr}^x(\langle\langle f_0 \rrbracket \rangle\rangle \cdot \langle\langle f_1 \rrbracket \rangle\rangle \cdots \langle\langle f_{n-1} \rrbracket \rangle\rangle)$ by Lemma 50. By sliding and yanking, each $\langle\langle \llbracket f_i \rrbracket \rangle\rangle = f_i$, and by composition, $\langle\langle \llbracket f \rrbracket \rangle\rangle = f$. □

7 Graph rewriting

For standard STMCs, reasoning diagrammatically using isomorphism of diagrams works well, as the axioms are absorbed into the graphical notation. However, we often wish to add extra structure to our categories, with associated axioms. To solve this problem in the language of terms we use *term rewriting*.

Definition 52 (Subterm). *For any morphisms $f, g \in \mathbf{Term}_\Sigma$, we say that g is a subterm of f if there exists $\hat{f}_1, \hat{f}_2 \in \mathbf{Term}_\Sigma$ and $n, x \in \mathbb{N}$ such that $f = \text{Tr}^x(\hat{f}_1 \cdot \text{id}_n \otimes g \cdot \hat{f}_2)$, where \hat{f}_1, \hat{f}_2 do not contain any traces.*

Definition 53 (Term rewriting). *A rewrite rule in \mathbf{Term}_Σ is a pair $\langle l, r \rangle$ where $l, r : X \rightarrow Y$ are terms in \mathbf{Term}_Σ with the same domain and codomain. We write rules as $\langle l, r \rangle : X \rightarrow Y$. A rewriting system \mathcal{E} is a set of rewrite rules. We can perform a rewrite step in \mathcal{E} for two terms g and h (written $g \Rightarrow_{\mathcal{E}} h$) if there exists rewriting rule $\langle l, r \rangle \in \mathcal{E}$ such that l is a subterm of g , i.e. we can write g in the form $\text{Tr}^x(\hat{f}_1 \cdot \text{id} \otimes l \cdot \hat{f}_2)$ for some trace-free terms \hat{f}_1 and \hat{f}_2 .*

As is often the case in the algebraic realm, it can be difficult to identify the occurrence of l in f due to functoriality. However, unlike with the axioms of STMCs, these extra axioms are not an intrinsic part of our diagrams. We can no longer rely purely on hypergraph isomorphisms. Fortunately, we can generalise term rewriting to *graph rewriting*. First we must formalise the notion of a subgraph of an interfaced linear hypergraph.

Definition 54 (Subgraph). *For any linear hypergraphs $F, G \in \mathbf{LHyp}_\Sigma^\bullet$, we say that G is a subgraph of F if there exists an embedding monomorphism $G \rightarrow F$.*

Intuitively, to transform some subgraph L into a larger graph G , we apply a sequence of operations.

Lemma 55. *For any minimal $F, G \in \mathbf{LHyp}_\Sigma^\bullet$ and $x \in \mathbb{N}$, there exist monomorphisms $F \rightarrow F \cdot G$, $F \rightarrow G \cdot F$, $F \rightarrow F \otimes G$, $F \rightarrow G \otimes F$ and $F \rightarrow \text{Tr}^x(F)$.*

Proof. This is immediate, as we simply embed the operand into the result. \square

Remark 56. *Note that the use of homeomorphism in the definition of trace guarantees that we can define a monomorphism $F \rightarrow \text{Tr}^x(F)$. Had we used a naive definition that simply coalesced the input and output vertices, the vertex map would not be injective.*

Lemma 57. *For any monomorphism $m : F \rightarrow G \in \mathbf{LHyp}_\Sigma^\bullet$, there exist interfaced linear hypergraphs F_1, F_2 and $n, x \in \mathbb{N}$ such that $m = \text{Tr}^x(-) \circ (F_2 \cdot -) \circ (- \cdot F_1) \circ (\text{id}_n \otimes -)$.*

Proof. Since composition, tensor and trace can all be represented as monomorphisms (Lemma 55), we apply the sequence of operations that transforms F into G . \square

Lemma 58. *For any morphisms $f, g \in \mathbf{Term}_\Sigma$, g is a subterm of f if and only if $\llbracket g \rrbracket$ is a subgraph of $\llbracket f \rrbracket$.*

Proof. For (\Rightarrow) we assume that g is a subterm of f . Therefore we know that we can write f in the form of Definition 52. By definition of $\llbracket - \rrbracket$, $\llbracket f \rrbracket = \text{Tr}^x(\llbracket \hat{f}_1 \rrbracket \cdot \llbracket \text{id}_n \rrbracket \otimes \llbracket g \rrbracket \cdot \llbracket \hat{f}_2 \rrbracket)$. As we can express all operations as monomorphisms (Lemma 55), there exists a monomorphism $\llbracket g \rrbracket \rightarrow \llbracket f \rrbracket$, namely $\text{Tr}^x(-) \circ (\llbracket \hat{f}_1 \rrbracket \cdot -) \circ (- \cdot \llbracket \hat{f}_2 \rrbracket) \circ (\llbracket \text{id}_n \rrbracket \otimes -)$.

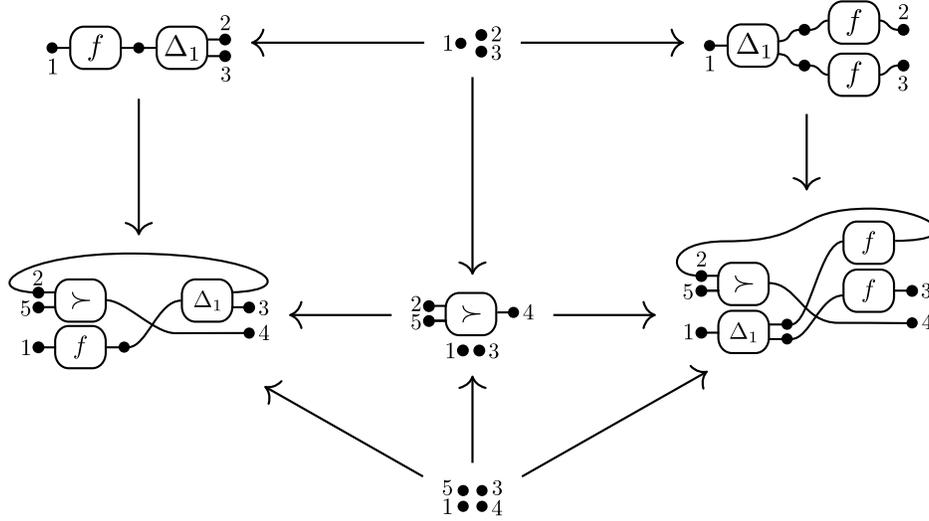


Figure 4: An example of DPO rewriting in \mathbf{Hyp}_Σ , using the axiom of naturality in a Cartesian category.

For (\Leftarrow) we assume that there exists an embedding monomorphism $m : \llbracket g \rrbracket \rightarrow \llbracket f \rrbracket$. Therefore by Lemma 57, there exists hypergraphs F_1, F_2 and $n, x \in \mathbb{N}$ such that $m = \text{Tr}^x(-) \circ (F_2 \cdot -) \circ (- \cdot F_1) \circ (\text{id}_n \otimes -)$. This means that $\llbracket f \rrbracket = \text{Tr}^x(F_2 \cdot \text{id}_n \otimes \llbracket g \rrbracket \cdot F_1)$. We apply the definability functor to both sides:

$$\begin{aligned}
 \langle\langle \llbracket f \rrbracket \rangle\rangle &= f = \langle\langle \text{Tr}^x(F_2 \cdot \text{id}_n \otimes \llbracket g \rrbracket \cdot F_1) \rangle\rangle && \text{Completeness II} \\
 &= \text{Tr}^x(\langle\langle F_2 \rangle\rangle \cdot \text{id}_n \otimes \langle\langle \llbracket g \rrbracket \rangle\rangle \cdot \langle\langle F_1 \rangle\rangle) && \text{Compositionality of definability} \\
 &= \text{Tr}^x(\langle\langle F_2 \rangle\rangle \cdot \text{id}_n \otimes g \cdot \langle\langle F_1 \rangle\rangle) && \text{Completeness II}
 \end{aligned}$$

Therefore there exist $f_1 = \langle\langle F_1 \rangle\rangle$ and $f_2 = \langle\langle F_2 \rangle\rangle$ such that $f = \text{Tr}^x(f_1 \cdot \text{id}_n \otimes g \cdot f_2)$, so g is a subterm of f . \square

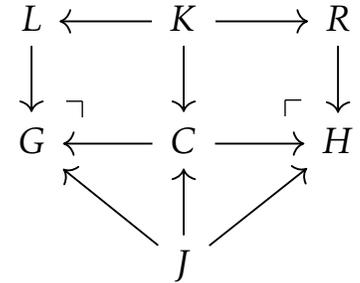
7.1 DPO rewriting

A popular approach to graph rewriting is known as double pushout (DPO) rewriting [15], and we use an extension of the traditional definition that introduces an ‘interface’ [6]. We start by recalling the definition for ordinary hypergraphs in \mathbf{Hyp}_Σ .

Definition 59 (DPO). A DPO rewrite rule $\langle L, R \rangle \in \mathbf{Hyp}_\Sigma$ is a span $L \leftarrow K \rightarrow R$, where K is discrete. A DPO system \mathcal{R} is a set of DPO rules. We write $G \rightsquigarrow_{\mathcal{R}} H$ if there exists span $L \leftarrow K \rightarrow R$ in \mathcal{R} , discrete hypergraph J , and cospan $K \rightarrow C \leftarrow J$, such that the diagram below commutes, and the two squares are pushouts.

To perform rewrite rule $\langle L, R \rangle$ in some larger graph G , a *matching* monomorphism $L \rightarrow G$ must first be identified. Then its *pushout complement* $K \rightarrow C \rightarrow G$ can be computed: C is effectively ‘ G with L removed’. Finally we compute the pushout $C \rightarrow H \leftarrow R$, yielding the graph H : the graph G with L replaced by R .

To perform rewrite rule $\langle L, R \rangle$ in some larger graph G , a *matching* $p : L \rightarrow G$ must first be identified. Its *pushout complement* $K \rightarrow C \rightarrow G$ and the pushout $C \rightarrow H \leftarrow R$ can then be computed, yielding us the graph H : the graph G with L replaced by R . An example of the procedure can be seen in Figure 4.



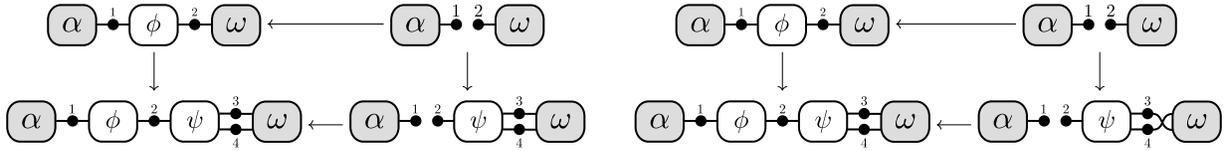
7.2 Adhesive categories

Not all structures are compatible with DPO rewriting. For example, the pushout complements may not be unique. This is essential, as it implies that for a given matching monomorphism, there is a unique rewrite of the graph. A commonly used framework that ensures the DPO procedure is always well-defined is that of *adhesive categories*, introduced by Lack and Sobociński [35]. The key property of these categories is that pushout complements are always unique for rewrite rules where each leg of the span is a monomorphism, if such a complement exists. Additionally, they also enjoy a local Church-Rosser theorem and a concurrency theorem. We have already met an adhesive category:

Proposition 60. \mathbf{Hyp}_Σ is adhesive.

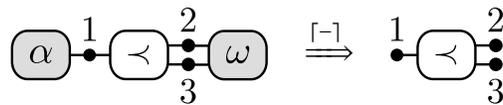
Proof. \mathbf{Hyp}_Σ is a slice category of a presheaf category, so is adhesive [35]. \square

We need to build on this to reach our *interfaced linear hypergraphs*. Unfortunately our category of interfaced linear hypergraphs $\mathbf{LHyp}_\Sigma^\bullet$ is not adhesive: pushout complements are not unique. This is because we do not require that the interface orders are preserved by homomorphism, as illustrated below.



The left case would be the ‘natural’ choice: we are only concerned with the section of graph being rewritten, so everything else should be left untouched. A possible solution would be to enforce that interfaces *are* preserved by homomorphism. This raises its own problems, as usually this interface flexibility is advantageous: for example, it allows us to model the operations of our hypergraphs (Lemma 55). A simpler option is to simply translate our interfaced linear hypergraphs into \mathbf{Hyp}_Σ and perform rewriting there. Since we are using DPO *with interfaces*, we know that the interfaces are preserved throughout the rewriting procedure, and thus we can recover it at the end.

The first thing we must do is to remove the interfaces of our interfaced linear hypergraphs, so that we have a regular (uninterfaced) linear hypergraph. We call this procedure *trimming*.



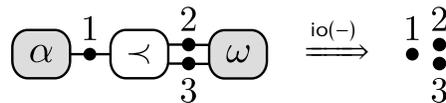
Definition 61 (Trimming). For any interfaced linear hypergraph F , we can trim its interfaces with the functor $[-] : \mathbf{LHyp}_\Sigma^\bullet \rightarrow \mathbf{Hyp}_\Sigma$, defined as $[F] = (V, E_F, \lambda_e.S_F[e], \lambda_e.\kappa_F^* \circ T_F[e])$, where $V = \bigcup_{e \in E} S_F[e]$.

We also write $[v]$ for the image of a vertex v and $[V]$ for the image of a set of vertices V under this functor.

Lemma 62. For any $F \in \mathbf{LHyp}_\Sigma^\bullet$, $[F]$ is linear.

Proof. Since each $S_F[e]$ is disjoint, each vertex is only in the sources of one edge. Since each $T_F[e]$ is disjoint and κ_F is bijective, each vertex is only in the targets of one edge. Therefore $[F]$ is linear. \square

Of course, we cannot simply ‘forget’ the interfaces – we will need them to recover the orders after we have performed rewriting. We can keep track of the interfaces using the J graph from the DPO diagram in Definition 59.



Definition 63 (Interface). For any $G : m \rightarrow n \in \mathbf{LHyp}_\Sigma^\bullet$, we write $\text{io}(G) \in \mathbf{Hyp}_\Sigma$ for the discrete hypergraph with $m + n$ vertices.

Definition 64 (Interfacing). For any $J, H \in \mathbf{Hyp}_\Sigma$, where J is discrete, we say that J interfaces H if there exists

- a partition $V_J = V_\alpha + V_\omega$ equipped with total orders on the two subsets
- a morphism $m : J \rightarrow H$ that is injective when restricted to the two subsets

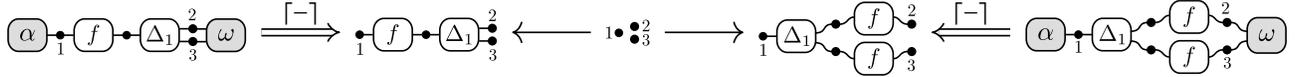
such that

- for any $v \in V_\alpha$, $\text{in}(m(v)) = 0$
- for any $v \in V_\omega$, $\text{out}(m(v)) = 0$

Furthermore, for any $v \in V_H$ not in the image of m , $\text{in}(v), \text{out}(v) > 0$.

We are now ready to formulate the notion of a rewrite rule in $\mathbf{LHyp}_\Sigma^\bullet$.

Definition 65 (Rewrite rule). For a pair of interfaced linear hypergraphs $L, R : m \rightarrow n \in \mathbf{LHyp}_\Sigma^\bullet$, we represent their corresponding rewrite rule in \mathbf{Hyp}_Σ as $[L] \leftarrow \text{io}(L) \rightarrow [R]$, with the monomorphisms $p : \text{io}(L) \rightarrow L$ and $q : \text{io}(L) \rightarrow R$ defined such that for any vertex $v \in \text{io}(L)$, there exists $x \in \mathbb{N}$ such that $p(v) = [\pi_x(T_L[\bullet])]$ and $q(v) = [\pi_x(S_L[\bullet])]$. We write this rewrite rule as $[\langle L, R \rangle]$.



For a rewriting system \mathcal{E} , we write $\llbracket \mathcal{E} \rrbracket$ for the interpretation of its rewrite rules as spans of hypergraphs as illustrated in 65.

Lemma 66. For any interfaced linear hypergraph G , $\text{io}(G)$ interfaces $[G]$ with the partitioning $V_\alpha = [\pi_x(T_G[\bullet])]$, $V_\omega = [\pi_x(S_G[\bullet])]$. For a rewrite span $[\langle L, R \rangle]$, $\text{io}(L)$ interfaces $[L]$ and $[R]$ with the partitioning $V_\alpha = [\pi_x(T_L[\bullet])]$, $V_\omega = [\pi_x(S_L[\bullet])]$.

Proof. For G and L this is immediate. For R , this follows as the left and right hand side of a rewrite rule must have the same domain and codomain. \square

Once we have translated back into regular hypergraphs, we can perform rewriting as described above. Once we have acquired a rewritten hypergraph, we must use the interface J to reconstruct an interfaced linear hypergraph.

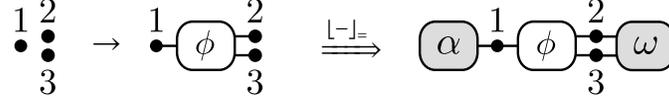
Proposition 67. For a rewrite span $[\langle L, R \rangle] \in \llbracket \mathcal{E} \rrbracket$ and linear hypergraph G , if there exists a matching $L \rightarrow G$, and $G \rightsquigarrow_{[\mathcal{R}]} H$, then H is linear.

Proof. Since $[G]$ is linear, then C must also be linear by homomorphism, and R is linear by Lemma 62. By the DPO diagram, we know that H is the pushout of $C \xleftarrow{p} K \xrightarrow{q} R$, where $K = V_\alpha + V_\omega$. For any $v \in V_\alpha$, $\text{in}(q(v)) = 0$ as K interfaces R and $\text{out}(p(v)) = 0$ as K interfaces L and C is the complement $G - L$. Therefore in the pushout $R \xrightarrow{r} H \xleftarrow{s} C$ for any $v \in V_\alpha$, $r(p(v)) = s(q(v))$ will be the source of an edge from R and the target of an edge from C , and only one each since R and C are linear. A similar argument follows for each $v \in V_\omega$. Therefore H is linear. \square

Lemma 68. For a DPO diagram as illustrated in Definition 59 and rewrite rule $[\langle L, R \rangle]$, if J interfaces G then it also interfaces H .

Proof. As J interfaces G and there exists a morphism $C \rightarrow G$, J must also interface C by homomorphism condition: a morphism must preserve sources and targets so if a vertex is not a source or target in G , it cannot be a source or target in C . The reasoning is then the same as Proposition 67: the pushout hypergraph H only ‘fills the hole’, so it will not affect the interfaces. Therefore J interfaces H . \square

To retrieve an interfaced linear hypergraph from a regular one with interfacing morphism, we use the *reinterface* functor $\llbracket - \rrbracket = : \mathbf{Hyp}_\Sigma \times \mathbf{Hyp}_\Sigma \rightarrow \mathbf{LHyp}_\Sigma^\bullet$.



Definition 69 (Reinterfacing). For any $H, J \in \mathbf{Hyp}_\Sigma$ and morphism $p : J \rightarrow H$ such that J interfaces H with partition $V_\alpha + V_\omega$, we can reinterface H with respect to J with the functor $\llbracket H \rrbracket_J : \mathbf{Hyp}_\Sigma \rightarrow \mathbf{LHyp}_\Sigma^\bullet$, defined as $\llbracket H \rrbracket_J = (E_H, S', T', \lambda_{s_v}, t_v, \Lambda_H)$ where

$$\begin{aligned} S[e \in E_H]' &= \{s_v \text{ fresh in } \mathbb{A} \mid v \in s_H(e)\} & S[\bullet]' &= \{s_v \text{ fresh in } \mathbb{A} \mid v \in p^\star(V_\omega)\} \\ T[e \in E_H]' &= \{t_v \text{ fresh in } \mathbb{A} \mid v \in t_H(e)\} & T[\bullet]' &= \{t_v \text{ fresh in } \mathbb{A} \mid v \in p^\star(V_\alpha)\} \end{aligned}$$

Definition 70 (Rewriting). For two hypergraphs $L, R \in \mathbf{LHyp}_\Sigma^\bullet$ and rewrite rule $\langle L, R \rangle \in \mathcal{R}$, we define the corresponding rewrite rule $\llbracket \langle L, R \rangle \rrbracket \in \mathbf{Hyp}_\Sigma$ as in Definition 65. For a morphism $L \rightarrow G \in \mathbf{LHyp}_\Sigma^\bullet$ such that $\llbracket L \rrbracket \rightarrow \llbracket G \rrbracket$ is a matching, we perform the rewrite $\llbracket G \rrbracket \rightsquigarrow_{\llbracket \mathcal{R} \rrbracket} H \in \mathbf{Hyp}_\Sigma$. The resulting hypergraph in $\mathbf{LHyp}_\Sigma^\bullet$ is $\llbracket H \rrbracket_{\text{io}}(G)$.

We call a monomorphism $L \rightarrow G \in \mathbf{LHyp}_\Sigma^\bullet$ an \bullet -matching if the corresponding morphism $\llbracket L \rrbracket \rightarrow \llbracket G \rrbracket$ is a matching, i.e. $\text{io}(L) \rightarrow \llbracket L \rrbracket \rightarrow \llbracket G \rrbracket$ has a pushout complement.

Theorem 71. For a rewrite rule $\langle L, R \rangle \in \mathbf{LHyp}_\Sigma^\bullet$ and \bullet -matching $L \rightarrow G$, the procedure in Definition 70 yields a unique interfaced linear hypergraph.

Proof. The DPO procedure yields us a unique hypergraph that is linear by Proposition 67. As $\text{io}(G)$ interfaces $\llbracket G \rrbracket$, it also interfaces H by Lemma 68, so we can obtain a unique interfaced linear hypergraph $\llbracket H \rrbracket_{\text{io}}(G)$. \square

In general, although we know that pushout complements are unique in adhesive categories, we do not actually have a guarantee that they exist for a given monomorphism $L \rightarrow G$. Usually some variety of the *no-dangling-hyperedge* condition is used to identify the monomorphisms that allow for the definition of pushout complements. However, in our case we actually can guarantee they exist!

Definition 72 (No-dangling-hyperedge condition). Morphisms $K \xrightarrow{p} L \xrightarrow{q} G$ in \mathbf{Hyp}_Σ satisfy the *no-dangling-hyperedge condition* if, for each hyperedge h not in the image of q , every source and target of h is either (i) not in the image of q or (ii) in the image of $q \circ p$.

Lemma 73. For any $L, G \in \mathbf{Hyp}_\Sigma$, the morphisms $\text{io}(L) \xrightarrow{p} \llbracket L \rrbracket \xrightarrow{q} \llbracket G \rrbracket$ always satisfy the *no-dangling-hyperedge condition*.

Proof. Assume there is an edge $e \in \llbracket G \rrbracket$ that is not in the image of q , with a source $v_G = m(v_L)$. Since $\llbracket L \rrbracket$ is linear (Lemma 62), $\text{out}(v_L) = 0$ by preservation of sources. By definition of $\text{io}(L)$, any vertex with out-degree 0 must be in the image of p , so v_G is in the image of $q \circ p$. The same follows for targets. \square

Theorem 74 (\bullet -matchings). All monomorphisms in $\mathbf{LHyp}_\Sigma^\bullet$ are \bullet -matchings.

Proof. To form a pushout complement C we remove elements of $\llbracket L \rrbracket$ from $\llbracket G \rrbracket$ in the obvious way. This is a well-formed linear hypergraph since $\text{io}(L) \rightarrow \llbracket L \rrbracket \rightarrow \llbracket G \rrbracket$ always satisfies the *no-dangling-hyperedge condition*. \square

Now all that remains is to show that our notion of rewriting in the graph language is equivalent to that in the term language.

Theorem 75. For a set of equations \mathcal{E} in \mathbf{Term}_Σ , $g \Rightarrow_{\mathcal{E}} h$ if and only if $\llbracket g \rrbracket \rightsquigarrow_{\llbracket \mathcal{E} \rrbracket} \llbracket h \rrbracket$.

Proof. For (\Rightarrow) , assume that $g \Rightarrow_{\mathcal{E}} h$. By Definition 53 this means that there exists $\langle l, r \rangle \in \mathcal{R}$ such that

$$\boxed{g} = \boxed{\hat{f}_1} \boxed{l} \boxed{\hat{f}_2} \Rightarrow_{\mathcal{R}} \boxed{\hat{f}_1} \boxed{r} \boxed{\hat{f}_2} = \boxed{h}$$

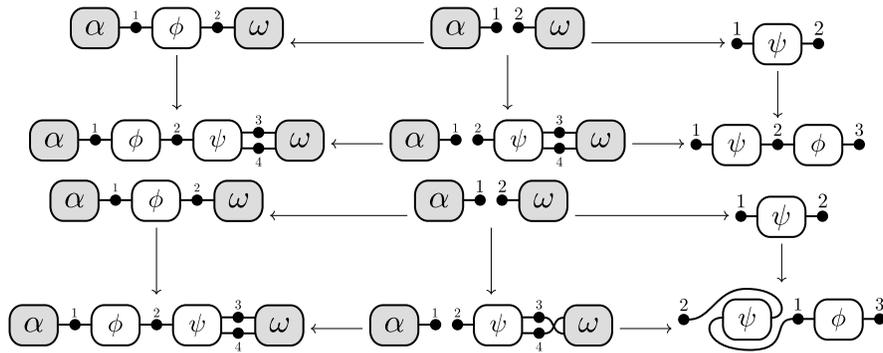
For the if direction, we use $\llbracket - \rrbracket$ to translate the rewrite rule $\langle l, r \rangle$ into a rewrite rule in the language of interfaced linear hypergraphs as described in Definition 65. Therefore there exists a span $\llbracket \llbracket l \rrbracket \rrbracket \leftarrow \text{io}(\llbracket \llbracket l \rrbracket \rrbracket) \rightarrow \llbracket \llbracket r \rrbracket \rrbracket$. Since l is a subterm of f , L corresponds to a subgraph of $\llbracket \llbracket g \rrbracket \rrbracket$ by Lemma 58, so there exists a monomorphism $\llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$. Any monomorphism $\llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$ is an \bullet -matching by Theorem 74 so the pushout complement of $\text{io}(\llbracket \llbracket l \rrbracket \rrbracket) \rightarrow \llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$ exists. We can use the same procedure for $\llbracket \llbracket r \rrbracket \rrbracket$ and $\llbracket \llbracket h \rrbracket \rrbracket$ and assert that the pushout complement of $\text{io}(\llbracket \llbracket r \rrbracket \rrbracket) \rightarrow \llbracket \llbracket r \rrbracket \rrbracket \rightarrow \llbracket \llbracket h \rrbracket \rrbracket$ also exists. We know that the two pushout complements are $C_L = \llbracket \llbracket g \rrbracket \rrbracket - \llbracket \llbracket l \rrbracket \rrbracket$ and $C_R = \llbracket \llbracket h \rrbracket \rrbracket - \llbracket \llbracket r \rrbracket \rrbracket$ respectively. Since the only difference between $\llbracket \llbracket g \rrbracket \rrbracket$ and $\llbracket \llbracket h \rrbracket \rrbracket$ is $\llbracket \llbracket l \rrbracket \rrbracket$ and $\llbracket \llbracket r \rrbracket \rrbracket$ by definition of rewrite rules, $C_L = C_R$. We name this pushout complement C , and note that this implies that $\llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket \leftarrow C$ and $\llbracket \llbracket r \rrbracket \rrbracket \rightarrow \llbracket \llbracket h \rrbracket \rrbracket \leftarrow C$ are pushouts. By Lemma 68, $\text{io}(\llbracket \llbracket g \rrbracket \rrbracket)$ interfaces $\llbracket \llbracket g \rrbracket \rrbracket$ and $\text{io}(\llbracket \llbracket h \rrbracket \rrbracket)$ interfaces $\llbracket \llbracket h \rrbracket \rrbracket$, and as $\llbracket \llbracket g \rrbracket \rrbracket$ and $\llbracket \llbracket h \rrbracket \rrbracket$ have the same type, $\text{io}(\llbracket \llbracket g \rrbracket \rrbracket) \equiv \text{io}(\llbracket \llbracket h \rrbracket \rrbracket)$: we name this hypergraph J . We can conclude that there exist morphisms $J \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$, $J \rightarrow C$ and $J \rightarrow \llbracket \llbracket h \rrbracket \rrbracket$. Therefore the DPO diagram specified in Definition 59 exists, so $G \rightsquigarrow_{\llbracket \mathcal{E} \rrbracket} H$.

For the only if direction, we assume that $\llbracket \llbracket g \rrbracket \rrbracket \rightsquigarrow_{\llbracket \mathcal{E} \rrbracket} \llbracket \llbracket h \rrbracket \rrbracket$ for some rewrite rule $\langle \llbracket \llbracket l \rrbracket \rrbracket, \llbracket \llbracket r \rrbracket \rrbracket \rangle$. This means there exists a DPO diagram with matchings $\llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$ and $\llbracket \llbracket r \rrbracket \rrbracket \rightarrow \llbracket \llbracket h \rrbracket \rrbracket$. Since every monomorphism in $\mathbf{LHyp}_{\Sigma}^{\bullet}$ corresponds to a matching in \mathbf{Hyp}_{Σ} by Theorem 74, this means there must exist monomorphisms $\llbracket \llbracket l \rrbracket \rrbracket \rightarrow \llbracket \llbracket g \rrbracket \rrbracket$ and $\llbracket \llbracket r \rrbracket \rrbracket \rightarrow \llbracket \llbracket h \rrbracket \rrbracket$. Therefore by Lemma 58, l is a subterm of g and r is a subterm of h , so $g \Rightarrow_{\mathcal{E}} h$. \square

The results of this section gives us a graph rewriting system for rules that are spans of monomorphisms, i.e. those in which no vertices of the interface K are collapsed into one. This is since the pushout complement is only uniquely defined for $K \xrightarrow{p} [L] \rightarrow [G]$ if p is mono. However, there are cases where rewrite rules that are *not* spans of monomorphisms might be desirable, such as a rule introducing an edge to an empty wire.

$$\boxed{\alpha} \xrightarrow{1} \boxed{\phi} \xrightarrow{2} \boxed{\omega} \longleftarrow \boxed{\alpha} \xrightarrow{1} \boxed{\omega} \xrightarrow{2} \boxed{\psi}$$

Since the left leg of the span is not a monomorphism, the pushout complement is not unique and therefore multiple derivations can be performed for one matching. Even more crucially, one of the derivations would be degenerate if translated back into an interfaced linear hypergraph: there is a situation where a source connects to a source!



In [5] this phenomenon is permitted thanks to a compact closed structure in which wires can be directed arbitrarily. However in our traced context we have a strict notion of source and target which cannot be

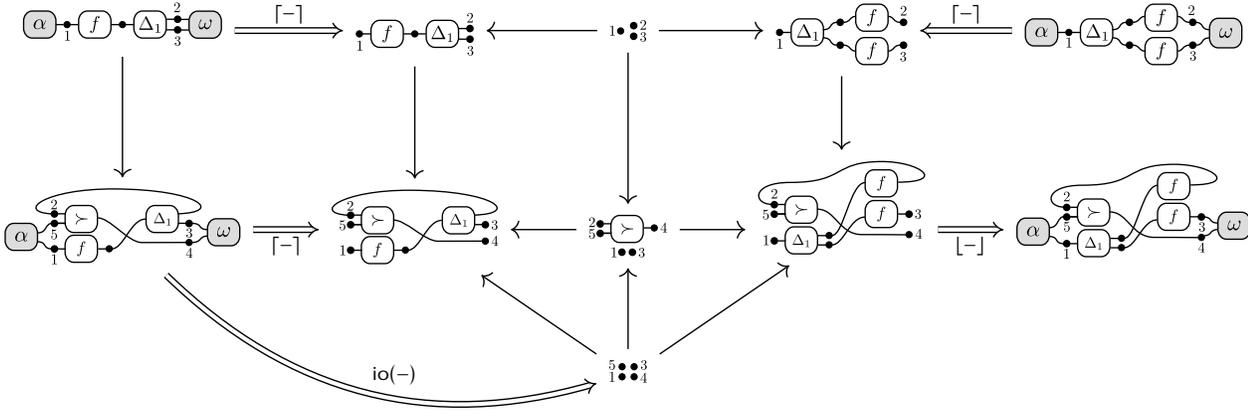
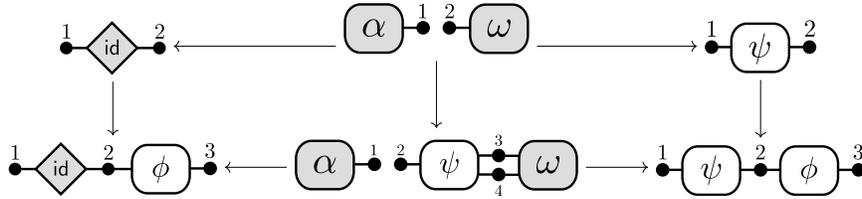


Figure 5: An example of a DPO diagram using the Cartesian copy axiom.

altered. To solve this issue, we can use homeomorphism to perform an expansion on the left hand side of the rule. This yields us a span of monomorphisms as desired and guarantees us a unique pushout complement. Once we have completed our rewriting, we can perform any smoothings if necessary in order to retrieve a minimal hypergraph.

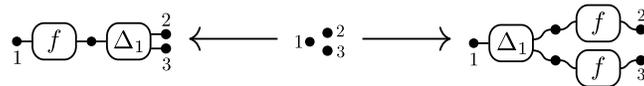


8 Case study: digital circuits

The initial motivation for developing a graph rewriting system was for use as an operational semantics for digital circuits. In this section we will detail how we can apply our framework for this purpose.

As detailed in §2.2.1, when a STMC is Cartesian, it admits a *fixpoint operator*, which we can use to represent feedback. This leads immediately to models of digital circuits. However, as we must now consider terms up to the axioms of Cartesian categories in addition to the axioms of STMCs, equality is no longer captured by graph isomorphism and we must use graph rewriting. In the hypergraph interpretation the diagonal morphism and the unique morphism into the terminal object are represented as edges, and the Cartesian axioms are expressed as graph rewrite rules.

Example 76. We wish to interpret the Cartesian axioms in \mathbf{Term}_Σ as graph rewrite rules in \mathbf{Hyp}_Σ . We start our example by interpreting naturality of Δ_1 as a span of monomorphisms $\Rightarrow_{\Delta_1}: L \xleftarrow{m} C \xrightarrow{n} R$ in \mathbf{Hyp}_Σ :



Terms such as $\text{Tr}^1(\triangleright \otimes f \cdot \sigma_{1,1} \cdot \Delta_1 \otimes 1)$ (with \triangleright as defined in § 3) illustrate why the graph rewriting approach is useful. The symmetry obscures the existence of the left hand side of \Rightarrow_{Δ_n} , so rewriting *qua term* requires the expenditure of additional computation to identify the redex. In contrast, in the hypergraph representation the rule is more easily identifiable as seen in Figure 5.

We present digital circuits as morphisms in a free traced PROP, where the objects correspond to the number of wires in a bus [19]. Morphisms are freely generated over a *circuit signature* containing values $v : 0 \rightarrow 1$, forming a lattice, gates $k : m \rightarrow 1$, and ‘special morphisms’ for forking ($< : 1 \rightarrow 2$), joining ($> : 2 \rightarrow 1$) and stubbing ($\sim : 1 \rightarrow 0$) wires. We write $\mathbf{v} = v_0 \otimes \dots \otimes v_m$. Gates and special morphisms have associated axioms:

$$\begin{array}{ll}
 v \cdot < = v \otimes v & 0 \rightarrow 2 \\
 v \otimes v' \cdot > = v \sqcup v' & 0 \rightarrow 1 \\
 v \cdot \sim = 0 & 0 \rightarrow 0 \\
 \mathbf{v} \cdot k = v \text{ s.t. if } v_i \in \mathbf{v} \sqsupseteq v'_i \in \mathbf{v}' \text{ then } \mathbf{v} \cdot k \sqsupseteq \mathbf{v}' \cdot k & 0 \rightarrow 1
 \end{array}$$

Additionally, we can use the special morphisms to define Cartesian copy and delete maps, along with a dual notion of maps ∇_n to join buses of arbitrary size. The outputs of circuits depend wholly on their inputs, so a circuit will always give the same output for some given inputs. It follows that any circuits with zero outputs are considered to be equal.

Delay is represented as a morphism $\delta_t : 1 \rightarrow 1$, parameterised over a *time monoid* $t \in \mathbf{T}$. Since this adds a temporal aspect to our circuits, we switch from values to *waveforms*, sequences of values over time. The axioms of delay are:

$$\begin{array}{ll}
 \delta \cdot k = k \cdot \delta & m \rightarrow 1 \\
 \delta^2 \otimes v \otimes v' \cdot \nabla_2 \cdot k = ((\delta^2 \cdot k) \otimes (v \otimes v' \cdot k)) \cdot \nabla_1 & 2 \rightarrow 1 \\
 \perp \cdot \delta = \perp & 0 \rightarrow 0 \\
 \delta \cdot \sim = \sim & 0 \rightarrow 0
 \end{array}$$

Of most interest is the second axiom (*Streaming*), an interaction between gates and time which corresponds to stream manipulation. Two copies of the gate k are used, one to handle the ‘head’ of the waveform and the other to handle the ‘tail’. Feedback is represented using the trace, so the fixpoint operator can be used to ‘unfold’ the circuit.

We have already seen in Example 76 that term rewriting can be computationally awkward. Another problem in the reduction of circuits is that some of the local traces can be unproductive due to infinite unfolding. The key result in [19] is that a circuit can be converted to a graph and brought to a normal form in which the reduction can be made efficient. A guarantee can be given for a circuit to be either productive or, if unproductive, the lack of productivity can be efficiently detected.

9 Generalisation

So far, we have considered only terms in PROPs, where objects are natural numbers and tensor product is addition. This corresponds especially well with the notion of a linear hypergraph with m inputs and n outputs corresponding to a morphism $m \rightarrow n$. However, we may also wish to represent categories with arbitrary objects, as we do with string diagrams. It is not difficult to generalise our hypergraphs to correspond to terms in *any* STMC with set of objects \mathcal{C} and morphisms freely generated over signature Σ . We fix this STMC here as $\mathbf{Term}_{\Sigma, \mathcal{C}}$.

Rather than just thinking of the type of linear hypergraphs as $m \rightarrow n$, we could instead decompose them into the form $1 \otimes 1 \otimes \dots \otimes 1 \rightarrow 1 \otimes 1 \otimes \dots \otimes 1$, since tensor product is addition. In fact, this corresponds even more closely to our graphical notation, as each of the ‘wires’ in the graph represents the natural number 1. However, when we write the types this way, we see that there is no reason to restrict ourselves to just numbers. The wires can instead correspond to arbitrary objects, as in string diagrams. We can communicate this idea in our hypergraphs by labelling vertices as well as edges. These labels correspond to the objects in our category.

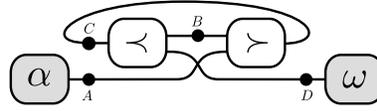
We extend hypergraph signatures with this vertex label set \mathcal{C} to create a *generalised hypergraph signature*. Since the inputs and outputs of boxes are tensors of objects rather than natural numbers, edge labels now have associated domains and codomains of elements of the free monoid under \otimes on \mathcal{C} rather than arities and coarities of natural numbers. We introduce the vertex labelling operations $\Lambda^T : T \rightarrow \mathcal{C}^*$ and $\Lambda^S : S \rightarrow \mathcal{C}^*$ and the following conditions:

- For all $e \in E$, $\Lambda^S(S[e]) = \text{dom}(\Lambda(e))$
- For all $e \in E$, $\Lambda^T(T[e]) = \text{cod}(\Lambda(e))$
- For all $t \in T$ and $s \in S$, $(\Lambda^T \circ \kappa)(t) = \Lambda^S(s)$

Example 77 (Generalised interfaced linear hypergraph). *Below is an example of a generalised interfaced linear hypergraph of type $A \rightarrow D$, for a generalised signature*

$$(\Sigma, \mathcal{C}) = (\{< : C \rightarrow B \otimes D, > : B \otimes A \rightarrow C\}, \{A, B, C\}).$$

The corresponding term is $\text{Tr}^C(< \otimes \text{id}_A \cdot \text{id}_B \otimes \sigma_{D,A} \cdot > \otimes \text{id}_D)$.



Generalised interfaced linear hypergraph homomorphisms are as with interfaced linear hypergraphs but with the addition of vertex label conditions.

$$\begin{array}{ccc} S_F & \xrightarrow{\Lambda_F^S} & \mathcal{C} \\ \downarrow h_S & & \downarrow \text{id} \\ S_G & \xrightarrow{\Lambda_G^S} & \mathcal{C} \end{array} \quad \begin{array}{ccc} T_F & \xrightarrow{\Lambda_F^T} & \mathcal{C} \\ \downarrow h_T & & \downarrow \text{id} \\ T_G & \xrightarrow{\Lambda_G^T} & \mathcal{C} \end{array}$$

As before, generalised interfaced linear hypergraphs over the signature (Σ, \mathcal{C}) are the objects in the category $\mathbf{LHyp}_{\Sigma, \mathcal{C}}^\bullet$, with generalised interfaced linear hypergraph homomorphisms as the morphisms between them. Operations on generalised hypergraphs are defined similarly to before but with the addition of vertex labels, which are affected in the same way as edge labels. Therefore, we can also form a category where the generalised linear hypergraphs are morphisms:

Definition 78 (Generalised category of hypergraphs). *Generalised interfaced linear hypergraphs over the signature (Σ, \mathcal{C}) form a symmetric traced monoidal category $\mathbf{HypTerm}_{\Sigma, \mathcal{C}}$ with objects the elements of \mathcal{C}^* , elements of $\mathbf{HypTerm}_{\Sigma, \mathcal{C}}(M, N)$ as the generalised hypergraphs of type $M \rightarrow N$, unit of composition as the identity hypergraph, monoidal tensor as \otimes as the empty hypergraph, symmetry on M and N as $\sigma_{M, N}$, and trace on X as*

$$\text{Tr}^X(-) : \mathbf{HypTerm}_{\Sigma, \mathcal{C}}(X \otimes M, X \otimes N) \rightarrow \mathbf{HypTerm}_{\Sigma, \mathcal{C}}(M, N).$$

We define the operations

$$\begin{aligned} \llbracket - \rrbracket_{\Sigma, \mathcal{C}} &: \mathbf{Term}_{\Sigma, \mathcal{C}} \rightarrow \mathbf{HypTerm}_{\Sigma, \mathcal{C}} \\ \langle\langle - \rangle\rangle_{\Sigma, \mathcal{C}, \leq} &: \mathbf{HypTerm}_{\Sigma, \mathcal{C}} \rightarrow \mathbf{Term}_{\Sigma, \mathcal{C}} \end{aligned}$$

in a similar manner to before. Soundness and completeness translate smoothly into the generalised case.

Theorem 79 (Generalised soundness). *For any morphisms $f, g \in \mathbf{Term}_{\Sigma, \mathcal{C}}$ if $f = g$ under the equational theory of the category then their interpretations as morphisms are isomorphic, $\llbracket f \rrbracket_{\Sigma, \mathcal{C}} \equiv \llbracket g \rrbracket_{\Sigma, \mathcal{C}}$.*

Theorem 80 (Generalised completeness). *For any generalised interfaced linear hypergraph $F \in \mathbf{HypTerm}_{\Sigma, \mathcal{C}}$ there exists a unique morphism $f \in \mathbf{Term}_{\Sigma, \mathcal{C}}$, up to the equations of the STMC, such that $\llbracket f \rrbracket_{\Sigma, \mathcal{C}} = F$. Furthermore, for any morphism $f \in \mathbf{Term}_{\Sigma, \mathcal{C}}$, $\langle\langle \llbracket f \rrbracket \rangle\rangle = f$.*

10 Related and further work

10.1 Related work

Diagrammatic languages for traced categories are certainly not new: their formal presentation as *string diagrams* has existed since the 90s [27, 28]. A soundness and completeness theorem for this language, while common knowledge for many years but often omitted or only proved for certain signatures [39], was finally formally proved in [31]. Combinatorial languages predate even this, having existed since at least the 80s in the guise of *flowchart schemes* [42, 10, 11]. This type of diagrams have also been used to show the completeness of finite dimensional vector spaces [24], and when equipped with a dagger, Hilbert spaces [40]. However, it was not shown whether these diagrams were also suitable for graph rewriting in the presence of Cartesian structure, which is essential for our goal of obtaining an operational semantics of digital circuits.

Graphical languages for traced categories have seen many applications, such as to illustrate cyclic lambda calculi [21], or to reason graphically about programs [38]. But we are not just concerned with diagrammatic languages as a standalone concept: we are interested in the context of performing *graph rewriting* with them in order to reason with additional axioms. This has been studied in the context symmetric traced categories before, most notably with *open graphs* [14], which were developed into *framed point graphs* [30], and *hypergraphs* [5, 44, 7, 8]. As we have already established, these approaches were unsuitable in the context of digital circuits, due to the use of a compact closed or Frobenius structure. However, our work uses some of the building blocks from these frameworks, such as the use of homeomorphism and rewriting using hypergraphs.

Unlike the explicit interfaces in our framework, the hypergraphs in the existing framework are defined via a cospan structure. Cospans are used in categorical representations of open networks [17, 1], with each leg of the cospan representing inputs and outputs respectively. This is natural in the presence of a Frobenius monoid which allows vertices to be identified arbitrarily. A similar recipe could have been followed here by requiring the morphisms of the cospan to be injective, which would have simplified some of the proofs. However, we considered a more elementary presentation in which one can arrive at the desired technical result without relying on avoidable mathematical concepts, which may make it more accessible to a wider audience. Regardless, it should be easy to see how to encode an interfaced linear hypergraph $H : m \rightarrow n$ as a cospan of regular hypergraphs $m \rightarrow [H] \leftarrow n$.

10.2 Further work

In this paper we have revisited and solidified the mathematical foundation upon which the graphical language for digital circuits of [19] is based. This graphical language opens up numerous avenues, such as additional opportunities for partial evaluation or allowing us to reason with circuits not handled well by traditional methods, such as cyclic combinational circuits [36].

A potential next step is to refine the existing categorical semantics of digital circuits and identify any missing axioms. Indeed, for us to have a complete diagrammatic semantics the underlying categorical framework must of course also be complete! Instantiating the categorical semantics to a concrete category [20] pointed towards such additional axioms, most notably regarding *unproductive circuits*, as mentioned in the case study. By identifying these axioms we can present our diagrammatic semantics as a complete package for reasoning about digital circuits.

Another natural future avenue is that of *automating* the graph rewrites. While it may be simple to identify potential redexes by eye in small systems, in practice there may be many and the derivation procedure would be long and tedious. Automating rewrites presents some additional issues, such as the task of choosing between different rewrites. We can take the *global trace-delay form* of [20] and verify in our formal framework that we do in fact have confluence for our operational semantics.

Our firm, mathematically sound graphical foundation also opens up the opportunity for the development of circuit design tools, following in the footsteps of tools like such as Quantomatic [33], Homo-

topy.io [25] or Cartographer [41], with the aim of bringing the reduction-based operational semantics into a field dominated by simulation. Our approach serves to contrast and complement, not replace, the existing paradigm, and offer an alternative insight into the design and evaluation of digital circuits.

References

- [1] J. C. Baez and K. Courser. Structured cospans, 2020. URL <https://arxiv.org/abs/1911.04630>.
- [2] S. L. Bloom and Z. Ésik. Iteration theories. In *Iteration Theories*, pages 159–213. Springer, 1993. doi:[10.1007/978-3-642-78034-9](https://doi.org/10.1007/978-3-642-78034-9).
- [3] F. Bonchi, P. Sobociński, and F. Zanasi. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*, pages 435–450. Springer, 2014. doi:[10.1007/978-3-662-44584-6_30](https://doi.org/10.1007/978-3-662-44584-6_30).
- [4] F. Bonchi, P. Sobocinski, and F. Zanasi. Full abstraction for signal flow graphs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '15*, page 515–526. Association for Computing Machinery, 2015. doi:[10.1145/2676726.2676993](https://doi.org/10.1145/2676726.2676993).
- [5] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, and F. Zanasi. Rewriting modulo symmetric monoidal structure. In *2016 31st Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–10. IEEE, 2016. doi:[10.1145/2933575.2935316](https://doi.org/10.1145/2933575.2935316).
- [6] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobociński, and F. Zanasi. Confluence of graph rewriting with interfaces. In *European Symposium on Programming*, pages 141–169. Springer, 2017. doi:[10.1007/978-3-662-54434-1_6](https://doi.org/10.1007/978-3-662-54434-1_6).
- [7] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski, and F. Zanasi. Rewriting with Frobenius. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 165–174, 2018. doi:[10.1145/3209108.3209137](https://doi.org/10.1145/3209108.3209137).
- [8] F. Bonchi, F. Gadducci, A. Kissinger, P. Sobocinski, and F. Zanasi. String diagram rewrite theory i: Rewriting with frobenius structure, 2020. URL <https://arxiv.org/abs/2012.01847>.
- [9] S. Castellan and P. Clairambault. Causality vs. interleavings in concurrent game semantics. In *The 27th International Conference on Concurrency Theory (CONCUR 2016)*, volume 32, pages 1 – 3214, Québec City, Canada, Aug. 2016. doi:[10.4230/LIPIcs.CONCUR.2016.32](https://doi.org/10.4230/LIPIcs.CONCUR.2016.32).
- [10] V. E. Cazanescu and G. Stefanescu. Towards a new algebraic foundation of flowchart scheme theory. *Fundamenta Informaticae*, 13:171–210, 1990. doi:[10.5555/97367.97373](https://doi.org/10.5555/97367.97373).
- [11] V. E. Cazanescu and G. Stefanescu. Feedback, iteration, and repetition. In *Mathematical Aspects of Natural and Formal Languages*, 1994. doi:[10.1142/9789814447133_0003](https://doi.org/10.1142/9789814447133_0003).
- [12] B. Coecke and A. Kissinger. Picturing quantum processes. In *International Conference on Theory and Application of Diagrams*, pages 28–31. Springer, 2018. doi:[10.1007/978-3-319-91376-6_6](https://doi.org/10.1007/978-3-319-91376-6_6).
- [13] B. Coecke, M. Sadrzadeh, and S. Clark. Mathematical foundations for a compositional distributional model of meaning, 2010. URL <https://arxiv.org/abs/1003.4394>.
- [14] L. Dixon and A. Kissinger. Open graphs and monoidal theories. *Mathematical Structures in Computer Science*, 23:308–359, 2013. doi:[10.1017/S0960129512000138](https://doi.org/10.1017/S0960129512000138).
- [15] H. Ehrig, M. Pfender, and H. J. Schneider. Graph-grammars: An algebraic approach. In *14th Annual Symposium on Switching and Automata Theory (swat 1973)*, pages 167–180. IEEE, 1973. doi:[10.1109/SWAT.1973.11](https://doi.org/10.1109/SWAT.1973.11).

-
- [16] H. Ehrig, A. Habel, H.-J. Kreowski, and F. Parisi-Presicce. Parallelism and concurrency in high-level replacement systems. *Mathematical Structures in Computer Science*, 1(3):361–404, 1991. doi:[10.1017/S0960129500001353](https://doi.org/10.1017/S0960129500001353).
- [17] B. Fong. Decorated cospans, 2015. URL <https://arxiv.org/abs/1502.00872>.
- [18] D. R. Ghica. A knot theory for eight year olds. *Mathematical Teaching*, (264-268), 2018. URL <https://www.atm.org.uk/Mathematics-Teaching-Journal-Archive/149275>.
- [19] D. R. Ghica and A. Jung. Categorical semantics of digital circuits. In *Proceedings of the 16th Conference on Formal Methods in Computer-Aided Design*, pages 41–48. FMCAD Inc, 2016. doi:[10.1109/FMCAD.2016.7886659](https://doi.org/10.1109/FMCAD.2016.7886659).
- [20] D. R. Ghica, A. Jung, and A. Lopez. Diagrammatic Semantics for Digital Circuits. In *26th EACSL Annual Conference on Computer Science Logic (CSL 2017)*, volume 82, pages 24:1–24:16. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017. doi:[10.4230/LIPIcs.CSL.2017.24](https://doi.org/10.4230/LIPIcs.CSL.2017.24).
- [21] M. Hasegawa. Recursion from cyclic sharing: traced monoidal categories and models of cyclic lambda calculi. In *International Conference on Typed Lambda Calculi and Applications*, pages 196–213. Springer, 1997. doi:[10.1007/3-540-62688-3_37](https://doi.org/10.1007/3-540-62688-3_37).
- [22] M. Hasegawa. On traced monoidal closed categories. *Mathematical Structures in Computer Science*, 19(2):217–244, 2009. doi:[10.1017/S0960129508007184](https://doi.org/10.1017/S0960129508007184).
- [23] M. Hasegawa. *Models of Sharing Graphs: A Categorical Semantics of let and letrec*. Springer Science & Business Media, 2012. doi:[10.1007/978-1-4471-0865-8](https://doi.org/10.1007/978-1-4471-0865-8).
- [24] M. Hasegawa, M. Hofmann, and G. Plotkin. Finite dimensional vector spaces are complete for traced symmetric monoidal categories. In *Pillars of computer science*, pages 367–385. Springer, 2008. doi:[10.1007/978-3-540-78127-1_20](https://doi.org/10.1007/978-3-540-78127-1_20).
- [25] L. Heidemann, N. Hu, and J. Vicary. *homotopy.io*, 2019. URL <https://doi.org/10.5281/zenodo.2540764>.
- [26] R. Houston. Finite products are biproducts in a compact closed category. *Journal of Pure and Applied Algebra*, 212(2):394–400, 2008. doi:[10.1016/j.jpaa.2007.05.021](https://doi.org/10.1016/j.jpaa.2007.05.021).
- [27] A. Joyal and R. Street. The geometry of tensor calculus, i. *Advances in mathematics*, 88(1):55–112, 1991. doi:[10.1016/0001-8708\(91\)90003-P](https://doi.org/10.1016/0001-8708(91)90003-P).
- [28] A. Joyal, R. Street, and D. Verity. Traced monoidal categories. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 119, pages 447–468. Cambridge University Press, 1996. doi:[10.1017/S0305004100074338](https://doi.org/10.1017/S0305004100074338).
- [29] G. M. Kelly and M. L. Laplaza. Coherence for compact closed categories. *Journal of pure and applied algebra*, 19:193–213, 1980. doi:[10.1016/0022-4049\(80\)90101-2](https://doi.org/10.1016/0022-4049(80)90101-2).
- [30] A. Kissinger. Pictures of processes: Automated graph rewriting for monoidal categories and applications to quantum computing, 2012. URL <https://arxiv.org/abs/1203.0202>.
- [31] A. Kissinger. Abstract tensor systems as monoidal categories. In *Categories and Types in Logic, Language, and Physics*, pages 235–252. Springer, 2014.
- [32] A. Kissinger and S. Uijlen. A categorical semantics for causal structure. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12, 2017. doi:[10.1109/LICS.2017.8005095](https://doi.org/10.1109/LICS.2017.8005095).
-

- [33] A. Kissinger and V. Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *Automated Deduction - CADE-25*, pages 326–336. Springer International Publishing, 2015. ISBN 978-3-319-21401-6. doi:[10.1007/F978-3-319-21401-6_22](https://doi.org/10.1007/F978-3-319-21401-6_22).
- [34] S. Lack. Composing PROPs. *Theory and Applications of Categories*, 13(9):147–163, 2004. URL <http://www.tac.mta.ca/tac/volumes/13/9/13-09abs.html>.
- [35] S. Lack and P. Sobociński. Adhesive categories. In *International Conference on Foundations of Software Science and Computation Structures*, pages 273–288. Springer, 2004. doi:[10.1007/978-3-540-24727-2_20](https://doi.org/10.1007/978-3-540-24727-2_20).
- [36] S. Malik. Analysis of cyclic combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(7):950–956, 1994. doi:[10.1109/43.293952](https://doi.org/10.1109/43.293952).
- [37] A. M. Pitts. *Nominal sets: Names and symmetry in computer science*. Cambridge University Press, 2013. ISBN 9781107017788. doi:[10.1017/CBO9781139084673](https://doi.org/10.1017/CBO9781139084673).
- [38] R. Schweimeier and A. Jeffrey. A categorical and graphical treatment of closure conversion. *Electronic Notes in Theoretical Computer Science*, 20:481–511, 1999. doi:[10.1016/S1571-0661\(04\)80090-2](https://doi.org/10.1016/S1571-0661(04)80090-2).
- [39] P. Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. doi:[10.1007/978-3-642-12821-9_4](https://doi.org/10.1007/978-3-642-12821-9_4).
- [40] P. Selinger. Finite dimensional hilbert spaces are complete for dagger compact closed categories. *Logical Methods in Computer Science*, 8(3), Aug 2012. ISSN 1860-5974. doi:[10.2168/lmcs-8\(3:6\)2012](https://doi.org/10.2168/lmcs-8(3:6)2012).
- [41] P. Sobociński, P. Wilson, and F. Zanasi. Cartographer: a tool for string diagrammatic reasoning. In *8th Conference on Algebra and Coalgebra in Computer Science (CALCO 2019)*, 2019. doi:[10.4230/LIPIcs.CALCO.2019](https://doi.org/10.4230/LIPIcs.CALCO.2019).
- [42] G. Ștefănescu. *Feedback Theories (a Calculus for Isomorphism Classes of Flowchart Schemes)*. Inst. de Mat., 1986.
- [43] G. Stefanescu. *Network Algebra*. Springer Science & Business Media, 2000. ISBN 978-1-4471-0479-7. doi:[10.1007/978-1-4471-0479-7](https://doi.org/10.1007/978-1-4471-0479-7).
- [44] F. Zanasi. Rewriting in free hypergraph categories. *Electronic Proceedings in Theoretical Computer Science*, 263:16–30, Dec 2017. ISSN 2075-2180. doi:[10.4204/eptcs.263.2](https://doi.org/10.4204/eptcs.263.2).

A Axioms of STMCS

A.1 Left identity of composition

For any interfaced linear hypergraph $F : m \rightarrow n$, $\text{id}_m \cdot F \equiv F$.

$$\begin{array}{c} \alpha \\ \omega \end{array} \cdot \begin{array}{c} \alpha \\ F \\ \omega \end{array} \equiv \begin{array}{c} \alpha \\ F \\ \omega \end{array}$$

Definitions

$$\begin{aligned}
H &= \text{id}_m \cdot F \\
E_H &= E_F & S_H[\bullet] &= S_F[\bullet] & S_H[e \in E_F] &= S_F[e] & T_H[\bullet] &= [m] & T_H[e \in E_F] &= T_F[e] & \Lambda_H &= \Lambda_F \\
\kappa_H(t) &= \begin{cases} \kappa_F(\pi_i(T_F[\bullet])) & \text{if } v = \pi_i([m]) \\ \kappa_F(t) & \text{otherwise} \end{cases}
\end{aligned}$$

Equivalence maps

$$h_S(v) = v \quad h_T(v) = \begin{cases} \pi_i(T_F[\bullet]) & \text{if } v = \pi_i([m]) \\ v & \text{otherwise} \end{cases} \quad h_E(e) = e$$

A.2 Right identity of composition

For any interfaced linear hypergraph $F : m \rightarrow n$, $F \cdot \text{id}_m \equiv F$.

$$\begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} F \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \equiv \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} F \\ \bullet \\ \omega \end{array}$$

Definitions

$$H = F \cdot \text{id}_n$$

$$E_H = E_F \quad S_H[\bullet] = [m] \quad S_H[e \in E_F] = S_F[e] \quad T_H[\bullet] = T_F[\bullet] \quad T_H[e \in E_F] = T_F[e] \quad \Lambda_H = \Lambda_F$$

$$\kappa_H(v) = \begin{cases} \pi_i([n]) & \text{if } \kappa_F(v) = \pi_i(S_F[\bullet]) \\ \kappa_F & \text{otherwise} \end{cases}$$

Equivalence maps

$$h_S(v) = v \quad h_S(v) = \begin{cases} \pi_i([n]) & \text{if } v = \pi_i(S[\bullet]) \\ v & \text{otherwise} \end{cases} \quad h_E(e) = e$$

A.3 Associativity of composition

For any interfaced linear hypergraphs $F : m \rightarrow n$, $G : n \rightarrow p$ and $H : p \rightarrow q$, $(F \cdot G) \cdot H \equiv F \cdot (G \cdot H)$.

$$\begin{array}{c} \left(\begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} F \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \right) \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} G \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \\ \equiv \\ \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} F \\ \bullet \\ \omega \end{array} \cdot \left(\begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} G \\ \bullet \\ \omega \end{array} \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \right) \cdot \begin{array}{c} \alpha \\ \bullet \\ \omega \end{array} \end{array}$$

Definitions

$$K = (F \cdot G) \cdot H$$

$$E_K = E_F + E_G + E_H \quad \Lambda_K = \Lambda_F + \Lambda_G + \Lambda_H$$

$$S_K = S_F - S_F[\bullet] + S_G - S_G[\bullet] + S_H \quad T_K = T_F + T_G - T_G[\bullet] + T_H - T_H[\bullet]$$

$$\kappa_K(v) = \begin{cases} \kappa_H(\pi_j(S_H[\bullet])) & \text{if } \kappa_F(v) = \pi_i(S_F[\bullet]) \wedge \kappa_G(\pi_i(T_G[\bullet])) = \pi_j(S_G[\bullet]) \\ \kappa_H(\pi_i(S_H[\bullet])) & \text{if } \kappa_G(v) = \pi_i(S_G[\bullet]) \\ \kappa_G(\pi_i(S_G[\bullet])) & \text{if } \kappa_F(v) = \pi_i(S_F[\bullet]) \\ \kappa_F + \kappa_G + \kappa_H & \text{otherwise} \end{cases}$$

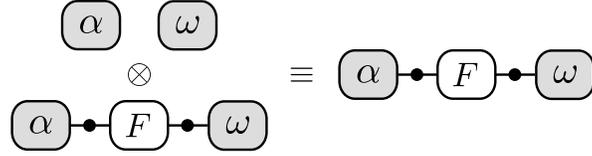
The definition of $L = F \cdot (G \cdot H)$ is identical.

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.4 Left identity of tensor

For any interfaced linear hypergraph $F : m \rightarrow n$, $\text{id}_0 \otimes F \equiv F$.



Definitions

$$H = \text{id}_0 \otimes F$$

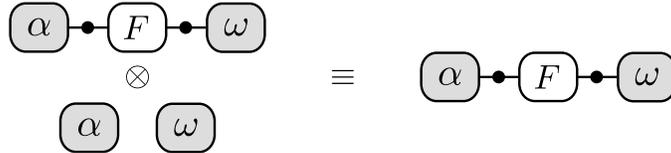
$$E_H = E_F \quad S_H = F \quad T_H = F \quad \kappa_H = \kappa_F \quad \Lambda_H = \Lambda_F$$

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.5 Right identity of tensor

For any interfaced linear hypergraph $F : m \rightarrow n$, $F \otimes \text{id}_0 \equiv F$.



Definitions

$$H = F \otimes \text{id}_0$$

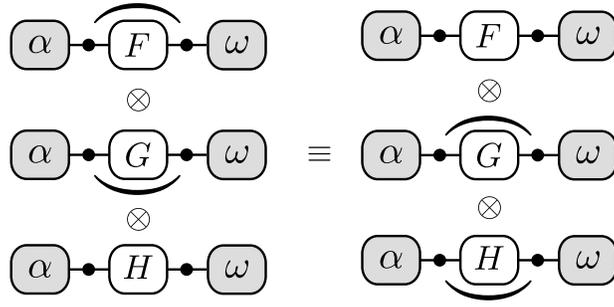
$$E_H = E_F \quad S_H = F \quad T_H = F \quad \kappa_H = \kappa_F \quad \Lambda_H = \Lambda_F$$

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.6 Associativity of tensor

For any interfaced linear hypergraphs $F : m \rightarrow n$, $G : p \rightarrow q$ and $H : r \rightarrow s$, $F \otimes (G \otimes H) \equiv (F \otimes G) \otimes H$.



Definitions

$$\begin{aligned}
 K &= F \otimes (G \otimes H) \\
 E_K &= E_F + E_G + E_H & \Lambda_H &= \Lambda_F + \Lambda_G + \Lambda_H \\
 S_K[e \in E_F] &= S_F[e] & S_K[e \in E_G] &= S_G[e] & S_K[e \in E_H] &= S_H[e] & S_K[\bullet] &= S_F[\bullet] + S_G[\bullet] + S_H[\bullet] \\
 T_K[e \in E_F] &= T_F[e] & T_K[e \in E_G] &= T_G[e] & T_K[e \in E_H] &= T_H[e] & T_K[\bullet] &= T_F[\bullet] + T_G[\bullet] + T_H[\bullet]
 \end{aligned}$$

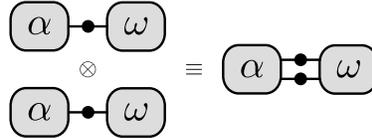
The definition of $L = (F \otimes G) \otimes H$ is identical.

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.7 Bifunctoriality of tensor I (Proposition 21)

For any $m, n \in \mathbb{N}$, $\text{id}_m \otimes \text{id}_n \equiv \text{id}_{m+n}$



Definitions

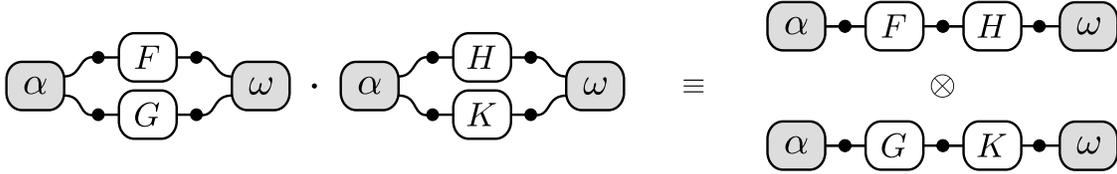
$$\begin{aligned}
 H &= \text{id}_m \otimes \text{id}_n \\
 E_H &= \emptyset & \Lambda_H &= \emptyset & S_H[\bullet] &= [m] + [n] & T_H[\bullet] &= [m] + [n] & \kappa(\pi_i(T_H[\bullet])) &= \pi_i(S_H[\bullet]) \\
 L &= \text{id}_{m+n} \\
 E_H &= \emptyset & \Lambda_H &= \emptyset & S_H[\bullet] &= [m+n] & T_H[\bullet] &= [m+n] & \kappa(\pi_i(T_H[\bullet])) &= \pi_i(S_H[\bullet])
 \end{aligned}$$

Equivalence maps

$$h_S(s) = \begin{cases} \pi_i([m+n]) & \text{if } s = \pi_i([m]) \\ \pi_{i+m}([m+n]) & \text{if } s = \pi_i[n] \end{cases} \quad h_T(t) = \begin{cases} \pi_i([m+n]) & \text{if } t = \pi_i([m]) \\ \pi_{i+m}([m+n]) & \text{if } t = \pi_i[n] \end{cases} \quad h_E = \emptyset$$

A.8 Bifactoriality of tensor II (Proposition 22)

For any interfaced linear hypergraphs $F : m \rightarrow n$, $G : r \rightarrow s$, $H : n \rightarrow p$ and $K : s \rightarrow t$, $F \otimes G \cdot H \otimes K \equiv (F \cdot H) \otimes (G \cdot K)$.



Definitions

$$\begin{aligned}
 L &= F \otimes G \cdot H \otimes K \\
 E_L &= E_F + E_G + E_H + E_K & \Lambda_L &= \Lambda_F + \Lambda_G + \Lambda_H + \Lambda_L \\
 S_L[\bullet] &= S_H[\bullet] + S_L[\bullet] & S_L[e \in E_F] &= S_F[e] & S_L[e \in E_G] &= S_G[e] & S_L[e \in E_H] &= S_H[e] & S_L[e \in E_K] &= S_K[e] \\
 T_L[\bullet] &= T_F[\bullet] + T_G[\bullet] & T_L[e \in E_F] &= T_F[e] & T_L[e \in E_G] &= T_G[e] & T_L[e \in E_H] &= T_H[e] & T_L[e \in E_K] &= T_K[e] \\
 \kappa(t) &= \begin{cases} \kappa(\pi_i(T_H[\bullet])) & \text{if } \kappa_F(t) = \pi_i(S_F[\bullet]) \\ \kappa(\pi_i(T_K[\bullet])) & \text{if } \kappa_G(t) = \pi_i(S_G[\bullet]) \\ (\kappa_F + \kappa_G + \kappa_H + \kappa_K)(t) & \text{otherwise} \end{cases}
 \end{aligned}$$

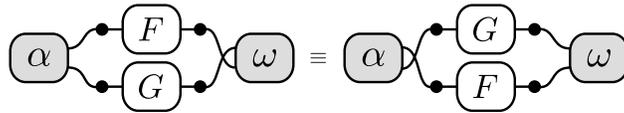
The definition of $A = (F \cdot H) \otimes (G \otimes K)$ is identical.

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.9 Naturality of swap (Proposition 26)

For any interfaced linear hypergraphs $F : m \rightarrow n$ and $G : p \rightarrow q$, $F \otimes G \cdot \sigma_{n,q} \equiv \sigma_{m,p} \cdot G \otimes F$.



Definition

We use the ‘alternate’ representation of the swap hypergraph defined in Lemma 25.

$$\begin{aligned}
 H &= F \otimes G \cdot \sigma_{n,q} \\
 E_H &= E_F + E_G & S_H[\bullet] &= [n] + [q] & S_H[e \in E_F] &= S_F[e] & S_H[e \in E_G] &= S_G[e] \\
 \Lambda_H &= \Lambda_F + \Lambda_G & T_H[\bullet] &= T_F[\bullet] & T_H[e \in E_F] &= T_F[e] & T_H[e \in E_G] &= T_G[e] \\
 \kappa_H(t) &= \begin{cases} \pi_i([n]) & \text{if } \kappa_F(t) = \pi_i(S_F[\bullet]) \\ \pi_i([q]) & \text{if } \kappa_G(t) = \pi_i(S_G[\bullet]) \\ (\kappa_F + \kappa_G)(t) & \text{otherwise} \end{cases}
 \end{aligned}$$

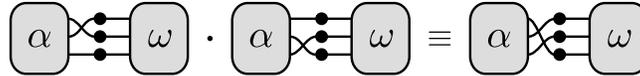
$$\begin{aligned}
 K &= \sigma_{m,p} \cdot F \otimes G \\
 E_H &= E_F + E_G & S_H[\bullet] &= S_F[\bullet] & S_H[e \in E_F] &= S_F[e] & S_H[e \in E_G] &= S_G[e] \\
 \Lambda_H &= \Lambda_F + \Lambda_G & T_H[\bullet] &= [m] + [p] & T_H[e \in E_F] &= T_F[e] & T_H[e \in E_G] &= T_G[e] \\
 \kappa_H(t) &= \begin{cases} \kappa_F(\pi_i(T_F[\bullet])) & \text{if } t = \pi_i([m]) \\ \kappa_G(\pi_i(T_G[\bullet])) & \text{if } t = \pi_i([p]) \\ (\kappa_F + \kappa_G)(t) & \text{otherwise} \end{cases}
 \end{aligned}$$

Equivalence maps

$$h_S(s) = \begin{cases} \pi_i(S_F[\bullet]) & \text{if } s = \pi_i([m]) \\ \pi_i(S_G[\bullet]) & \text{if } s = \pi_i([p]) \\ s & \text{otherwise} \end{cases} \quad h_T(t) = \begin{cases} \pi_i([m]) & \text{if } s = \pi_i(T_F[\bullet]) \\ \pi_i([p]) & \text{if } s = \pi_i(T_G[\bullet]) \\ t & \text{otherwise} \end{cases} \quad h_E = \text{id}$$

A.10 Hexagon axiom

For any $m, n, p \in \mathbb{N}$, $\sigma_{m,n} \otimes \text{id}_p \cdot \text{id}_n \otimes \sigma_{m,p} \equiv \sigma_{m,n+p}$.



Definitions

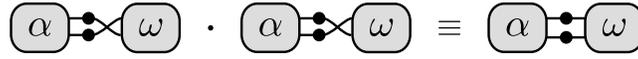
$$\begin{aligned}
 H &= \sigma_{m,n} \otimes \text{id}_p \cdot \text{id}_n \otimes \sigma_{m,p} \\
 E_H &= \emptyset & \Lambda_H &= \emptyset & S_H[\bullet] &= [n]_s + [p]_s + [m]_s & T_H[\bullet] &= [m]_t + [n]_t + [p]_t \\
 \kappa_H(\pi_i([m]_t)) &= \pi_i([m]_s) & \kappa_H(\pi_i([n]_t)) &= \pi_i([n]_s) & \kappa_H(\pi_i([p]_t)) &= \pi_i([p]_s) \\
 K &= \sigma_{m,n+p} \\
 E_K &= \emptyset & \Lambda_K &= \emptyset & S_K[\bullet] &= [n+p]_s + [m]_s & T_K[\bullet] &= [m]_t + [n+p]_t \\
 \kappa_K(\pi_i([m]_t)) &= \pi_i([m]_s) & \kappa_K(\pi_i([n+p]_t)) &= \pi_i([n+p]_s)
 \end{aligned}$$

Equivalence maps

$$h_S(v) = \begin{cases} \pi_i([n+p]_s) & \text{if } v = \pi_i([n]_s) \\ \pi_{i+n}([n+p]_s) & \text{if } v = \pi_i([p]_s) \\ \pi_i([m]_s) & \text{if } v = \pi_i([m]_s) \end{cases} \quad h_T(v) = \begin{cases} \pi_i([n+p]_t) & \text{if } v = \pi_i([n]_t) \\ \pi_{i+n}([n+p]_t) & \text{if } v = \pi_i([p]_t) \\ \pi_i([m]_t) & \text{if } v = \pi_i([m]_t) \end{cases} \quad h_E = \emptyset$$

A.11 Self-invertability

For any $m, n \in \mathbb{N}$, $\sigma_{m,n} \cdot \sigma_{n,m} \equiv \text{id}_{m+n}$.



Definitions

$$\begin{aligned}
H &= \sigma_{m,n} \cdot \sigma_{n,m} \\
E_H &= \emptyset & \Lambda_H &= \emptyset & S_H[\bullet] &= [m]_s + [n]_s & T_H[\bullet] &= [m]_t + [n]_t \\
\kappa_H(\pi_i([m]_t)) &= \pi_i([m]_s) & \kappa_H(\pi_i([n]_t)) &= \pi_i([n]_s)
\end{aligned}$$

The definition of id_{m+n} is identical.

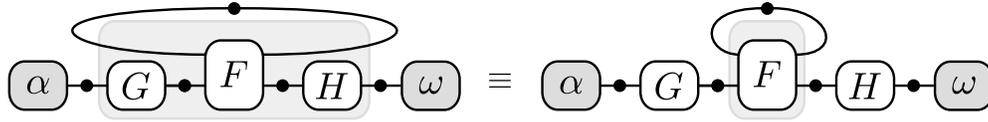
Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad =_E \emptyset$$

A.12 Tightening

For any interfaced linear hypergraphs $F : x + m \rightarrow x + n$, $G : p \rightarrow m$ and $H : n \rightarrow q$,

$$\text{Tr}^x(\text{id}_x \otimes G \cdot F \cdot \text{id}_x \otimes H) \equiv G \cdot \text{Tr}^x(F) \cdot H.$$



This proof is by induction on x .

Zero case: $x = 0$

$$\text{Tr}^0(\text{id}_0 \otimes G \cdot F \cdot \text{id}_0 \otimes H) \equiv G \cdot \text{Tr}^0(F) \cdot H$$

$$\begin{aligned}
\text{Tr}^0(\text{id}_0 \otimes G \cdot F \cdot \text{id}_0 \otimes H) &\equiv \text{id}_0 \otimes G \cdot F \cdot \text{id}_0 \otimes H && \text{definition of trace} \\
&\equiv G \cdot F \cdot \text{id}_0 \otimes H && \text{left identity of tensor} \\
&\equiv G \cdot F \cdot H && \text{left identity of tensor} \\
&\equiv G \cdot \text{Tr}^0(F) \cdot H && \text{definition of trace}
\end{aligned}$$

Base case: $x = 1$

$$\text{Tr}^1(\text{id}_1 \otimes G \cdot F \cdot \text{id}_1 \otimes H) \equiv G \cdot \text{Tr}^1(F) \cdot H$$

Definitions

$$\begin{aligned}
 K &= \text{Tr}^1(\text{id}_1 \otimes G \cdot F \cdot \text{id}_1 \otimes H) \\
 E_K &= E_F + E_G + E_H & E[\text{id}] &= \{e_{\text{id}}\} & \Lambda_K &= \Lambda_F + \Lambda_G + \Lambda_H \\
 S_K[\bullet] &= S_G[\bullet] & S_K[e_{\text{id}}] &= [1]_s & S_K[e \in E_F] &= S_F[e] & S_K[e \in E_G] &= S_G[e] & S_K[e \in E_H] &= S_H[e] \\
 T_K[\bullet] &= T_H[\bullet] & T_K[e_{\text{id}}] &= [1]_t & T_K[e \in E_F] &= T_F[e] & T_K[e \in E_G] &= T_G[e] & T_K[e \in E_H] &= T_H[e] \\
 \kappa_K(t) &= \begin{cases} \kappa_H(\pi_{i-1}(T_H[\bullet])) & \text{if } t = \pi_0([1]_t) \wedge \kappa_F(\pi_0(T_F[\bullet])) = \pi_i(S_F[\bullet]) \\ \kappa_F(\pi_0(T_F[\bullet])) & \text{if } t = \pi_0([1]_t) \\ \pi_0([1]_s) & \text{if } \kappa_F(t) = \pi_0(S_F[\bullet]) \\ \kappa_H(\pi_{i-1}(T_H[\bullet])) & \text{if } \kappa_F(t) = \pi_i(S_F[\bullet]) \\ \pi_0([1]_s) & \text{if } \kappa_G(t) = \pi_i(S_G[\bullet]) \wedge \kappa_F(\pi_i(T_F[\bullet])) = \pi_0(S_F[\bullet]) \\ \kappa_F(\pi_{i+1}(T_F[\bullet])) & \text{if } \kappa_G(t) = \pi_i(S_G[\bullet]) \\ (\kappa_F + \kappa_G + \kappa_H)(v) & \text{otherwise} \end{cases}
 \end{aligned}$$

$$\begin{aligned}
 L &= G \cdot \text{Tr}^1(F) \cdot H \\
 E_L &= E_F + E_G + E_H & E[\text{id}] &= \{e_{\text{id}}\} & \Lambda_L &= \Lambda_F + \Lambda_G + \Lambda_H \\
 S_L[\bullet] &= S_G[\bullet] & S_L[e_{\text{id}}] &= \{\pi_0(S_F[\bullet])\} & S_L[e \in E_F] &= S_F[e] & S_L[e \in E_G] &= S_G[e] & S_L[e \in E_H] &= S_H[e] \\
 T_L[\bullet] &= T_H[\bullet] & T_L[e_{\text{id}}] &= \{\pi_0(T_F[\bullet])\} & T_L[e \in E_F] &= T_F[e] & T_L[e \in E_G] &= T_G[e] & T_L[e \in E_H] &= T_H[e] \\
 \kappa_L(t) &= \begin{cases} \kappa_H(\pi_{i-1}(T_H[\bullet])) & \text{if } \kappa_F(t) = \pi_i(S_F[\bullet]) \\ \kappa_F(\pi_{i+1}(T_F[\bullet])) & \text{if } \kappa_G(t) = \pi_i(S_G[\bullet]) \\ (\kappa_F + \kappa_G + \kappa_H)(v) & \text{otherwise} \end{cases}
 \end{aligned}$$

Equivalence maps

$$h_S(v) = \begin{cases} \pi_0(S_F[\bullet]) & \text{if } v = \pi_0([1]_s) \\ v & \text{otherwise} \end{cases} \quad h_T = \begin{cases} \pi_0(T_F[\bullet]) & \text{if } v = \pi_0([1]_t) \\ v & \text{otherwise} \end{cases} \quad h_E = \text{id}$$

Inductive case: $x = k + 1$ for $k > 1$

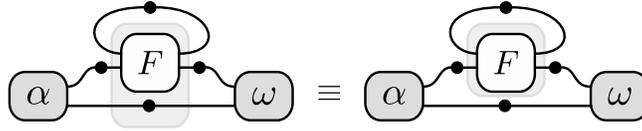
$$\begin{aligned}
 \text{Tr}^{k+1}(\text{id}_{k+1} \otimes G \cdot F \cdot \text{id}_{k+1} \otimes H) &\equiv G \cdot \text{Tr}^{k+1}(F) \cdot H \\
 \text{Tr}^{k+1}(\text{id}_k + 1 \otimes G \cdot F \cdot \text{id}_k + 1 \otimes H) &\equiv \text{Tr}^1(\text{Tr}^k(\text{id}_k + 1 \otimes G \cdot F \cdot \text{id}_k + 1 \otimes H)) && \text{definition of trace} \\
 &\equiv \text{Tr}^1(\text{Tr}^k(k \otimes \text{id}_1 \otimes G \cdot F \cdot k \otimes \text{id}_1 \otimes H)) && \text{bif of tensor I} \\
 &\equiv \text{Tr}^1(\text{id}_1 \otimes G \cdot \text{Tr}^k(F) \cdot \text{id}_1 \otimes H) && \text{IH (1)} \\
 &\equiv G \cdot \text{Tr}^1(\text{Tr}^k(F)) \cdot H && \text{base case (2)} \\
 &\equiv G \cdot \text{Tr}^{k+1}(F) \cdot H && \text{definition of trace}
 \end{aligned}$$

At (1), we use our inductive hypothesis with $m = 1 + m$ and $n = 1 + n$, since it applies for any $m, n \in \mathbb{N}$. At (2), we can use our base case since $\text{Tr}^k(F)$ has type $1 + m \rightarrow 1 + n$.

A.13 Superposing

For any interfaced linear hypergraph $F : x + m \rightarrow x + n$ and $n \in \mathbb{N}$,

$$\text{Tr}^x(F \otimes \text{id}_n) \equiv \text{Tr}^x(F) \otimes \text{id}_n.$$



This proof is by induction on x .

Zero case: $x = 0$

$$\text{Tr}^0(F \otimes \text{id}_n) \equiv \text{Tr}^0(F) \otimes \text{id}_n$$

This follows immediately by definition of trace.

Base case: $x = 1$

$$\text{Tr}^1(F \otimes \text{id}_n) \equiv \text{Tr}^1(F) \otimes \text{id}_n$$

Definitions

$$\begin{aligned} H &= \text{Tr}^x(F \otimes G) \\ E_H &= E_F + E_G & E_H[\text{id}] &= \{e_{\text{id}}\} & \Lambda_H &= \Lambda_F + \Lambda_G \\ S_H[\bullet] &= S_F[\bullet] - \pi_0(S_F[\bullet]) + [n]_s & S[e_{\text{id}}] &= \pi_0(S_F[\bullet]) & S_H[e \in E_F] &= S_F[e] \\ T_H[\bullet] &= T_F[\bullet] - \pi_0(T_F[\bullet]) + [n]_t & T[e_{\text{id}}] &= \pi_0(T_F[\bullet]) & T_H[e \in E_F] &= T_F[e] \\ \kappa_H(t) &= \begin{cases} \pi_i([n]_s) & \text{if } t = \pi_i([n]_t) \\ \kappa_F(t) & \text{otherwise} \end{cases} \end{aligned}$$

The definition of $L = \text{Tr}^x(F) \otimes G$ is identical.

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

Inductive case: $x = k + 1$ for $k > 1$

$$\text{Tr}^{k+1}(F \otimes G) \equiv \text{Tr}^{k+1}(F) \otimes G$$

$$\begin{aligned} \text{Tr}^{k+1}(F \otimes G) &\equiv \text{Tr}^1(\text{Tr}^k(F \otimes G)) && \text{definition of trace} \\ &\equiv \text{Tr}^1(\text{Tr}^k(F) \otimes G) && \text{IH (1)} \\ &\equiv \text{Tr}^1(\text{Tr}^k(F)) \otimes G && \text{base case (2)} \\ &\equiv \text{Tr}^{k+1}(F) \otimes G && \text{definition of trace} \end{aligned}$$

At (1), $\text{Tr}^1(F) : k + m \rightarrow k + n$ so we can apply our inductive hypothesis. At (2), we observe that as $F : k + 1 + m \rightarrow k + 1 + n$ then $\text{Tr}^k(F) : 1 + m \rightarrow 1 + n$, so we can use our base case. Therefore $\text{Tr}^{k+1}(F \otimes G) = \text{Tr}^{k+1}(F) \otimes G$. Therefore for any $x \in \mathbb{N}$, $\text{Tr}^x(F \otimes G) = \text{Tr}^x(F) \otimes G$.

A.14 Yanking

For any $x \in \mathbb{N}$, $\text{Tr}^x(\sigma_{x,x}) \equiv \text{id}_x$.



This proof is by induction on x .

Zero case: $x = 0$

$$\text{Tr}^0(\sigma_{0,0}) \equiv \text{id}_0$$

This follows immediately by definition of trace and symmetry.

Base case: $x = 1$

$$\text{Tr}^1(\sigma_{1,1}) \equiv \text{id}_1$$

Definitions

$$\begin{aligned} H &= \text{Tr}^1(\sigma_{1,1}) \\ E_H &= \emptyset \quad E_H[\text{id}] = \{e_{\text{id}}\} \quad \Lambda_H = \emptyset \\ S_H[\bullet] &= [1]_{s1} \quad S_H[e_{\text{id}}] = [1]_{s2} \quad T_H[\bullet] = [1]_{t1} \quad T_H[e_{\text{id}}] = [1]_{t2} \\ \kappa_H(\pi_0([1]_{t1})) &= \pi_0([1]_{s2}) \quad \kappa_H(\pi_0([1]_{t2})) = \pi_0([1]_{s1}) \end{aligned}$$

By performing a smoothing we obtain the following:

$$\begin{aligned} L &= \text{smooth}(\text{Tr}^1(\sigma_{1,1})) \\ E_L &= \emptyset \quad E_L[\text{id}] = \emptyset \quad \Lambda_L = \emptyset \\ S_L[\bullet] &= [1]_{s1} \quad T_L[\bullet] = [1]_{t1} \quad \kappa_L(\pi_0([1]_{t1})) = \pi_0([1]_{s1}) \end{aligned}$$

Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

Inductive case: $x = k + 1$ for $k > 1$

$$\text{Tr}^{k+1}(\sigma_{k+1,k+1}) \equiv \text{id}_{k+1}$$

The inductive case begins by manipulating $\text{Tr}^{k+1}(\sigma_{k+1,k+1})$ into a form with traces of only one wire.

$$\begin{aligned} \text{Tr}^{k+1}(\sigma_{k+1,k+1}) &\equiv \text{Tr}^{k+1}(k \otimes \sigma_{1,n} \otimes 1 \cdot \sigma_{k,k} \otimes \sigma_{1,1} \cdot k \otimes \sigma_{k,1} \otimes 1) && \text{inductive swap} \\ &\equiv \text{Tr}^1(\text{Tr}^k(k \otimes \sigma_{1,k} \otimes 1 \cdot \sigma_{k,k} \otimes \sigma_{1,1} \cdot k \otimes \sigma_{k,1} \otimes 1)) && \text{definition of trace} \\ &\equiv \text{Tr}^1(\sigma_{1,k} \otimes 1 \cdot \text{Tr}^k(\sigma_{k,k} \otimes \sigma_{1,1}) \cdot \sigma_{k,1} \otimes 1) && \text{tightening} \\ &\equiv \text{Tr}^1(\sigma_{1,k} \otimes 1 \cdot \text{Tr}^k(\sigma_{k,k}) \otimes \sigma_{1,1} \cdot \sigma_{k,1} \otimes 1) && \text{superposing} \\ &\equiv \text{Tr}^1(\sigma_{1,k} \otimes 1 \cdot k \otimes \sigma_{1,1} \cdot \sigma_{k,1} \otimes 1) && \text{IH} \end{aligned}$$

Definitions

$$\begin{aligned} H &= \text{Tr}^1(\sigma_{1,k} \otimes 1 \cdot k \otimes \sigma_{1,1} \cdot \sigma_{k,1} \otimes 1) \\ E_H &= \emptyset \quad E_H[\text{id}] = \{e_{\text{id}}\} \quad \Lambda = \emptyset \\ S_H[\bullet] &= [k]_s + [1]_{s1} \quad S_H[e_{\text{id}}] = [1]_{s2} \quad T_H[\bullet] = [k]_t + [1]_{t1} \quad T_H[e_{\text{id}}] = [1]_{t2} \\ \kappa(t) &= \begin{cases} \pi_0([1]_{s2}) & \text{if } t = \pi_0([1]_{t1}) \\ \pi_0([1]_{s1}) & \text{if } t = \pi_0([1]_{t2}) \\ \pi_i([k]_s) & \text{if } t = \pi_i([k]_t) \end{cases} \end{aligned}$$

By performing a smoothing we obtain the following:

$$\begin{aligned}
 L &= \text{smooth}(\text{Tr}^1(\sigma_{1,k} \otimes 1 \cdot k \otimes \sigma_{1,1} \cdot \sigma_{k,1} \otimes 1)) \\
 E_L &= \emptyset \quad E_L[\text{id}] = \emptyset \Lambda = \emptyset \\
 S_L[\bullet] &= [k]_s + [1]_{s1} \quad T_L[\bullet] = [k]_t + [1]_{t1} \quad \kappa(t) = \begin{cases} \pi_0([1]_{s1}) & \text{if } t = \pi_0([1]_{s1}) \\ \pi_i([k]_s) & \text{if } t = \pi_i([k]_t) \end{cases}
 \end{aligned}$$

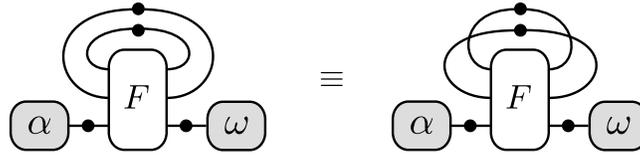
Equivalence maps

$$h_S = \text{id} \quad h_T = \text{id} \quad h_E = \text{id}$$

A.15 Exchange

For any interfaced linear hypergraph $F : x + y + m \rightarrow x + y + n$,

$$\text{Tr}^y(\text{Tr}^x(F)) \equiv \text{Tr}^x(\text{Tr}^y(\sigma_{y,x} \otimes \text{id}_m \cdot F \cdot \sigma_{x,y} \otimes \text{id}_n)).$$



This proof is by induction on x and y .

Zero case I: $x = 0, y = k$

$$\text{Tr}^k(\text{Tr}^0(F)) \equiv \text{Tr}^0(\text{Tr}^k(\sigma_{k,0} \otimes \text{id}_m \cdot F \cdot \sigma_{0,k} \otimes \text{id}_n))$$

$$\begin{aligned}
 \text{Tr}^k(\text{Tr}^0(F)) &\equiv \text{Tr}^k(F) && \text{definition of trace} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(F)) && \text{definition of trace} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(\text{id}_{k+m} \cdot F \cdot \text{id}_{k+n})) && \text{left/right identity} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(\text{id}_k \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \text{id}_n)) && \text{bifactoriality I} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(\sigma_{k,0} \otimes \text{id}_m \cdot F \cdot \sigma_{0,k} \otimes \text{id}_n)) && \text{definition of swap}
 \end{aligned}$$

Zero case II: $x = k, y = 0$

$$\text{Tr}^0(\text{Tr}^k(F)) \equiv \text{Tr}^k(\text{Tr}^0(\sigma_{0,k} \otimes \text{id}_m \cdot F \cdot \sigma_{k,0} \otimes \text{id}_n))$$

$$\begin{aligned}
 \text{Tr}^0(\text{Tr}^k(F)) &\equiv \text{Tr}^0(\text{Tr}^k(\text{id}_{k+m} \cdot F \cdot \text{id}_{k+n})) && \text{left/right identity} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(\text{id}_k \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \text{id}_n)) && \text{bifactoriality I} \\
 &\equiv \text{Tr}^0(\text{Tr}^k(\sigma_{0,k} \otimes \text{id}_m \cdot F \cdot \sigma_{k,0} \otimes \text{id}_n)) && \text{definition of swap} \\
 &\equiv \text{Tr}^k(\text{id}_k \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \text{id}_n) && \text{definition of trace} \\
 &\equiv \text{Tr}^k(\text{Tr}^0(\text{id}_k \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \text{id}_n)) && \text{definition of trace}
 \end{aligned}$$

Base case: $x = 1, y = 1$

$$\text{Tr}^1(\text{Tr}^1(F)) \equiv \text{Tr}^1(\text{Tr}^1(\sigma_{1,1} \otimes \text{id}_m \cdot F \cdot \sigma_{1,1} \otimes \text{id}_n))$$

Definitions

$$\begin{aligned}
 H &= \text{Tr}^1(\text{Tr}^1(F)) \\
 E_H &= E_F \quad E_H[\text{id}] = \{e_{\text{id}0}, e_{\text{id}1}\} \quad \Lambda_H = \Lambda_F \quad \kappa_H = \kappa_F \\
 S_H[\bullet] &= S_F[\bullet] - (\pi_0(S_F[\bullet]) + \pi_1(S_F[\bullet])) \quad S_H[e \in E_F] = S_F[e] \quad S[e_{\text{id}0}] = \{\pi_0(S_F[\bullet])\} \quad S[e_{\text{id}1}] = \{\pi_1(S_F[\bullet])\} \\
 T_H[\bullet] &= T_F[\bullet] - (\pi_0(T_F[\bullet]) + \pi_1(T_F[\bullet])) \quad T_H[e \in E_F] = T_F[e] \quad T[e_{\text{id}0}] = \{\pi_0(T_F[\bullet])\} \quad T[e_{\text{id}1}] = \{\pi_1(T_F[\bullet])\}
 \end{aligned}$$

$$\begin{aligned}
 L &= \text{Tr}^1(\text{Tr}^1(\sigma_{1,1} \otimes \text{id}_m \cdot F \cdot \sigma_{1,1} \otimes \text{id}_n)) \\
 E_L &= E_F \quad E_H[\text{id}] = \{e_{\text{id}0}, e_{\text{id}1}\} \quad \Lambda_H = \Lambda_F \\
 S_H[\bullet] &= [n] \quad S_H[e \in E_F] = S_F[e] \quad S[e_{\text{id}0}] = [1]_{s0} \quad S[e_{\text{id}1}] = [1]_{s1} \\
 T_H[\bullet] &= [m] \quad T_H[e \in E_F] = T_F[e] \quad T[e_{\text{id}0}] = [1]_{t0} \quad S[e_{\text{id}1}] = [1]_{t1} \\
 \kappa_H(t) &= \begin{cases} \kappa_F(\pi_1(T_F[\bullet])) & \text{if } t = [1]_{t0} \\ \kappa_F(\pi_0(T_F[\bullet])) & \text{if } t = [1]_{t1} \\ \kappa_F(\pi_{i+2}(T_F[\bullet])) & \text{if } t = \pi_i([m]) \\ \pi_0([1]_{s1}) & \text{if } \kappa_F(t) = \pi_0(S_F[\bullet]) \\ \pi_0([1]_{s0}) & \text{if } \kappa_F(t) = \pi_1(S_F[\bullet]) \\ \pi_{i-2}([n]) & \text{if } \kappa_F(t) = \pi_i(S_F[\bullet]) \\ \kappa_F(t) & \text{otherwise} \end{cases}
 \end{aligned}$$

Equivalence maps

$$h_S(v) = \begin{cases} \pi_0([1]_{s0}) & \text{if } v = \pi_0(S_F[\bullet]) \\ \pi_0([1]_{s1}) & \text{if } v = \pi_1(S_F[\bullet]) \\ \pi_{i-2}([n]) & \text{if } v = \pi_i(S_F[\bullet]) \\ v & \text{otherwise} \end{cases} \quad h_T(v) = \begin{cases} \pi_0([1]_{t0}) & \text{if } v = \pi_0(T_F[\bullet]) \\ \pi_0([1]_{t1}) & \text{if } v = \pi_1(T_F[\bullet]) \\ \pi_{i-2}([n]) & \text{if } v = \pi_i(T_F[\bullet]) \\ v & \text{otherwise} \end{cases} \quad h_E = \text{id}$$

Inductive case I: $x = k + 1, y = 1$

$$\text{Tr}^1(\text{Tr}^{k+1}(F)) \equiv \text{Tr}^{k+1}(\text{Tr}^1(\sigma_{1,k+1} \otimes \text{id}_m \cdot \sigma_{k+1,1} \otimes \text{id}_m))$$

$$\begin{aligned}
\text{Tr}^1(\text{Tr}^{k+1}(F)) &\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^k(F))) \\
&\quad \text{definition of trace} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\sigma_{1,1} \otimes \text{id}_m \cdot \text{Tr}^k(F) \cdot \sigma_{1,1} \otimes \text{id}_n)) \\
&\quad \text{base case} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^k(\text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_n))) \\
&\quad \text{tightening} \\
&\equiv \text{Tr}^1(\text{Tr}^k(\text{Tr}^1(\sigma_{1,k} \otimes \text{id}_1 \otimes \text{id}_m \cdot \text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_n \cdot \sigma_{k,1} \otimes \text{id}_1 \otimes \text{id}_n))) \\
&\quad \text{IH} \\
&\equiv \text{Tr}^{k+1}(\text{Tr}^1(\sigma_{1,k} \otimes \text{id}_1 \otimes \text{id}_m \cdot \text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,1} \otimes \text{id}_n \cdot \sigma_{k,1} \otimes \text{id}_1 \otimes \text{id}_n)) \\
&\quad \text{definition of trace} \\
&\equiv \text{Tr}^{k+1}(\text{Tr}^1((\sigma_{1,k} \otimes \text{id}_1 \cdot \text{id}_k \otimes \sigma_{1,1}) \otimes \text{id}_m \cdot F \cdot (\text{id}_k \otimes \sigma_{1,1} \cdot \sigma_{k,1} \otimes \text{id}_1) \otimes \text{id}_n)) \\
&\quad \text{bifactoriality II} \\
&\equiv \text{Tr}^{k+1}(\text{Tr}^1(\sigma_{1,k+1} \otimes \text{id}_m \cdot F \cdot \sigma_{k+1,1} \otimes \text{id}_n)) \\
&\quad \text{definition of swap}
\end{aligned}$$

Inductive case II: $x = 1, y = k' + 1$

$$\text{Tr}^{k'+1}(\text{Tr}^1(F)) \equiv \text{Tr}^1(\text{Tr}^{k'+1}(\sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \sigma_{1,k'+1} \otimes \text{id}_n))$$

$$\begin{aligned}
\text{Tr}^{k'+1}(\text{Tr}^1(F)) &\equiv \text{Tr}^1(\text{Tr}^{k'}(\text{Tr}^1(F))) \\
&\quad \text{definition of trace} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^{k'}(\sigma_{k',1} \otimes \text{id}_1 \otimes \text{id}_m \cdot F \cdot \sigma_{1,k'} \otimes \text{id}_1 \otimes \text{id}_n))) \\
&\quad \text{IH} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\sigma_{1,1} \otimes \text{id}_m \cdot \text{Tr}^{k'}(\sigma_{k',1} \otimes \text{id}_1 \otimes \text{id}_m \cdot F \cdot \sigma_{1,k'} \otimes \text{id}_1 \otimes \text{id}_n) \cdot \sigma_{1,1} \otimes \text{id}_n)) \\
&\quad \text{Base case} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^{k'}(\text{id}'_k \otimes \sigma_{1,1} \otimes \text{id}_m \cdot \sigma_{k',1} \otimes \text{id}_1 \otimes \text{id}_m \cdot F \cdot \sigma_{1,k'} \otimes \text{id}_1 \otimes \text{id}_n \cdot \text{id}'_k \otimes \sigma_{1,1} \otimes \text{id}_n))) \\
&\quad \text{tightening} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^{k'}((\text{id}'_k \otimes \sigma_{1,1} \cdot \sigma_{k',1} \otimes \text{id}_1) \otimes \text{id}_m \cdot F \cdot (\sigma_{1,k'} \otimes \text{id}_1 \cdot \text{id}'_k \otimes \sigma_{1,1}) \otimes \text{id}_n))) \\
&\quad \text{bifactoriality II} \\
&\equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^{k'}(\sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \sigma_{1,k'+1} \otimes \text{id}_n))) \\
&\quad \text{definition of swap}
\end{aligned}$$

Inductive case III: $x = k + 1, y = k' + 1$

$$\text{Tr}^{k'+1}(\text{Tr}^{k+1}(F)) \equiv \text{Tr}^{k+1}(\text{Tr}^{k'+1}(\sigma_{k'+1,k+1} \otimes \text{id}_m \cdot F \cdot \sigma_{k+1,k'+1} \otimes \text{id}_n))$$

$$\begin{aligned}
& \text{Tr}^{k'+1}(\text{Tr}^{k+1}(F)) \\
& \equiv \text{Tr}^{k'+1}(\text{Tr}^1(\text{Tr}^k(F))) \\
& \quad \text{definition of trace} \\
& \equiv \text{Tr}^1(\text{Tr}^{k'+1}(\sigma_{k'+1,1} \otimes \text{id}_m \cdot \text{Tr}^k(F) \cdot \sigma_{1,k'+1} \otimes \text{id}_n)) \\
& \quad \text{inductive case II} \\
& \equiv \text{Tr}^1(\text{Tr}^{k'+1}(\text{Tr}^k(\text{id}_k \otimes \sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,k'+1} \otimes \text{id}_n))) \\
& \quad \text{tightening} \\
& \equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^{k'}(\text{Tr}^k(\text{id}_k \otimes \sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,k'+1} \otimes \text{id}_n)))) \\
& \quad \text{definition of trace} \\
& \equiv \text{Tr}^1(\text{Tr}^1(\text{Tr}^k(\text{Tr}^{k'}(\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,k'+1} \otimes \text{id}_n \cdot \sigma_{k,k'} \otimes \text{id}_{1+n})))) \\
& \quad \text{IH} \\
& \equiv \text{Tr}^1(\text{Tr}^{k+1}(\text{Tr}^{k'}(\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1} \otimes \text{id}_m \cdot F \cdot \text{id}_k \otimes \sigma_{1,k'+1} \otimes \text{id}_n \cdot \sigma_{k,k'} \otimes \text{id}_{1+n}))) \\
& \quad \text{definition of trace} \\
& \equiv \text{Tr}^1(\text{Tr}^{k+1}(\text{Tr}^{k'}((\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1}) \otimes \text{id}_m \cdot F \cdot (\text{id}_k \otimes \sigma_{1,k'+1} \cdot \sigma_{k,k'} \otimes \text{id}_{1+m}) \otimes \text{id}_n))) \\
& \quad \text{bifunctoriality I/II} \\
& \equiv \text{Tr}^{k+1}(\text{Tr}^1(\sigma_{1,k+1} \otimes \text{id}_m \cdot \text{Tr}^{k'}((\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1}) \otimes \text{id}_m \cdot F \cdot (\text{id}_k \otimes \sigma_{1,k'+1} \cdot \sigma_{k,k'} \otimes \text{id}_{1+m}) \otimes \text{id}_n) \cdot \sigma_{k+1,1} \otimes \text{id}_m)) \\
& \quad \text{inductive case I} \\
& \equiv \text{Tr}^{k+1}(\text{Tr}^1(\text{Tr}^{k'}(\text{id}_{k'} \otimes \sigma_{1,k+1} \otimes \text{id}_m \cdot (\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1}) \otimes \text{id}_m \cdot F \cdot \\
& \quad \text{tightening} \quad (\text{id}_k \otimes \sigma_{1,k'+1} \cdot \sigma_{k,k'} \otimes \text{id}_{1+m}) \otimes \text{id}_n \cdot \text{id}_{k'} \otimes \sigma_{k+1,1} \otimes \text{id}_m))) \\
& \equiv \text{Tr}^{k+1}(\text{Tr}^{k'+1}(\text{id}_{k'} \otimes \sigma_{1,k+1} \otimes \text{id}_m \cdot (\sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1}) \otimes \text{id}_m \cdot F \cdot \\
& \quad \text{definition of trace} \quad (\text{id}_k \otimes \sigma_{1,k'+1} \cdot \sigma_{k,k'} \otimes \text{id}_{1+m}) \otimes \text{id}_n \cdot \text{id}_{k'} \otimes \sigma_{k+1,1} \otimes \text{id}_m)) \\
& \equiv \text{Tr}^{k+1}(\text{Tr}^{k'+1}((\text{id}_{k'} \otimes \sigma_{1,k+1} \cdot \sigma_{k',k} \otimes \text{id}_{1+m} \cdot \text{id}_k \otimes \sigma_{k'+1,1}) \otimes \text{id}_m \cdot F \cdot (\text{id}_k \otimes \sigma_{1,k'+1} \cdot \sigma_{k,k'} \otimes \text{id}_{1+m} \cdot \text{id}_{k'} \otimes \sigma_{k+1,1}) \otimes \text{id}_n)) \\
& \quad \text{bifunctoriality II} \\
& \equiv \text{Tr}^{k+1}(\text{Tr}^{k'+1}(\sigma_{k'+1,k+1} \otimes \text{id}_m \cdot F \cdot \sigma_{k+1,k'+1} \otimes \text{id}_n)) \\
& \quad \text{definition of swap}
\end{aligned}$$