A First-order Logic for String Diagrams

Aleks Kissinger and David Quick

Department of Computer Science
University of Oxford, UK
{aleks.kissinger,david.quick}@cs.ox.ac.uk

Abstract

Equational reasoning with string diagrams provides an intuitive means of proving equations between morphisms in a symmetric monoidal category. This can be extended to proofs of infinite families of equations using a simple graphical syntax called !-box notation. While this does greatly increase the proving power of string diagrams, previous attempts to go beyond equational reasoning have been largely ad hoc, owing to the lack of a suitable logical framework for diagrammatic proofs involving !-boxes. In this paper, we extend equational reasoning with !-boxes to a fully-fledged first order logic with conjunction, implication, and universal quantification over !-boxes. This logic, called !L, is then rich enough to properly formalise an induction principle for !-boxes. We then build a standard model for !L and give an example proof of a theorem for non-commutative bialgebras using !L, which is unobtainable by equational reasoning alone.

1998 ACM Subject Classification D.3.1 Formal Definitions and Theory, F.4.1 Mathematical Logic

Keywords and phrases string diagrams, compact closed monoidal categories, abstract tensor systems, first-order logic

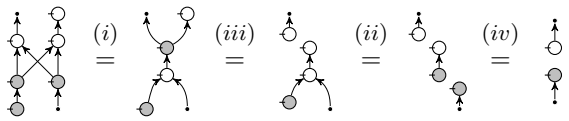
Digital Object Identifier 10.4230/LIPIcs.CALCO.2015.171

1 Introduction

Many processes come with natural notions of parallel and sequential composition. In such cases, it is advantageous to switch from traditional term-based (i.e. one-dimensional) syntax to the two-dimensional syntax of string diagrams. This diagrams, which consist of boxes (or various other shapes) connected by wires, form a sound and complete language for compositions of morphisms in a monoidal category [8]. Recently, the use of string diagrams has gained much interest in a wide variety of areas, including categorical quantum mechanics [4, 3, 5], computational linguistics [9] and control theory [2, 1].

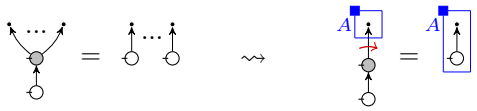
What many of these applications have in common is they make extensive use of equational reasoning for string diagrams. That is, proofs are constructed by starting with a fixed set of diagram equations, e.g.

and using those to construct new equations by substitution of sub-diagrams. For example, the following is a derivation making use of the four rules above:



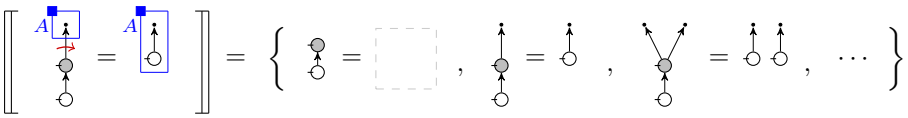
6th International Conference on Algebra and Coalgebra in Computer Science (CALCO'15).
Editors: Lawrence S. Moss and Pawel Sobocinski; pp. 171–189
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

However, to prove more powerful theorems, one often needs to pass from statements about single diagrams to entire families of diagrams and diagram equations. One way to do this, while staying within the realm of string diagrams is to use !-box notation (pronounced 'bang-box notation'), introduced in [6] and formalised in [11]. In this notation, certain sub-diagrams are wrapped in boxes, which mean 'repeat this sub-diagram any number of times'. For example, suppose we considered a family of 'copy' operations with 1 input and n outputs. Then, if we had some other map with just a single output, we might ask that connecting it to the n-fold 'copy' results in n copies. We can represent this family of rules using !-box notation as follows:



(1)

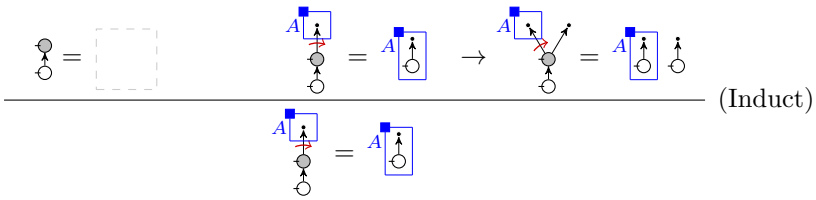
Whereas the equation on the left is informal, the expression on the right defines a family of equations without ambiguity. Formally, a !-box rule represents a set of string diagram rules obtained by *instantiating* the !-box, which essentially amounts fixing the number of times to copy each !-box. For example, the instances of the !-box rule above are precisely the ones we meant to capture with the informal expression:



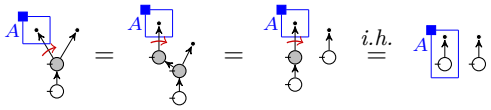
where the 'blank space' in the first equation represents the monoidal unit. We can even use this more expressive notation to make recursive definitions. For instance, we could recursively define the n-fold copy operation as a tree of binary copy operations:

(2)

Using just equational reasoning, there is no way to get from the equations in (2) to the n-fold copy equation (1). However, if we introduce an induction principle:

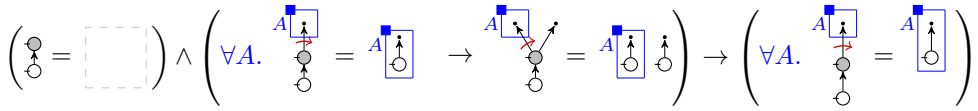


we can split into a base case (zero copies of the !-box) and a step case (n copies implies $n+1$ copies). Taking the base case as given, we can prove the step case using the induction hypothesis and the rules in (2):



Unfortunately, this doesn't quite work. If we interpret $\rightarrow$ to mean 'the rule on the left can be used in the proof of the rule on the right', the step case is vacuous. The rule on the right is *already* an instance of the rule on the left. This is a bit like saying: $(\forall n.Pn) \rightarrow (\forall n.P(n+1))$, which is of course true for any P.

The problem is, when we pass to !-box notation, where single diagram rules now represent whole families of rules, our existing reasoning tools do not provide enough control over instances of rules, and how those instances interact with each other. This problem was solved for the specific case of induction in [15] using an operation called *fixing*, which essentially freezes a !-box so it can't be instantiated. However, this was introduced more as a stopgap, until a proper logic could be developed, suitable for handling conjunction, implication, and crucially universal quantification over !-boxes. In this paper, we develop that logic. With this new *!-logic* in hand, we can correct our failed attempt at induction to:



In addition to giving a solid foundation for proofs constructed using !-boxes, a major motivating factor for the development of a formal logic of !-boxes is its implementation in the proof assistant Quantomatic [14]. Currently, Quantomatic supports pure equational reasoning on string diagrams with !-boxes. The implementation of !-logic will allow it to support diagrammatic versions of all the usual trappings of a fully-featured proof assistant, such as local assumptions, goal-driven (i.e. backward) reasoning, and of course inductive proofs.

There are two essentially equivalent ways to formalise string diagrams with !-boxes: one combinatoric (as in the original formulation) and one syntactic, building on the *tensor notation* for compact closed categories. Here we opt for the latter, as it more conveniently fits into the presentation of the logic and provides a means of elegantly representing commutative *and non-commutative* generators. Equational reasoning using *!-tensor* notation was presented in [12] allowing some basic rules which were shown to be sound in the extended version [13]. In this paper we begin by reviewing compact closed categories, tensor notation, and !-tensors in Section 2. Next, we define the concept of an instantiation, which will play a central role in the logic in Section 3. We introduce the syntax of our logic, namely *!-formulas*, in Section 4 and give the rules of the logic in Section 5. We provide a semantics for !-formulas based on sets of instantiations evaluated in a compact closed category $\mathcal{C}$ in Section 6. We conclude by exhibiting a non-trivial proof involving non-commutative bialgebras, which can be done entirely within !L and diagram rewriting.

2 Preliminaries

2.1 Compact closed categories and signatures

Throughout this paper, we will work with *compact closed categories*, i.e. symmetric monoidal categories where every object X has a *dual* object X^* and two morphisms $\eta_X : I \to X^* \otimes X$, $\epsilon_X : X \otimes X^* \to I$ satisfying the *yanking equations*:

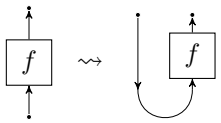
$$(\epsilon_X \otimes 1_X) \circ (1_X \otimes \eta_X) = 1_X \qquad (1_{X^*} \otimes \epsilon_X) \circ (\eta_X \otimes 1_{X^*}) = 1_{X^*}$$

For simplicity, we will focus on *strict* compact closed categories, where associativity and unitality of $\otimes$ hold on-the-nose. However, all of the concepts we will use in this paper go through virtually unmodified by Mac Lane's coherence theorem.

As string diagrams, we will depict X as a wire directed upwards, and X^* as a wire directed downwards. Thus η_X and ϵ_X can be depicted as half-turns:

which we typically call 'cups' and 'caps', respectively. Using this notation, the yanking equations resemble their namesake:

One consequence of the inclusion of cups and caps is that we can now introduce 'feedback loops', allowing us to make sense of arbitrary string diagrams, not just directed acyclic ones. A second consequence is that any map $f : X \to Y$ can be equivalently represented as a map of the form $\widetilde{f} : I \to X^* \otimes Y$ just by 'bending' the input up to be an output:



Thus, we will always assume that our generating morphisms can be written in the form $\phi : I \to X_1 \otimes X_2 \otimes \ldots \otimes X_n$ for objects $X_1, X_2, \ldots, X_n$. A morphism whose domain is the monoidal unit is called a *point*.

▶ **Definition 1.** A *compact closed signature* Σ consists of a set $\mathcal{O} := \{x, y, \ldots\}$ and a set $\mathcal{M}$ of pairs (ψ, w), where w is a word in $\{x, x^*, y, y^*, \ldots\}$. If ψ occurs precisely once in $\mathcal{M}$, it is said to have *fixed arity*, otherwise it has *variable arity*.

For simplicity, we will assume every generator in Σ is defined for every arity. This can be avoided if we add extra conditions to Definition 6 to ensure we never get 'undefined' generators, but for our purposes, this won't be necessary.

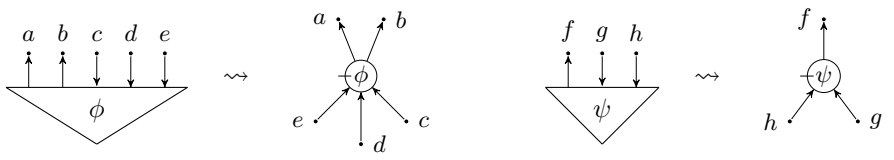
▶ **Definition 2.** For a compact closed category $\mathcal{C}$, a *valuation* $[\![-]\!] : \Sigma \to \mathcal{C}$ is a choice of object $X \in \mathrm{ob}(\mathcal{C})$ for every $x \in \mathcal{O}$, and a choice of point $[\![\psi]\!] : I \to X_1 \otimes X_2^* \otimes \ldots \otimes X_n$ for every $(\psi, x_1 x_2^* \ldots x_n) \in \mathcal{M}$.

When there can be no confusion, we write pairs $(\psi, x_1 x_2^* \ldots x_n)$ also as $\psi : I \to X_1 \otimes X_2^* \otimes \ldots X_n$. As usual , the free compact closed category $\mathrm{Free}(\Sigma)$ is characterised by the universal property that any valuation lifts uniquely to functor $[\![-]\!] : \mathrm{Free}(\Sigma) \to \mathcal{C}$ preserving all of the compact closed structure [12]. In the next section, we will give a convenient syntactic presentation of this category.

2.2 Tensor notation for compact closed categories

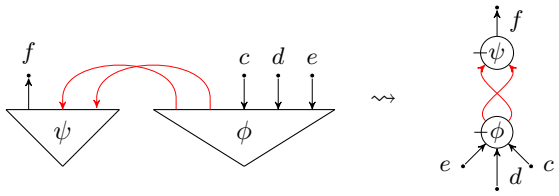
From now on, we will assume that Σ only has one object X, so morphisms will be maps from I to monoidal products of X and X^*.

Suppose that we have two generators in Σ, $\phi : I \to X \otimes X \otimes X^* \otimes X^* \otimes X^*$ and $\psi : I \to X \otimes X^* \otimes X^*$. Diagrammatically we will depict these generators as circular nodes with the edges ordered clockwise around the node. To avoid ambiguity we place a tick on the node between the last and first edge. We will name free edges so they can be referred to when manipulating diagrams. Hence the generators in our example (with arbitrarily named edges) are:



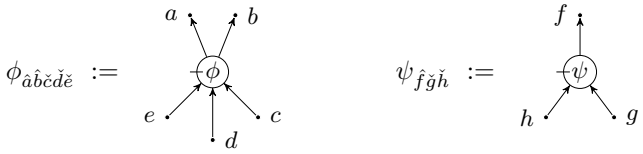
(3)

Now, wires connecting these dots indicate the presence of caps:

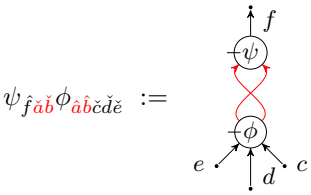


(4)

To succinctly express these kinds of string diagrams syntactically, we can use *tensor notation*. Here, we represent generators by writing their names, followed by a list of subscripts indicating their (named) inputs and outputs:



Inputs (i.e. outputs of type X^*) are represented as names with 'checks' $\check{a}, \check{b}, \ldots$, whereas outputs are represented as names with 'hats' $\hat{a}, \hat{b}, \ldots$. We combine generators into a single diagram by concatenating them, and the process of connecting generators together by caps—which we call *contraction*—is indicated by repeating names:



(5)

If a name occurs once, it is called a *free edgename*. If it is repeated, it is called a *bound edgename*. As the name would suggest, bound edgenames have no meaning in their own right, and can be changed (a.k.a. α-converted) at will. Hence the expressions $\psi_{\hat{f}\check{a}\check{b}}\phi_{\hat{a}\hat{b}\check{c}\check{d}\check{e}}$ and $\phi_{\hat{g}\hat{h}\check{c}\check{d}\check{e}}\psi_{\hat{f}\check{g}\check{h}}$ both represent (5). Also, since it is the names that indicate inputs/outputs of a tensor expression, the order in which we write tensor symbols is irrelevant. So, for example, $\psi_{\hat{f}\check{a}\check{b}}\phi_{\hat{a}\hat{b}\check{c}\check{d}\check{e}} = \phi_{\hat{a}\hat{b}\check{c}\check{d}\check{e}}\psi_{\hat{f}\check{a}\check{b}}$.

This notation gives a simple presentation of string diagrams, and hence of morphisms in the free compact closed category over Σ. The only mismatch between tensors and morphisms in the free category is that tensors use *names* to identify inputs/outputs, whereas categories use *positions*. Thus, to relate the two concepts, we assume the set of edgenames contains two disjoint sets $\{a_1, a_2, \ldots\}$ and $\{b_1, b_2, \ldots\}$ that are totally ordered and (countably) infinite, and introduce the notion of *canonically named tensors*.

▶ **Definition 3.** A tensor is *canonically named* if its free names are $a_1, \ldots a_m, b_1, \ldots, b_n$ for some $m, n \geq 0$.

We can then express a morphism in Free(Σ) as a tensor whose i-th input is named a_i and whose j-th output is named b_j. It was shown in [10] (for the traced case) and [13] (for the compact closed case) that Free(Σ) is equivalent to the category whose morphisms are canonically-named tensors, with $\circ$ and $\otimes$ defined in the obvious way using renaming and contraction. This gives us an important consequence:

▶ **Theorem 4.** *For any compact closed signature Σ, a valuation $[\![-]\!] : \Sigma \to \mathcal{C}$ lifts uniquely to an operation which sends canonically named tensors G over Σ to morphisms $[\![G]\!]$ in $\mathcal{C}$.*

2.3 !-tensors

As mentioned in the intro, a string diagram with !-boxes represents a family of string diagrams, where the sub-diagram in the !-box has been copied an arbitrary number of times. To formalise this, we extend the tensor syntax to include !-boxes. These extended expressions are called !-tensors. Fix disjoint, infinite sets $\mathcal{E}$ and $\mathcal{B}$ of *edgenames* and *boxnames*, respectively.

▶ **Definition 5.** The set of *edgeterms* $\mathcal{T}_e$ is defined inductively as follows:

- $\epsilon \in \mathcal{T}_e$ (empty edgeterm)
- $\check{a}, \hat{a} \in \mathcal{T}_e$ $\quad a \in \mathcal{E}$
- $\langle e]^A, [e\rangle^A \in \mathcal{T}_e$ $\quad e \in \mathcal{T}_e,\ A \in \mathcal{B}$
- $ef \in \mathcal{T}_e$ $\quad e, f \in \mathcal{T}_e$

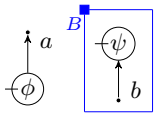
Letting 1 represent the empty !-tensor and $1_{\hat{a}\check{b}}$ represent an identity edge with input named b and output named a, we can define !-tensor expressions as follows:

▶ **Definition 6.** The set of all !-tensor expressions $\mathcal{T}_\Sigma$ for a signature Σ is defined inductively as:

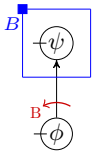
- $1, 1_{\hat{a}\check{b}} \in \mathcal{T}_\Sigma$ $\quad a, b \in \mathcal{E}$
- $\phi_e \in \mathcal{T}_\Sigma$ $\quad e \in \mathcal{T}_e, \phi \in \Sigma$
- $[G]^A \in \mathcal{T}_\Sigma$ $\quad G \in \mathcal{T}_\Sigma,\ A \in \mathcal{B}$
- $GH \in \mathcal{T}_\Sigma$ $\quad G, H \in \mathcal{T}_\Sigma$

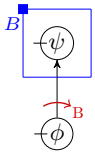
Subject to the conditions that (F1) $\check{a}$ and $\hat{a}$ must occur at most once for each edgename a and (F2) $[\ldots]^A$ must occur at most once for each boxname A, as well as consistency conditons (C1)–(C3) for !-boxes given in [13].

We omit the formal statement of (C1)-(C3) here, as they are easiest to understand in the graphical presentation of !-tensors. Sub-expressions of the form $[\ldots]^A$ are represented by wrapping a box around part of the string diagram:

$\phi_{\hat{a}}[\psi_{\check{b}}]^B :=$ 

Edges connecting into or out of a !-box must be annotated with the !-box name and a direction, indicating whether the new edgenames should be produced to the left (anticlockwise) or to the right (clockwise) when a !-box is expanded. We indicate this direction by drawing an arc over the annotated edges:

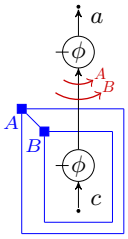
$\phi_{\langle\hat{a}]^B}[\psi_{\check{a}}]^B :=$ 

vs. $\phi_{[\hat{a}\rangle^B}[\psi_{\check{a}}]^B :=$ 

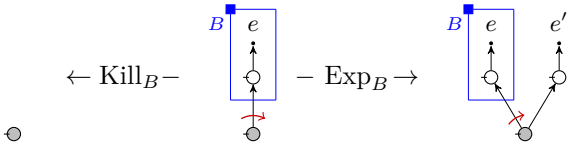
We drop the label on the arc when it can be inferred from context. The remaining consistency conditions say that any edge connecting into or out of a !-box must have an annotation, and

those annotations should respect nesting of !-boxes, as in e.g.:

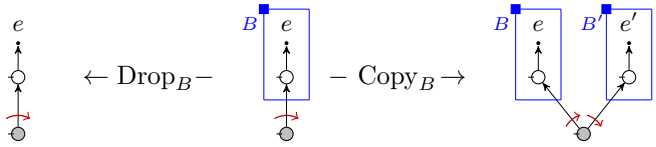
$\phi_{\hat{a}\langle\check{b}]^B]^A}[[\phi_{\hat{b}\check{c}}]^B]^A :=$



For a fully rigorous account of these conditions, see [13]. However, the above description should suffice for the purposes of this paper, so we'll proceed to how !-tensors are instantiated. The primary instantiation operations are *expand*, which produces a new copy of the contents of a !-box and *kill*, which removes the !-box from the diagram:



These two operations suffice to produce all *concrete instances*, that is all instances not involving any !-boxes, of a !-tensor. If we wish to get *all* instances of a !-tensor, including those with !-boxes, we factorise expand into two additional operations: *copy*, which makes a copy of the !-box and its contents, and *drop*, which removes a !-box and leaves its contents behind:



We can define all four of these operations recursively on !-tensor expressions. We first give the recursive cases where all four operations behave the same:

$$\begin{aligned}
\mathrm{Op}_B(GH) &:= \mathrm{Op}_B(G)\,\mathrm{Op}_B(H) & \mathrm{Op}_B(ef) &:= \mathrm{Op}_B(e)\,\mathrm{Op}_B(f)\\
\mathrm{Op}_B([G]^A) &:= [\mathrm{Op}_B(G)]^A & \mathrm{Op}_B([e\rangle^A) &:= [\mathrm{Op}_B(e)\rangle^A\\
\mathrm{Op}_B(\phi_e) &:= \phi_{\mathrm{Op}_B(e)} & \mathrm{Op}_B(\langle e]^A) &:= \langle \mathrm{Op}_B(e)]^A\\
\mathrm{Op}_B(x) &:= x
\end{aligned}$$

where $A \neq B$ and $x \in \{1, 1_{\hat{a}\check{b}}, \check{a}, \hat{a}, \epsilon\}$. The four operations are distinguished on the remaining three cases:

$$\begin{aligned}
\mathrm{Exp}_B([G]^B) &:= [G]^B\,\mathbf{fr}(G) & \mathrm{Kill}_B([G]^B) &:= 1\\
\mathrm{Exp}_B([e\rangle^B) &:= [e\rangle^B\,\mathbf{fr}(e) & \mathrm{Kill}_B([e\rangle^B) &:= \epsilon\\
\mathrm{Exp}_B(\langle e]^B) &:= \mathbf{fr}(e)\langle e]^B & \mathrm{Kill}_B(\langle e]^B) &:= \epsilon
\end{aligned}$$

$$\begin{aligned}
\mathrm{Copy}_B([G]^B) &:= [G]^B[\mathbf{fr}(G)]^{\mathbf{fr}(B)} & \mathrm{Drop}_B([G]^B) &:= G\\
\mathrm{Copy}_B([e\rangle^B) &:= [e\rangle^B[\mathbf{fr}(e)\rangle^{\mathbf{fr}(B)} & \mathrm{Drop}_B([e\rangle^B) &:= e\\
\mathrm{Copy}_B(\langle e]^B) &:= \langle \mathbf{fr}(e)]^{\mathbf{fr}(B)}\langle e]^B & \mathrm{Drop}_B(\langle e]^B) &:= e
\end{aligned}$$

Where **fr** is a function assigning fresh names to all edges and !-boxes in an expression. We occasionally write $\mathrm{Exp}_{B,\mathbf{fr}}$ and $\mathrm{Copy}_{B,\mathbf{fr}}$ to explicitly reference the freshness function of a !-box operation. Note that if B is not contained in G, each of these operations will leave G unchanged.

3 Compatibility and instantiations of !-boxes

In Section 4, we will define the formulas of !-logic. It only makes sense to combine !-tensors into single formulas if their !-boxes are compatible in some sense, so we first provide some basic notions relating to compatibility.

▶ **Definition 7.** If F is a set and $\prec$ is a binary relation on F then the pair $(F, \prec)$ is called a *forest* if it forms a cycle-free directed graph where each node A has at most one node B s.t $A \prec B$. A forest can also be seen as a graph made up of disconnected directed trees. We write $<$ for the transitive closure and $\leq$ for the reflexive and transitive closure of $\prec$.

Let $\downarrow X$ and $\uparrow X$ be the downward and upward closure of $X \subseteq F$, respectively. For a single element $A \in F$, we write $\downarrow A$ for $\downarrow\{A\}$.

▶ **Definition 8.** If a subset $X \subseteq F$ is both upward and downward closed (i.e. $X = \downarrow X = \uparrow X$) then we say X is a *component* of $(F, \prec)$. If it contains no proper sub-components, it is called a *connected component*.

We write $F^{\top} \subseteq F$ for the set of maximal elements with respect to $\leq$. Note that for $A \in F^{\top}$ the set $\downarrow A$ is always a connected component, and for F finite, all connected components are of this form.

▶ **Definition 9.** Two forests F, F' are said to be *compatible*, written $F \triangle F'$, if the intersection $F \cap F'$ is a (possibly empty) component of both F and F'.

Equivalently, F, F' are compatible if and only if there exist forests X, Y, Z such that $F = X \uplus Y$ and $G = Y \uplus Z$. As a consequence, the union of compatible forests is always well-defined ($F \cup F' := X \uplus Y \uplus Z$), and itself a forest. For any !-tensor, we can always associate a forest of !-boxes:

▶ **Definition 10.** For a !-tensor G, let $(\mathrm{Boxes}(G), \prec_G)$ be the forest of !-boxes in G, where $A \prec_G B$ iff A is a direct descendent of B. That is, A is nested inside of B with no intervening !-boxes.

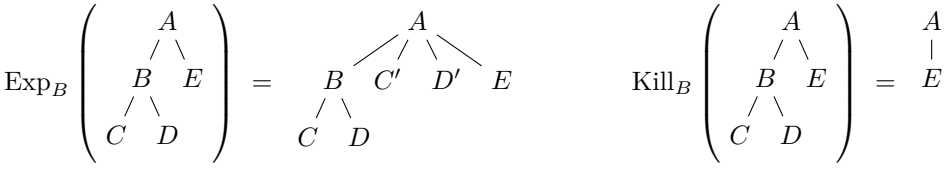
An important concept for !-tensors is that of *instantiations*. These capture precisely the sequence of operations by which a !-tensor is transformed into some instance of itself. For a !-tensor G, an *instantiation* i of G is a sequence of zero or more Exp and Kill operations such that $i(G)$ doesn't contain any !-boxes.

In fact, we can divorce the notion of instantiation from a particular !-tensor if we notice that instantiations make sense for any forest. For a forest F, define the Exp_B and Kill_B operations as identity maps if $B \notin F$ and else as follows:

$$\mathrm{Exp}_B(F) := F \cup \mathbf{fr}(\downarrow B \setminus B) \qquad \mathrm{Kill}_B(F) := F \setminus \downarrow B$$

where the top elements of $\mathbf{fr}(\downarrow B \setminus B)$ are added as descendants of the parent of B (if it has one). So, Kill_B removes B and all of its children, whereas Exp_B behaves just like expanding

a !-box, in that it adds a fresh copy of all of the children as siblings:



We can now define instantiations in a way that only refers to forests:

▶ **Definition 11.** For a forest F, an *instantiation* of F is a composition i of zero or more operations Exp_B, Kill_B such that B is in the domain of each operation and $i(F) = \{\}$. Let $\mathrm{Inst}(F)$ be the set of all instantiations of F.

In particular, if F is empty, $\mathrm{Inst}(F)$ only contains the trivial instantiation 1. Note that for *any* instantiation $i \in \mathrm{Inst}(F)$ where $F \triangle \mathrm{Boxes}(G)$, $i(G)$ is a well defined !-tensor. This added flexibility will be important to the interpretation of !-logic formulas, where instantiations may act on many !-tensors simultaneously.

4 !-logic formulas

In this section, we will introduce the syntax of !-logic. The atomic !-logic formulas are well-formed equations between !-tensors and generic formulas are built up from the atomic formulas using conjunction, implication, and universal quantification. There does not appear to be any obstacle to adding negation (and hence existential quantification) to !-logic formulae but no application has currently been found by the authors.

Well-formed !-tensor equations are pairs of !-tensors with the property that any simultaneous instantiation of the LHS and RHS produces a valid equation between tensors. That is, the LHS and the RHS of any instance of the equation should have identical free edgenames for their inputs and outputs.

▶ **Definition 12.** A !-tensor equation $G = H$ is *well-formed* if G and H have identical inputs and outputs, $\mathrm{Boxes}(G) \triangle \mathrm{Boxes}(H)$, and an input $\check{a}$ (resp. output $\hat{a}$) occurs in a !-box A in G iff it occurs in the same !-box in H.

Note that by '$\check{a}$ occurs in A' we mean $\check{a}$ occurs as a sub-expression of $[\ldots]^A$, $\langle\ldots]^A$ or $[\ldots\rangle^A$. Also note that we only require the !-boxes on the LHS and RHS be compatible, rather than identical. This is unproblematic, since as mentioned at the end of Section 2.3, operations on !-boxes that are missing from the LHS or the RHS will simply be ignored.

The other formulas are built inductively, while maintaining the property that the subformulas have compatible !-boxes. To accomplish this, it is most convenient to define the set of !-formulas while simultaneously defining the operation $\mathrm{Boxes}(X)$ for any !-formula X.

▶ **Definition 13.** The set of *!-formulas*, $\mathcal{F}_\Sigma$, for a signature Σ is defined inductively as:

- $G = H \in \mathcal{F}_\Sigma$ — $G, H \in \mathcal{T}_\Sigma,\ G = H$ well-formed
- $X \wedge Y \in \mathcal{F}_\Sigma$ — $X, Y \in \mathcal{F}_\Sigma,\ \mathrm{Boxes}(X) \triangle \mathrm{Boxes}(Y)$
- $X \to Y \in \mathcal{F}_\Sigma$ — $X, Y \in \mathcal{F}_\Sigma,\ \mathrm{Boxes}(X) \triangle \mathrm{Boxes}(Y)$
- $\forall A.\ X \in \mathcal{F}_\Sigma$ — $X \in \mathcal{F}_\Sigma,\ A \in \mathrm{Boxes}(X)^\top$

where $\mathrm{Boxes}(-)$ is defined recursively on !-formulas by:

- $\mathrm{Boxes}(G = H) := \mathrm{Boxes}(G) \cup \mathrm{Boxes}(H)$
- $\mathrm{Boxes}(X \wedge Y) := \mathrm{Boxes}(X) \cup \mathrm{Boxes}(Y)$
- $\mathrm{Boxes}(X \to Y) := \mathrm{Boxes}(X) \cup \mathrm{Boxes}(Y)$
- $\mathrm{Boxes}(\forall A.\ X) := \mathrm{Boxes}(X) \setminus \downarrow A$

Just like one can read formulas in predicate logic as mappings from values of the free variables to truth values, one should read !-formulas as mappings from *instantiations of !-boxes* to truth values. Thus, universal quantification over !-boxes states that a particular formula holds for *all instantiations* involving those !-boxes. We will make this interpretation precise in Section 6.

One important thing to note is that universal quantification over a top-level !-box A should be interpreted as quantifying over the entire *connected component* $\downarrow A$. In the absence of nesting, this is the same as quantifying over individual !-boxes. However, in the presence of nesting, this restriction to only quantifying over entire components seems to be necessary for giving a consistent interpretation to !-logic formulas. This boils down to the fact that !-box operations on separate components of $\mathrm{Boxes}(X)$ commute, whereas arbitrary !-box operations do not.

▶ Remark. Note that the set $\mathcal{F}_\Sigma$ in Definition 13 is defined inductively by relying on a simultaneous recursive definition of Boxes. This is non-circular, since the inductive steps always rely on calls to Boxes on strictly smaller formulas. Unsurprisingly, this style of definition is called *induction-recursion* [7].

In order to talk about instances of !-formulas, we must extend !-box operations from !-tensors to arbitrary formulas.

▶ **Definition 14.** For Op_B one of the operations $\mathrm{Kill}_B, \mathrm{Exp}_{B,\mathbf{fr}}, \mathrm{Copy}_{B,\mathbf{fr}}, \mathrm{Drop}_B$:

- $\mathrm{Op}_B(G = H) := \mathrm{Op}_B(G) = \mathrm{Op}_B(H)$
- $\mathrm{Op}_B(X \wedge Y) := \mathrm{Op}_B(X) \wedge \mathrm{Op}_B(Y)$
- $\mathrm{Op}_B(X \to Y) := \mathrm{Op}_B(X) \to \mathrm{Op}_B(Y)$
- $\mathrm{Op}_B(\forall A.\ X) := \begin{cases} \forall A.\ X & B \in \downarrow A \\ \forall A.\ \mathrm{Op}_B(X) & B \notin \downarrow A \end{cases}$

▶ **Theorem 15.** *!-box operations preserve the property of being a formula.*

Proof. We prove this using structural induction on !-formulas.

- If $G = H$ is a formula then G and H have the same free edges in the same !-boxes. Hence $\mathrm{Op}_B(G)$ and $\mathrm{Op}_B(H)$ have the same free edges (a or $\mathbf{fr}(a)$ for a free in $G = H$) and these are in the same !-boxes.
- For the next two cases we have $\mathrm{Boxes}(X)$ and $\mathrm{Boxes}(Y)$ compatible. Op_B takes the unique connected component, S, containing B and replaces it with $\mathrm{Op}_B(S)$. This can only have gained fresh !-box names so $\mathrm{Boxes}(\mathrm{Op}_B(X))$ and $\mathrm{Boxes}(\mathrm{Op}_B(Y))$ are still compatible.
- If $B \in \downarrow A$ then the final case is trivial. If $B \notin \downarrow A$ then the component $\downarrow A$ is not affected by Op_B so is still a component of $\mathrm{Op}_B(X)$. ◀

5 The rules of !L

We now define a simple logic over !-formulas, which we call !L. Our presentation is given in terms of sequents of the form: $\Gamma \vdash Y$, where $\Gamma := X_1, X_2, \ldots, X_n$ is a finite sequence of

!-formulas. We will always assume in writing a sequent that all of the formulas involved have compatible !-boxes. We take the core logical rules to be those from positive intuitionistic logic with cut:

$$\frac{}{X \vdash X}\ (\text{Ident}) \qquad \frac{\Gamma \vdash Y}{\Gamma, X \vdash Y}\ (\text{Weaken}) \qquad \frac{\Gamma, X, Y, \Delta \vdash Z}{\Gamma, Y, X, \Delta \vdash Z}\ (\text{Perm}) \qquad \frac{\Gamma, X, X \vdash Y}{\Gamma, X \vdash Y}\ (\text{Contr})$$

$$\frac{\Gamma \vdash X \qquad \Delta \vdash Y}{\Gamma, \Delta \vdash X \wedge Y}\ (\wedge I) \qquad \frac{\Gamma \vdash X \wedge Y}{\Gamma \vdash X}\ (\wedge E_1) \qquad \frac{\Gamma \vdash X \wedge Y}{\Gamma \vdash Y}\ (\wedge E_2)$$

$$\frac{\Gamma \vdash X \to Y}{\Gamma, X \vdash Y}\ (\to E) \qquad \frac{\Gamma, X \vdash Y}{\Gamma \vdash X \to Y}\ (\to I) \qquad \frac{\Gamma \vdash X \qquad \Delta, X \vdash Y}{\Gamma, \Delta \vdash Y}\ (\text{Cut})$$

The rules for introducing and eliminating $\forall$ are also analogous to the usual rules. Given any $\mathbf{rn} : \mathcal{B} \to \mathcal{B}$ a bijective renaming function for !-boxes that is identity except on $\downarrow A$, and let $\mathbf{rn}(X)$ be the application of that renaming to a formula. Then:

$$\frac{\Gamma \vdash \mathbf{rn}(X)}{\Gamma \vdash \forall A.\ X}\ (\forall I) \qquad \frac{\Gamma \vdash \forall A.\ X}{\Gamma \vdash \mathbf{rn}(X)}\ (\forall E)$$

where in the case of $\forall I$ we also require that $\mathbf{rn}(\downarrow A)$ is disjoint from $\text{Boxes}(\Gamma)$.

To these core logical rules, we add rules capturing the fact that $=$ is an equivalence relation and a congruence:

$$\frac{}{\Gamma \vdash G = G}\ (\text{Refl}) \qquad \frac{\Gamma \vdash G = H}{\Gamma \vdash H = G}\ (\text{Symm}) \qquad \frac{\Gamma \vdash G = H \qquad \Gamma \vdash H = K}{\Gamma \vdash G = K}\ (\text{Trans})$$

$$\frac{\Gamma \vdash G = H}{\Gamma \vdash [G]^A = [H]^A}\ (\text{Box}) \qquad \frac{\Gamma \vdash G = H}{\Gamma \vdash FG = FH}\ (\text{Prod}) \qquad \frac{\Gamma \vdash G = G'}{\Gamma \vdash \text{Ins}_{A \ni K}(G) = \text{Ins}_{A \ni K}(G')}\ (\text{Ins})$$

where $\text{Ins}_{A \ni K}$ inserts the expression K into the !-box $A \in \text{Boxes}(G)$. The last three rules allow an equation to be applied to a sub-expression. The first two rules allow us to build the context on to the outside of an equation, whereas the third one allows us to add some extra context within any !-box in an equation. These are precisely the equational reasoning rules introduced for !-tensors in [13]. The only difference is we call the 'weakening' operation from that paper 'insertion' to avoid clash with the logical notion.

The main utility of universal quantification is to control the application of !-box operations. In order to start instantiating a !-box (or one of its children), it must be under a universal quantifier:

$$\frac{\Gamma \vdash \forall A.\ X}{\Gamma \vdash \text{Kill}_B(X)}\ (\text{Kill}) \qquad \frac{\Gamma \vdash \forall A.\ X}{\Gamma \vdash \text{Exp}_B(X)}\ (\text{Exp})$$

$$\frac{\Gamma \vdash \forall A.\ X}{\Gamma \vdash \text{Drop}_B(X)}\ (\text{Drop}) \qquad \frac{\Gamma \vdash \forall A.\ X}{\Gamma \vdash \text{Copy}_B(X)}\ (\text{Copy})$$

where $B \leq A \in \text{Boxes}(X)$. These rules, along with $(\forall E)$ play an analogous role to the substitution of a universally-quantified variable for an arbitrary term.

The final rule of the logic is *!-box induction*, which allows us to introduce new !-boxes. For a top-level !-box A, we have:

$$\frac{\Gamma \vdash \text{Kill}_A(X) \qquad \Delta, X \vdash \forall B_1.\ \ldots \forall B_n.\ \text{Exp}_A(X)}{\Gamma, \Delta \vdash X}\ (\text{Induct})$$

where A does not occur free in Γ or Δ and B_1 to B_n are the fresh names of children of A coming from its expansion.

6 Semantics

In this section, we give a semantic interpretation for !-logic formulas using a compact closed category $\mathcal{C}$. For any compact closed category $\mathcal{C}$, a choice of valuation $[\![-]\!] : \Sigma \to \mathcal{C}$ of the generators in Σ will fix a unique morphism $[\![G]\!]$ for any *concrete* (i.e. !-box-free) tensor G. Thus $\mathcal{C}$ comes with an interpretation for equality between concrete tensors. From this, we can build up everything else. For concrete tensors G, H, there is an obvious way to assign a truth value to the formula $G = H$:

$$[\![G = H]\!] := \begin{cases} T & \text{if } [\![G]\!] = [\![H]\!] \\ F & \text{otherwise} \end{cases} \tag{6}$$

As we first mentioned in Section 4, !-logic formulas should be thought of as mappings from instantiations to truth values. Equivalently, they can be thought of as sets of instantiations: namely the set of all instantiations for which the formula holds. Applying this interpretation to atomic formulas yields the following definition:

▶ **Definition 16.** For an atomic !-formula $G = H$ and a valuation $[\![-]\!] : \Sigma \to \mathcal{C}$, we let:

$$[\![G = H]\!] = \left\{ i \in \mathrm{Inst}(\mathrm{Boxes}(G = H)) \;\middle|\; [\![i(G)]\!] = [\![i(H)]\!] \right\} \tag{7}$$

Concrete tensors are equal if and only if they are equal for the trivial instantiation 1. We can interpret truth values as a special case of sets of instantiations: $T = \{1\}$ and $F = \{\}$. Then, in the case of concrete tensors, (7) reduces to (6).

For a forest F and any $i \in \mathrm{Inst}(F)$, and a component $S \subseteq F$, we write $i|_S$ for the restriction of i to only operations involving elements of S (or fresh copies thereof). For a !-formula X, we write $i|_X$ for $i|_{\mathrm{Boxes}(X)}$. Using restrictions of instantiations, we can lift the above definition from atoms to all formulas.

▶ **Definition 17.** The interpretation $[\![-]\!]$ of a !-logic formula is defined recursively as:

$$[\![X \wedge Y]\!] := \left\{ i \in \mathrm{Inst}(\mathrm{Boxes}(X \wedge Y)) \;\middle|\; i|_X \in [\![X]\!] \wedge i|_Y \in [\![Y]\!] \right\}$$
$$[\![X \to Y]\!] := \left\{ i \in \mathrm{Inst}(\mathrm{Boxes}(X \to Y)) \;\middle|\; i|_X \in [\![X]\!] \to i|_Y \in [\![Y]\!] \right\}$$
$$[\![\forall A.\ X]\!] := \left\{ i \in \mathrm{Inst}(\mathrm{Boxes}(\forall A.\ X)) \;\middle|\; \forall j \in \mathrm{Inst}(\downarrow A)\ .\ i \circ j \in [\![X]\!] \right\}$$

This style of interpretation is very much analogous to that of predicate logic. Whereas one can interpret a predicate with free variables as the set of all values of those variables for which the predicate is true, a !-logic formula is interpreted as the set of all instantiations at which the resulting concrete formula holds.

By contrast, we always interpret sequents as truth values. To do so, we push all of the assumptions to the right and universally quantify over any free !-boxes:

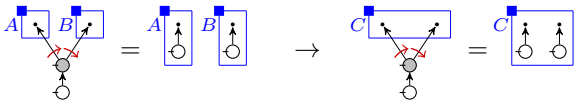
$$[\![X_1, \ldots, X_n \vdash Y]\!] := [\![\forall A_1 \ldots \forall A_m.((X_1 \wedge \ldots \wedge X_n) \to Y)]\!]$$

where $\downarrow A_1, \ldots, \downarrow A_m$ are the free !-boxes in $X_1, \ldots, X_n, Y$.

▶ **Theorem 18** (Soundness). *If $\Gamma \vdash X$ is derivable in !L, then $[\![\Gamma \vdash X]\!]$ is true for any compact closed category $\mathcal{C}$.*

Proof. See Appendix A. ◀

The question of completeness for !L is still open. For the case of atomic !-formulas, this seems to follow straightforwardly from the fact that string diagrams (or equivalently, tensors) are sound and complete for compact closed categories. So, concrete !-tensor equations are true in all models if and only if they are identical tensors. Thus, for the case of general !-tensor equations, the problem reduces to deciding whether two !-tensors with corresponding !-boxes always have identical instances. However, once implication enters the game, we get many non-trivial formulas that hold in all models. For example, an equation with two !-boxes without edges between them always implies another equation obtained by *merging* those !-boxes:

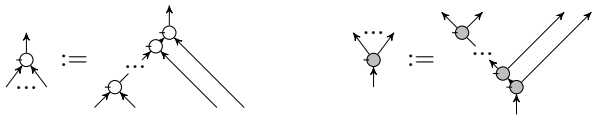


In this case, it is always possible to use !-box induction to prove such an implication (and many others). However, whether the rules in Section 5 suffice to get everything is a topic of continuing research.

7 Inductive proofs for non-commutative bialgebras

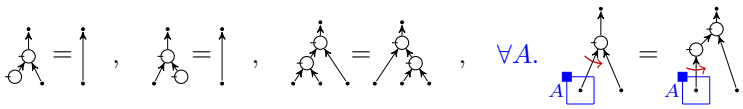
In this section, we will give a flavour for formal proofs in !L and show how they can be used to derive highly non-trivial !-box equations using a combination of !-box induction and rewriting. To avoid massive proof trees, we will abbreviate stacks of equational reasoning rules as sequences of rewrite steps (marked with (*)'s), suppress ∀-intro/elim, and write (Assm) to abbreviate using an assumption.

Recall that a *bialgebra* consists of a monoid, a comonoid, and four extra equations governing their interaction. We will extend the signature of (co)monoids to also allow for n-ary operations, standing for left-associated trees of multiplications and comultiplications:

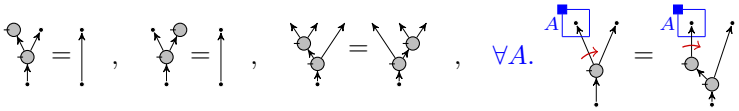


We then assume the usual (co)monoid laws, along with the definition of a higher-arity tree:

$\Gamma_M :=$



$\Gamma_C :=$



For bialgebras, we start with these equations and add four more:

$\Gamma_{BA} := \Gamma_M, \Gamma_C,$

As can be seen from the second bialgebra equation, units are copied by comultiplications. We saw an n-ary generalisation of this in equation (1), which we can now prove formally (and succinctly!):

▶ **Theorem 19.**

$$\frac{}{\Gamma_{BA} \vdash \forall A.\ \ A = A} \ (19)$$

Proof.

$$\dfrac{\dfrac{}{\Gamma_{BA} \vdash \ = \ } \ (\text{Assm}) \qquad \dfrac{}{\Gamma_{BA},\ = \ \vdash \ = \ } \ (*)}{\Gamma_{BA} \vdash \ = \ } \ (\text{Induct})$$

$$(*) \quad = \quad = \quad = \quad \overset{i.h.}{=}$$

◀

The fact that counits are copied by trees of multiplications could be proved similarly, but we can generalise even more. We now prove that a tree of multiplications, followed by a tree of comultiplications is equal to a complete bipartite graph of comultiplications before multiplications. This rule generalises (and hence can replace) all 4 of the existing bialgebra rules. First we need a little lemma:

▶ **Lemma 20.**

$$\frac{}{\Gamma_{BA} \vdash \forall A.\ \ A = A} \ (20)$$

Proof.

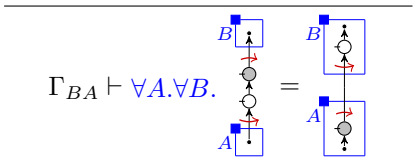
$$\dfrac{\dfrac{}{\Gamma_{BA} \vdash \ = \ } \ (\text{Assm}) \qquad \dfrac{}{\Gamma_{BA},\ = \ \vdash \ = \ } \ (**)}{\Gamma_{BA} \vdash \ = \ } \ (\text{Induct})$$

$$(**) \quad = \quad = \quad = \quad \overset{i.h.}{=} \quad = \quad = \quad =$$

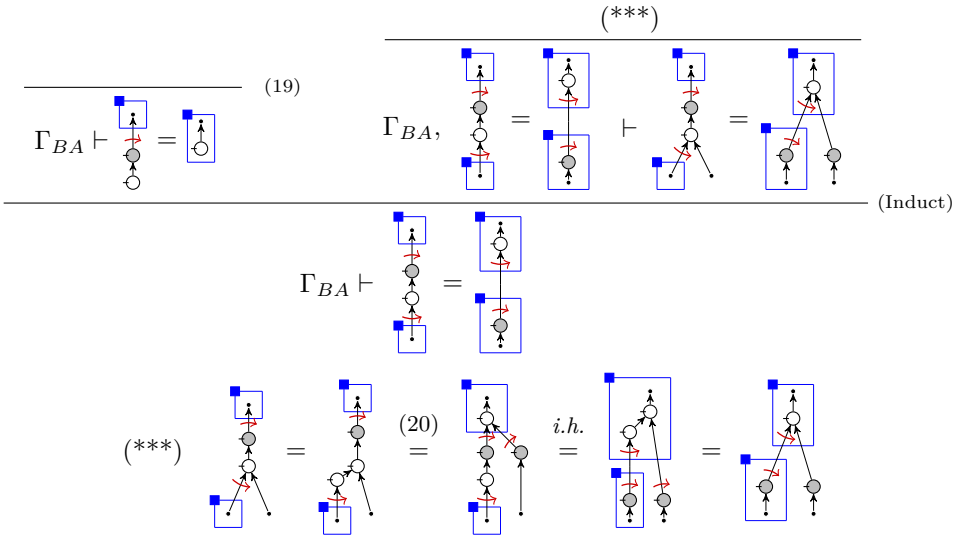
◀

. . . from which we can prove the main theorem:

▶ **Theorem 21.**



Proof.



◀

References

1. John C. Baez and Jason Erbele. Categories in control. Technical report, arXiv:1405.6881, 2014.
2. F. Bonchi, P. Sobocinski, and F. Zanasi. A categorical semantics of signal flow graphs. In *CONCUR'14: Concurrency Theory.*, volume 8704 of *Lecture Notes in Computer Science*, pages 435–450. Springer, 2014.
3. B. Coecke. Quantum picturalism. *Contemporary Physics*, 51:59–83, 2009. arXiv:0908.1787.
4. B. Coecke and R. Duncan. Interacting quantum observables. In *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science, 2008.
5. B. Coecke, R. Duncan, A. Kissinger, and Q. Wang. Strong complementarity and non-locality in categorical quantum mechanics. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS)*. IEEE Computer Society, 2012. arXiv:1203.4988.
6. Lucas Dixon and Ross Duncan. Extending Graphical Representations for Compact Closed Categories with Applications to Symbolic Quantum Computation. *AISC/MKM/Calculemus*, pages 77–92, 2008.
7. Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer Berlin Heidelberg, 1999.
8. Andre Joyal and Ross Street. The geometry of tensor calculus I. *Advances in Mathematics*, 88:55–113, 1991.
9. D. Kartsaklis. *Compositional Distributional Semantics with Compact Closed Categories and Frobenius Algebras.* PhD thesis, University of Oxford, 2014.

10 Aleks Kissinger. Abstract tensor systems as monoidal categories. In C Casadio, B Coecke, M Moortgat, and P Scott, editors, *Categories and Types in Logic, Language, and Physics: Festschrift on the occasion of Jim Lambek's 90th birthday*, volume 8222 of *Lecture Notes in Computer Science*. Springer, 2014. arXiv:1308.3586 [math.CT].

11 Aleks Kissinger, Alex Merry, and Matvey Soloviev. Pattern graph rewrite systems. In *Proceedings of DCM 2012*, volume 143 of *EPTCS*, 2012. arXiv:1204.6695 [math.CT].

12 Aleks Kissinger and David Quick. Tensors, !-graphs, and non-commutative quantum structures. In *Proceedings of the 11th workshop on Quantum Physics and Logic, QPL 2014, Kyoto, Japan, 4-6th June 2014.*, pages 56–67, 2014. arXiv:1412.8552 [cs.LO].

13 Aleks Kissinger and David Quick. Tensors, !-graphs, and non-commutative quantum structures (extended version), 2015. arXiv:1503.01348.

14 Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning, 2015. arXiv:1503.01034.

15 Alexander Merry. *Reasoning with !-Graphs*. PhD thesis, University of Oxford, 2014.

A Proof of soundness for !L

In this section, we prove Theorem 18, i.e. the soundness of $[\![-]\!]$ with respect to !L. To do so, it suffices to show that $[\![-]\!]$ respects each of the rules of the logic.

For $i \in \mathrm{Inst}(F)$ and a formula X such that $\mathrm{Boxes}(X)$ is a component of F, we will write $i \vDash X$ as shorthand for $i|_X \in [\![X]\!]$. Using this notation, we can rewrite the interpretation as follows:

$$\begin{aligned}
i \vDash G = H &\iff [\![i(G)]\!] = [\![i(H)]\!] && i \in \mathrm{Inst}(\mathrm{Boxes}(G = H)) \\
i \vDash X \wedge Y &\iff i \vDash X \wedge i \vDash Y && i \in \mathrm{Inst}(\mathrm{Boxes}(X \wedge Y)) \\
i \vDash X \to Y &\iff i \vDash X \to i \vDash Y && i \in \mathrm{Inst}(\mathrm{Boxes}(X \to Y)) \\
i \vDash \forall A.X &\iff \forall j \in \mathrm{Inst}(\downarrow A).\ i \circ j \vDash X && i \in \mathrm{Inst}(\mathrm{Boxes}(\forall A.X))
\end{aligned}$$

Universal quantification over entire components of $\mathrm{Boxes}(X)$ is well-behaved for the following reason:

▶ **Lemma 22.** *For a forest F, let A, B be elements in distinct connected components of F, and let $\mathrm{Boxes}(X) \triangle F$. Then, $\mathrm{Op}_A(\mathrm{Op}'_B(X)) = \mathrm{Op}'_B(\mathrm{Op}_A(X))$ for any !-box operations $\mathrm{Op}_A, \mathrm{Op}'_B$.*

Proof. Since !-box operations recurse down to equations between !-tensors, it suffices to show that $\mathrm{Op}_A(\mathrm{Op}'_B(G = H)) = \mathrm{Op}'_B(\mathrm{Op}_A(G = H))$. Since neither A nor B is a child of the other, this is easy to check. The only complication is dealing with the freshness functions $\mathbf{fr}_A, \mathbf{fr}_B$ (possibly) associated with the two operations. These necessarily operate on disjoint sets of boxnames, so the only overlap might be on edgenames. However, since there is an infinite supply of fresh edgenames, it is always possible to choose new freshness functions such that $\mathbf{fr}_A \circ \mathbf{fr}_B = \mathbf{fr}'_B \circ \mathbf{fr}'_A$. Then, it is straightforward to check that $\mathrm{Op}_{A,\mathbf{fr}_A}(\mathrm{Op}'_{B,\mathbf{fr}_B}(G = H)) = \mathrm{Op}'_{B,\mathbf{fr}'_B}(\mathrm{Op}_{A,\mathbf{fr}'_A}(G = H))$. ◀

A related fact about re-ordering operations in an instantiation is that they can always be put in normal form:

▶ **Lemma 23.** *Given an instantiation $i \in \mathrm{Inst}(X)$ and a top-level !-box $A \in X^\top$, i can be rewritten as $i' \circ \mathrm{Kill}_A \circ \mathrm{Exp}^n_A$ where $i' \in \mathrm{Inst}(\mathrm{Kill}_A \circ \mathrm{Exp}^n_A(X))$.*

Proof. We need to check that operations on A can always be commuted to the right, past other operations. If B is not nested in A, this is true by Lemma 22. Otherwise, $B \leq A$ and:

- If $\mathrm{Op}_A = \mathrm{Kill}_A$ then killing A will erase any part of the !-formula resulting from Op_B, i.e. $\mathrm{Kill}_A \circ \mathrm{Op}_B = \mathrm{Kill}_A$.
- If $\mathrm{Op}_A = \mathrm{Exp}_{A,\mathbf{fr}}$ then $\mathrm{Exp}_{A,\mathbf{fr}} \circ \mathrm{Op}_B = \mathrm{Op}_{\mathbf{fr}(B)} \circ \mathrm{Op}_B \circ \mathrm{Exp}_{A,\mathbf{fr}}$. In the case that $\mathrm{Op}_B = \mathrm{Exp}_B$, freshness functions on the RHS need to be chosen to produce identical names to the LHS. ◀

▶ **Notation 24.** We will write KE^n_A as a shorthand for $\mathrm{Kill}_A \circ \mathrm{Exp}^n_A$.

▶ **Lemma 25.** *For any !-formula X and for $B_1, \ldots B_n$ the free, top-level !-boxes in X:*

$$\forall i \in \mathrm{Inst}(\mathrm{Boxes}(X)).\ i \vDash X \iff [\![\forall B_1 \ldots \forall B_n.X]\!] = \{1\} = T$$

Proof. First, assume the LHS, which is equivalent to $[\![X]\!] = \mathrm{Inst}(\mathrm{Boxes}\, X)$. For any !-formula Y, if $B_k \in \mathrm{Boxes}(Y)^\top$ and $[\![Y]\!] = \mathrm{Inst}(\mathrm{Boxes}(Y))$, then $[\![Y]\!]$ contains all possible instantiations of $\mathrm{Boxes}(Y)$. In particular, it contains $i \circ j$ for any $i \in \mathrm{Inst}(\mathrm{Boxes}(\forall B_k.Y))$

and $j \in \mathrm{Inst}(\downarrow B_k)$. Thus, $[\![\forall B_k.Y]\!] = \mathrm{Inst}(\mathrm{Boxes}(\forall B_k.Y))$. Iterating this implication, we have $[\![\forall B_1 \ldots \forall B_n.X]\!] = \mathrm{Inst}(\mathrm{Boxes}(\forall B_1 \ldots \forall B_n.X)) = \{1\} = T$.

Conversely, assume $[\![\forall B_1 \ldots \forall B_n.X]\!] = T$. Then every instantiation of the form $j = i_1 \circ i_2 \circ \ldots \circ i_n$, where the operations in i_k only involve !-boxes in $\downarrow B_k$ is in $[\![X]\!]$. But then, by Lemma 22, we can freely commute !-box operations in distinct components of $\mathrm{Boxes}(X)$. So, in fact, every instantiation $i \in \mathrm{Inst}(\mathrm{Boxes}(X))$ is equivalent to an instantiation of the form of j. Then, since $j \in [\![X]\!]$, so is i. ◀

▶ **Theorem 26.** *For any valuation $[\![-]\!] : \Sigma \to \mathcal{C}$, the rules (Ident), (Weaken), (Perm), (Contr), ($\wedge I$), ($\wedge E_1$), ($\wedge E_2$), ($\to E$), ($\to I$), (Cut), ($\forall I$), ($\forall E$), (Refl), (Symm), (Tran), (Box), (Prod), (Ins), (Kill), (Exp), (Drop), (Copy), and (Induct) are sound with respect to $[\![-]\!]$.*

Proof. The basic structural rules just reduce to the same rules concerning instantiations. Let K be the conjunction of Γ and K' the conjunction of Δ throughout. By Lemma 25, to check that $[\![\Gamma \vdash X]\!]$ is true, it suffices to check that, for all $i \in \mathrm{Inst}(\mathrm{Boxes}(K \to X))$, $i \vDash K \to X$.

- (Ident) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(X))$. We need to show $i \in X \to X$, but this is equivalent to $i \vDash X \to i \vDash X$, which is trivially true.
- (Weaken) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}((K \wedge X) \to Y))$ and assume $i \vDash K \to Y$. Then, if $i \vDash K \wedge X$, then $i \vDash K$. So, by assumption, $i \vDash Y$. Thus $i \vdash (K \wedge X) \to Y$.
- (Perm) and (Contr) follow from associativity, commutativity and idempotence of $\wedge$.
- ($\wedge I$) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}((K \wedge K') \to (X \wedge Y)))$ and assume $i \vDash K \to X$ and $i \vDash K' \to Y$. If $i \vDash K \wedge K'$, we have $i \vDash K$ and hence $i \vDash X$. We also have $i \vDash K'$ and hence $i \vDash Y$. Thus $i \vDash X \wedge Y$.
- ($\wedge E1$) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(K \to X))$. Then, there exists $i' \in \mathrm{Inst}(\mathrm{Boxes}(K \to (X \wedge Y)))$ that restricts to i. Assume $i' \vDash K \to (X \wedge Y)$. If $i \vDash K$ then $i' \vDash K$ and hence $i' \vDash X \wedge Y$, which implies that $i' \vDash X$. So, $i \vDash X$.
- ($\wedge E2$) is similar to ($\wedge E1$).
- ($\to E$) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}((K \wedge X) \to Y))$ and assume $i \vDash K \to (X \to Y)$. Then, if $i \vDash K \wedge X$ then $i \vDash K$. So, $i \vDash X \to Y$. But, since it is also the case that $i \vDash X$, $i \vDash Y$. Thus $i \vDash (K \wedge X) \to Y$.
- ($\to I$) is the same as ($\to E$) in reverse.
- (Cut) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(K \wedge K' \to Y))$. Then, there exists $i' \in \mathrm{Inst}(\mathrm{Boxes}(K \to X) \cup \mathrm{Boxes}((K' \wedge X) \to Y))$ that restricts to i. Assume $i' \vDash K \to X$ and $i' \vDash (K' \wedge X) \to Y$. If $i \vDash K \wedge K'$, then $i' \vDash K \wedge K'$ so $i' \vDash K$ and $i' \vDash K'$. The former also implies that $i' \vDash X$. So, $i' \vDash K' \wedge X$ and hence $i' \vDash Y$. Finally, this implies $i \vDash Y$.

($\forall I$) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(K \to \forall A.X))$. We need to show that for any $j \in \mathrm{Inst}(\downarrow A)$, $i \circ j \vDash K \to X$. Assume without loss of generality that any !-box names on operations in i are disjoint from $\mathbf{rn}(\downarrow A)$. This is possible because $\mathbf{rn}(\downarrow A)$ must already be disjoint from $\mathrm{Boxes}(\Gamma)$ (by side-condition) and it must be disjoint from $\mathrm{Boxes}(\forall A.X) = \mathrm{Boxes}(X) \setminus \downarrow A$ by injectivity of $\mathbf{rn}$. The only other !-box names in i are those introduced during instantiation, which can be freely chosen. Let $\mathbf{rn}(j)$ be the instantiation of $\mathbf{rn}(\downarrow A)$ obtained by renaming operations according to $\mathbf{rn}$. Then, by assumption of the rule, we have $i \circ \mathbf{rn}(j) \vDash K \to \mathbf{rn}(X)$. Since $\mathbf{rn}$ is identity except on $\downarrow A$, we have $\mathbf{rn}(i \circ j) \vDash \mathbf{rn}(K \to X)$ and thus $i \circ j \vDash K \to X$.

($\forall E$) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(K \to \mathbf{rn}(X)))$. Suppose $i \vDash K$, then, by assumption $i \vDash \forall A.X$. Let $i' = i|_{\forall A.X}$, then $i' \vDash \forall A.X$, which implies that for all $j \in \mathrm{Inst}(\downarrow A)$, we have $i' \circ j \vDash X$. Renaming both sides yields $\mathbf{rn}(i' \circ j) \vDash \mathbf{rn}(X)$, and since $\mathbf{rn}$ is identity except on $\downarrow A$,

$i' \circ \mathbf{rn}(j) \vDash \mathbf{rn}(X)$. Now, since we are free to choose j, we choose it such that $(i' \circ \mathbf{rn}(j))|_{\mathbf{rn}(X)}$ is equivalent to $i|_{\mathbf{rn}(X)}$. Then $i \vDash \mathbf{rn}(X)$.

The rules (Refl), (Symm), and (Trans) reduce to the properties of equality in $\mathcal{C}$. The congruence rules (Box), (Prod), and (Ins) were proven sound in [13], where the only difference here is the additional (unused) context Γ.

(Kill) Fix $i \in \mathrm{Inst}(\mathrm{Boxes}(K \to \mathrm{Kill}_B(X)))$. Then if $i \vDash K$, by assumption $i \vDash \forall A.X$. Since $B \leq A$ does not occur free in $\forall A.X$, $i \circ \mathrm{Kill}_B \vDash \forall A.X$. For $i' = i|_{\forall A.X}$, choose $j \in \mathrm{Inst}(\downarrow A)$ such that $(i' \circ j)|_X$ is equivalent to $(i \circ \mathrm{Kill}_B)|_X$. Then, $i' \circ j \vDash X$, so $i \circ \mathrm{Kill}_B \vDash X$, and $i \vDash \mathrm{Kill}_B(X)$. (Exp) is similar.

(Copy) and (Drop) are also similar. However, when we choose $j \in \mathrm{Inst}(\downarrow A)$ such that $(i' \circ j)|_X$ is equivalent to $(i \circ \mathrm{Copy}_B)|_X$ or $(i \circ \mathrm{Copy}_B)|_X$, we make use of the fact that instantiations involving Copy / Drop can always be reduced to a normal form which only includes Exp and Kill. This was proven in [13].

Finally, we prove the (Induct) rule. For any top-level !-box A, Lemma 23 says that we can write any instantiation i equivalently as $j \circ \mathrm{KE}_A^n$, where j doesn't contain A. Thus, we will show that, for all n, and all instantiations of the form $i := j \circ \mathrm{KE}_A^n$, $i \vDash (K \wedge K') \to X$. We proceed by induction on n.

For the base case, $i = j \circ \mathrm{Kill}_A$. If $i \vDash K$, then since K doesn't contain A, $i \vDash K$ implies $j \vDash K$. So, by the first premise $j \vDash \mathrm{Kill}_A(X)$. Thus $j \circ \mathrm{Kill}_A \vDash X$, as required. For the step case, assume that for all instantiations of $(K \wedge K') \to X$ of the form $i := j \circ \mathrm{KE}_A^n$, $i \vDash (K \wedge K') \to X$. We need to show for all $i' := j \circ \mathrm{KE}_A^{n+1}$, $i' \vDash (K \wedge K') \to X$. If $i' \vDash K \wedge K'$, then $i' \vDash K'$. Then, since K doesn't contain A, $i \vDash K'$. Combining this with the induction hypothesis yields $i \vDash \forall B_1 \ldots \forall B_m.\, \mathrm{Exp}_A(X)$. Thus, for any instantiation k of the $\downarrow B_1, \ldots, \downarrow B_m$, $i \circ k \vDash \mathrm{Exp}_A(X)$. So, $i \circ k \circ \mathrm{Exp}_A \vDash X$. i' is equivalent to $i \circ k \circ \mathrm{Exp}_A$ for some i, k, so $i' \vDash X$. ◀

Soundness of !L with respect to $[\![-]\!]$ then follows from Theorem 26.