Encoding !-tensors as !-graphs with neighbourhood orders

David Quick
University of Oxford
david.quick@cs.ox.ac.uk

Diagrammatic reasoning using string diagrams provides an intuitive language for reasoning about morphisms in a symmetric monoidal category. To allow working with infinite families of string diagrams, !-graphs were introduced as a method to mark repeated structure inside a diagram. This led to !-graphs being implemented in the diagrammatic proof assistant Quantomatic. Having a partially automated program for rewriting diagrams has proven very useful, but being based on !-graphs, only commutative theories are allowed. An enriched abstract tensor notation, called !-tensors, has been used to formalise the notion of !-boxes in non-commutative structures. This work-in-progress paper presents a method to encode !-tensors as !-graphs with some additional structure. This will allow us to leverage the existing code from Quantomatic and quickly provide various tools for non-commutative diagrammatic reasoning.

Contents

1 Background **2**
1.1 Motivation . . . 2
1.2 String graphs as typed graphs . . . 4
1.3 !-Graphs . . . 5

2 !-Graphs with simple overlap and ordered neighbourhoods **5**
2.1 Simple overlap . . . 6
2.2 Adding orders to neighbourhoods . . . 6

3 Encoding !-tensors **7**
3.1 Defining the mapping $\mathscr{I}$ . . . 7
3.2 $\mathscr{I}$ is Well-Behaved . . . 9

4 Future work **11**
4.1 Theory . . . 11
4.2 Implementation . . . 12

A Graphical user interface **14**

To appear in EPTCS.

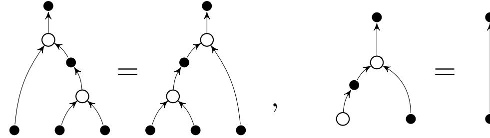
© D. Quick
This work is licensed under the
Creative Commons Attribution License.

1 Background

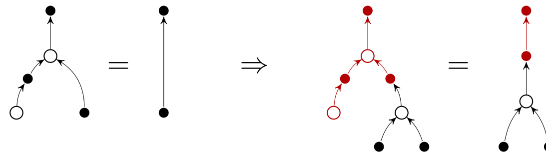
1.1 Motivation

Reasoning using diagrams can often be simpler and more intuitive than using a term-based mathematical syntax [7]. There has recently been a shift toward using graphical calculi in a number of different fields. The *String graph* formalism is one which has far-reaching applications in areas including categorical quantum mechanics [4, 3, 5], computational linguistics [8] and control theory [2, 1]. String graphs are a formalism for typed directed graphs, with the idea of edges replaced by (directed) wires. Unlike edges, wires do not need to have vertices at the ends. Those without starting vertices can be considered inputs and those without end vertices can be considered outputs. Wires also allow for identity edges (neither end connected to a node) or loops (a wire connected to itself). String graphs achieve this by having wire-vertices along a wire, so that a loop for example could be two wire-vertices a and b with edges $a \to b$ and $b \to a$. String graphs can be combined by plugging outputs from one in to inputs of another.

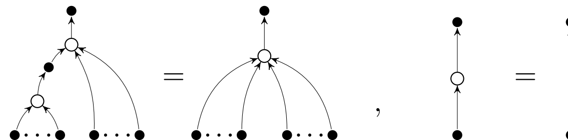
A diagrammatic theory is a set of equations between string diagrams on some set of generating morphisms. For example, a (commutative) associative multiplication operation with a unit can be described by the string diagram equations:



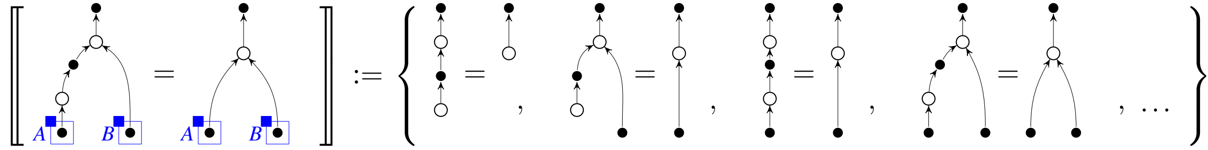
An equation, $G = H$, between string graphs can be used in a form of rewriting where a subgraph matching G can be replaced by H.



Many theories allow for nodes to have variable arity edges. For example, the multiplication and unit above can be replaced by a single operation taking the product of an arbitrary number of inputs (order unimportant), along with two diagram equations.



This extension, while powerful, loses the mathematical rigour from the previous formalism (as can be seen from the use of ellipses). To return to a rigorous semantics, !-boxes (pronounced bang-boxes) were introduced in [6] to designate (by enclosing in a blue box) parts of a diagram which are allowed to be repeated. Adding !-boxes to string graphs results in the notion of a *!-graph*. This was formalised in [10, 15] and has many applications, particularly in categorical quantum mechanics. Note that a !-graph represents a family of string graphs which can be recovered by two !-box operations, Kill_B and Exp_B. Kill_B is the operation removing the !-box B and all contents from a !-graph. Exp_B is the operation which creates a new copy of the contents of B attached to the same nodes. Equations between !-graphs have the restriction that the !-graphs have the same !-boxes with the same nesting (!-boxes can contain other !-boxes). Hence the ellipsis equation in our example theory describes the following set:



As !-graphs get applied to more complicated systems, rewriting by hand becomes difficult. This problem was solved by the introduction of Quantomatic [11], an automated theorem prover which allows rewriting of !-graphs using substitution. Quantomatic encompasses many tools for automated reasoning with string diagrams and has the ability to output graphical derivations directly to LaTeX.

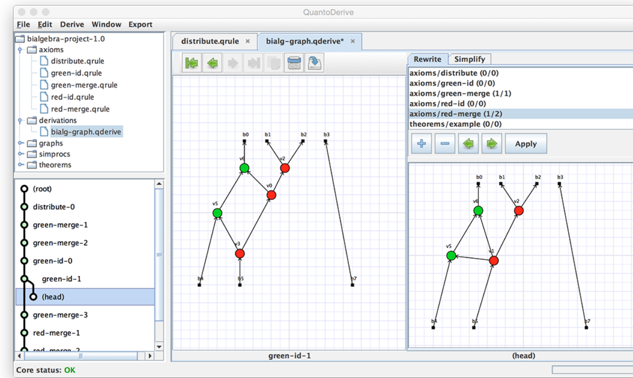


Figure 1: Example derivation in Quantomatic

One major drawback of both !-graphs and Quantomatic is that they are only designed to allow theories in which all nodes are commutative on inputs and outputs. This not only restricts the types of theory we can work with but more subtly, it rules out the option to definitionally extend a theory. i.e. defining new nodes (possibly inductively) as diagrams of other nodes. This is important if we want to rigorously replace finite arity nodes with their more intuitive arbitrary arity counterparts.

Since graphs inherently lack any notion of order on the edges surrounding a node, a more logical choice for representing non-commutative systems is a tensor notation in which edges are listed (in order) in a subscript. This tensor syntax was suggested in [9] and then extended in [12] to !-tensors which use an enriched version of Penrose's abstract tensor notation [16] allowing non-commutative nodes and !-boxes. Here nodes are of the form ϕ_e where ϕ is the type and e is called the edgeterm (describes order of edges). Nodes are enclosed in $[_]^B$ to represent that they are inside a !-box named B. Edgeterms contain edges out $\hat{a}$ and edges in $\check{b}$ where a and b are the names of the edges. If $\hat{a}$ and $\check{a}$ appear in a !-tensor it represents an edge between their nodes. In the edgeterms anything inside $[_\rangle^B$ is an edge entering B which should be expanded clockwise during Exp_B and $\langle_]^B$ represents edges with anticlockwise expansion.

$$\Psi_{\hat{f}\check{a}\check{b}}\phi_{\hat{a}\hat{b}\check{c}\check{d}\check{e}} \quad := \quad , \qquad [\phi_{\hat{a}\check{c}\hat{b}}\Psi_{\hat{c}\check{d}}]^B\xi_{[\check{a}\rangle^B}\zeta_{\langle\check{b}\hat{d}]^B\check{e}} \quad := \qquad (1)$$

While it would be possible to write new automation tools based on term rewriting in the !-tensor formalism, this would require rebuilding the features already present in [11] from scratch. Instead we wish to leverage this pre-existing code but extend it to work with the logic of non-commutative theories as formalised in [13], by adding edgeterms to !-graphs. This paper shows work in progress demonstrating how we can fully encode !-tensors as !-graphs with added data. For the remainder of this section we recall the definitions of string graphs and !-graphs as defined in [15].

1.2 String graphs as typed graphs

The standard definition of a graph in mathematics is not powerful enough for our purposes. As can be seen from the string graphs in Section 1.1 our vertices need to be labelled (as generator types or wire-vertices). There should also be some restrictions as to which vertex types edges can connect. We achieve this using a typegraph, where every vertex gets mapped to a type.

Definition 1.1. A graph G along with a graph morphism $\tau : G \to \mathscr{G}$ is said to be a *$\mathscr{G}$-typed graph*.

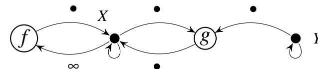
Definition 1.2. If O and M are sets and $\mathrm{dom}, \mathrm{cod} : M \to (O \times \{\bullet, \infty\})^*$ are functions into lists of pairs of elements of O and either $\bullet$ or ∞, then $(O, M, \mathrm{dom}, \mathrm{cod})$ is a *compressed monoidal signature*.

We refer to the set O as the objects, the set M as the morphisms and dom and cod assign the domain and codomain of each morphism allowing edges to be tagged as single ($\bullet$) or variable (∞) arity.

Definition 1.3. Given a compressed monoidal signature $T = (O, M, \mathrm{dom}, \mathrm{cod})$, the *derived compressed typegraph* $\mathscr{G}_T$ has vertices $O \cup M$, a self-loop for every $X \in O$ and, for every $f \in M$,

- one $\bullet$-edge from X to f for each $(X, \bullet)$ in $\mathrm{dom}(f)$,
- one $\bullet$-edge from f to X for each $(X, \bullet)$ in $\mathrm{cod}(f)$,
- one ∞-edge from X to f for each (X, ∞) in $\mathrm{dom}(f)$, and
- one ∞-edge from f to X for each (X, ∞) in $\mathrm{cod}(f)$.

For example, the compressed monoidal signature T with the two morphisms $f : [X^\infty] \to [X^\bullet]$ and $g : [Y^\bullet, X^\bullet] \to [X^\bullet]$ results in a typegraph $\mathscr{G}_T$ of the form:



If a graph G is $\mathscr{G}_T$-typed then we write $W(G)$ for the wire-vertices (those mapped into O) and $N(G)$ for the node-vertices (those mapped into M). $\bullet$-tagged edges represent edges of fixed (single) arity whereas ∞-tagged edges represent edges of variable arity.

Definition 1.4. A $\mathscr{G}_T$-typed graph (G, τ) is called a *string graph* if τ is arity-matching (a bijection between the $\bullet$-edge neighbourhoods of N and $\tau(N)$, for each $N \in N(G)$) and each wire-vertex in G has at most one incoming edge and at most one outgoing edge.

The notion of a string graph requires node vertices to be connected by chains of wire vertices which we refer to as wires. Wire points with no incoming edge represent an input to the graph and wire points with no outgoing edge represent outputs. We refer to the input vertices and output vertices together as the boundary. The set of wire-vertices in a wire are referred to as the interior of the wire.

Remark 1.5. The edges around a node-vertex in a string graph are either $\bullet$-tagged, which means they are single arity; or they are ∞-tagged meaning there could be many copies made. This is important when we transition to !-graphs where we need to be careful not to copy single arity edges.

1.3 !-Graphs

!-boxes denote subsections of a graph which can be repeated many times. We represent a !-box B as a node (of the new type !) with an edge to each vertex contained in B.

Definition 1.6. A subgraph O of a string graph G is said to be *open* if it is not adjacent to any wire-vertices or incident to any fixed-arity edges.

Openness (as described in [15]) encapsulates the property of being able to repeat that part of a graph an arbitrary number of times. It ensures adjacent edges are copied with nodes; fixed arity edges are not copied without the adjacent node; and that wires cannot be partially inside a !-box (to avoid wire splitting).

Definition 1.7. Given a compressed monoidal signature T, the *derived compressed !-typegraph* $\mathscr{G}_{T!}$ is $\mathscr{G}_T$ with the addition of a vertex ! along with edges from ! to every vertex (including itself).

Our previous example of the compressed monoidal signature T with morphisms $f : [X^\infty] \to [X^\bullet]$ and $g : [Y^\bullet, X^\bullet] \to [X^\bullet]$ results in a !-typegraph $\mathscr{G}_{T!}$ of the form:

(2)

If G is $\mathscr{G}_{T!}$-typed, write $!(G)$ for the box-vertices in G (those mapped to !) and $U(G)$ for the full subgraph of G with all vertices except $!(G)$. Given $B \in !(G)$ let $C(B)$ be the full subgraph of G with nodes which have edges from B. So $C(B)$ represents the contents of a !-box B. We use this notation in the !-graph conditions to ensure !-boxes are well-behaved under !-box operations.

Definition 1.8. A $\mathscr{G}_{T!}$-typed graph G is called a *!-graph* if the following hold:

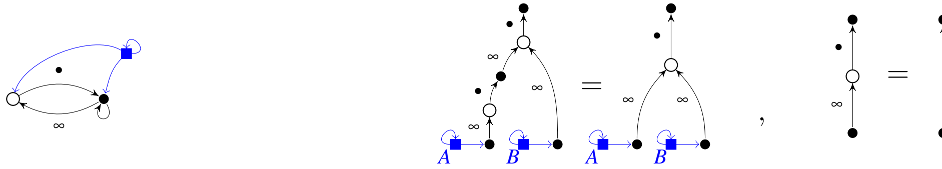
BG1. $U(G)$ is a $\mathscr{G}_T$-typed string graph;

BG2. the full subgraph with vertices $!(G)$ is posetal;

BG3. $\forall B \in !(G)$, $U(C(B))$ is an open subgraph of $U(G)$; and

BG4. $\forall B, B' \in !(G)$, if $B' \in C(B)$ then $C(B') \subseteq C(B)$.

As an example, the theory described in Section 1.1 has !-typegraph and !-graph rules as follows:



2 !-Graphs with simple overlap and ordered neighbourhoods

Given a compact closed signature Σ we wish to encode the !-tensors of $\mathscr{T}_\Sigma$ as !-graphs with some additional data. However, !-graphs allow for two or more non-nested !-boxes to share nodes, which is not permitted with !-tensors. In this section we define a restricted set of !-graphs and the additional structure recording non-commutativity.

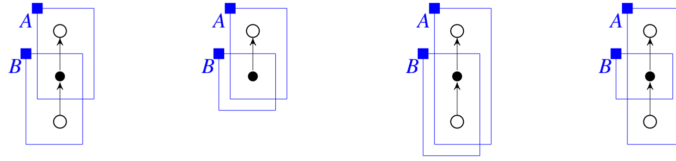
2.1 Simple overlap

We must now add the restriction to rule out !-graphs with overlap on nodes, we do this by introducing the idea of simple overlap of !-boxes.

Definition 2.1. Given a pair of non-nested !-boxes B and B', we say they *overlap simply* if $C(B) \cap C(B')$ consists of only the interior of zero or more wires, where at least one endpoint is a node-vertex and any node-vertex endpoints are in either $C(B)$ or $C(B')$.

Definition 2.2. A !-graph where any two non-nested !-boxes overlap simply is called a *!-graph with simple overlap*.

Compare this definition to !-graphs with trivial overlap (BGTO) which were first defined in [14] where it was shown that they can be encoded using a context-free grammar. The difference here is that we allow non-nested overlap at a boundary rather than just between two node-vertices.



The first two !-graphs have only simple overlap (though the second is non-trivial) the final two have non-simple overlap.

2.2 Adding orders to neighbourhoods

We now only need to describe the additional data to keep track of edge orders. For this we need the notion of an edgeterm which was originally described in [12]. Edgeterms list, in clockwise order, the edges (in $\check{a}$ or out $\hat{a}$) around a node grouping those which expand clockwise under !-box B inside $[_\rangle^B$ and those which expand anticlockwise inside $\langle_]^B$. For examples see (1) and [12].

Definition 2.3. The set of *edgeterms* $\mathscr{T}_\mathscr{E}$ over the edge names $\mathscr{E}$ and !-box names $\mathscr{B}$ is defined inductively as follows:

• $\varepsilon \in \mathscr{T}_\mathscr{E}$	(empty edgeterm)
• $\check{a}, \hat{a} \in \mathscr{T}_\mathscr{E}$	$a \in \mathscr{E}$
• $\langle e]^A, [e\rangle^A \in \mathscr{T}_\mathscr{E}$	$e \in \mathscr{T}_\mathscr{E}$, $A \in \mathscr{B}$
• $ef \in \mathscr{T}_\mathscr{E}$	$e, f \in \mathscr{T}_\mathscr{E}$

Two edgeterms are equivalent if one can be transformed into the other by:

$$e(fg) \equiv (ef)g \qquad \varepsilon e \equiv e \equiv e\varepsilon \qquad [\varepsilon\rangle^A \equiv \varepsilon \equiv \langle\varepsilon]^A \tag{3}$$

Definition 2.4. Given a !-graph with simple overlap G, a *neighbourhood order* on G is a function $\mathrm{nhd} : N(G) \to \mathscr{T}_{W(G)}$ satisfying $\forall N \in N(G)$:

- $\mathrm{nhd}(N)$ is an edgeterm with input edge names $\{w \in W(G) : w \to N\}$ and output edge names $\{w \in W(G) : N \to w\}$,
- The !-boxes with edges to $a \in \mathrm{nhd}(N)$ but not to N are precisely those containing a in $\mathrm{nhd}(N)$ (ordered by nesting order).

We define three !-box operations: Kill_B removes B and its contents; Drop_B removes B leaving its contents; and Copy_B add a new copy of B and its contents. The operation Exp_B can be applied using Copy_B and then dropping the new !-box.

Definition 2.5. For Op_B any !-box operation, $\mathrm{Op}_B(G,\mathrm{nhd}) := (\mathrm{Op}_B(G), \mathrm{Op}_B \circ \mathrm{nhd})$. Where our three !-box operations are applied to !-graphs by:

- $\mathrm{Copy}_B(G)$ is defined by a pushout of inclusions:

$$\begin{array}{ccc} G\backslash C(B) & \hookrightarrow & G \\ \downarrow & & \downarrow \\ G & \longrightarrow & \mathrm{Copy}_B(G) \end{array}$$

- $\mathrm{Drop}_B(G) := G\backslash B$
- $\mathrm{Kill}_B(G) := G\backslash C(B)$

and Op_B is applied to an edgeterm in the following trivial cases:

$$\mathrm{Op}_B([e\rangle^A) := [\mathrm{Op}_B(e)\rangle^A \qquad \mathrm{Op}_B(ef) := \mathrm{Op}_B(e)\,\mathrm{Op}_B(f)$$
$$\mathrm{Op}_B(\langle e]^A) := \langle \mathrm{Op}_B(e)]^A \qquad \mathrm{Op}_B(x) := x$$

where $A \neq B$ and $x \in \{\check{a}, \hat{a}, \varepsilon\}$ and for the final two cases:

$$\mathrm{Copy}_B([e\rangle^B) := [e\rangle^B[\mathbf{fr}(e)\rangle^{\mathbf{fr}(B)} \qquad \mathrm{Drop}_B([e\rangle^B) := e \qquad \mathrm{Kill}_B([e\rangle^B) := \varepsilon$$
$$\mathrm{Copy}_B(\langle e]^B) := \langle \mathbf{fr}(e)]^{\mathbf{fr}(B)}\langle e]^B \qquad \mathrm{Drop}_B(\langle e]^B) := e \qquad \mathrm{Kill}_B(\langle e]^B) := \varepsilon$$

3 Encoding !-tensors

3.1 Defining the mapping $\mathscr{I}$

We wish to allow theories with generating morphisms which have input or output edges of single ($\bullet$-tagged) or variable (∞-tagged) arity. Given a set $\mathscr{O}$ of object types and $X \in \mathscr{O}$ we write $\check{X}^\bullet, \hat{X}^\bullet, \check{X}^\infty, \hat{X}^\infty$ to represent fixed arity input and output edges and variable arity input and output edges respectively. We need a different definition of a signature when working with !-tensors.

Definition 3.1. A *compact closed signature* Σ consists of a set $\mathscr{O} := \{X, Y, \ldots\}$ of object types and a set $\mathscr{M}$ of pairs (ψ, w), where w is a word in $\{\check{X}^\bullet, \hat{X}^\bullet, \check{X}^\infty, \hat{X}^\infty : X \in \mathscr{O}\}$.

Recall the definition of a !-tensor expression and equivalence of !-tensors from [12].

Definition 3.2. The set of all !-tensor expressions $\mathscr{T}_\Sigma$ for a signature Σ is defined recursively as:

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

Where $\mathscr{E}$ is a set of possible edge names and $\mathscr{B}$ is a set of possible !-box names. Subject to the conditions (F1) $\check{a}$ and $\hat{a}$ must occur at most once for each edge name a and (F2) $[\ldots]^A$ must occur at most once for each !-box name A, as well as some consistency conditions (C1)-(C3) for !-boxes [12].

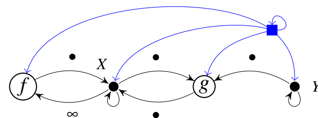
Definition 3.3. Two !-tensor expressions G and H are considered equivalent, $G \equiv H$, if one can be obtained from the other by replacing bound names and/or applying one or more of the following enforced identities:

$$(GH)K \equiv G(HK) \qquad GH \equiv HG \qquad G1 \equiv G$$
$$\mathrm{Ins}_{B \ni 1_{\hat{b}\check{a}}}(G) \equiv G[\check{b} \mapsto \check{a}] \qquad \mathrm{Ins}_{B \ni 1_{\hat{a}\check{b}}}(H) \equiv H[\hat{b} \mapsto \hat{a}]$$

where for the last two identities $\check{b}$ and $\hat{b}$ are free in G and H respectively. Note Ins is inserting the identity wires inside the appropriate !-box B such that the edge b is well defined.

Definition 3.4. Given a compact closed signature $\Sigma = (\mathscr{O}, \mathscr{M})$, we define the corresponding compressed monoidal signature $\mathscr{I}(\Sigma)$ to have objects $\mathscr{O}$, morphisms $\{\phi : (\phi, w) \in \mathscr{M}\}$, dom is defined to map ϕ to the input edges of w for each $(\phi, w) \in \mathscr{M}$ and similarly cod maps ϕ to the output edges of w.

For example, the !-tensor signature Σ with objects $\{X, Y\}$ and morphisms $\{(f, \hat{X}^{\bullet}\check{X}^{\infty}), (g, \hat{X}^{\bullet}\check{Y}^{\bullet}\check{X}^{\bullet})\}$ becomes the !-graph signature with the two morphisms $f : [X^{\infty}] \to [X^{\bullet}]$ and $g : [Y^{\bullet}, X^{\bullet}] \to [X^{\bullet}]$. This results in a !-typegraph $\mathscr{G}_{\mathscr{I}(\Sigma)!}$ of the form:



For the rest of this section we will suppose we are working in a fixed signature $\Sigma = (\mathscr{O}, \mathscr{M})$.

The !-tensor formalism avoids the need to name nodes, but we will need to assign them with unique names if we want to convert to !-graphs. We do this using an indexing set. Say G is a !-tensor and let J be a set indexing the nodes (subexpressions of the form ϕ_e, $1_{\hat{b}\check{a}}$ or 1) of G, write N_j for the node corresponding to $j \in J$. In converting to a !-graph, the node N_j can be labelled j.

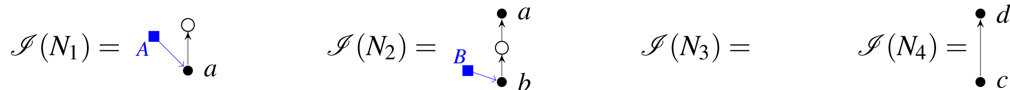
We can define a function $\mathscr{I}$ taking !-tensors to what we will prove to be !-graphs with simple overlap and neighbourhood orders:

Definition 3.5. $\mathscr{I}$ taking !-tensor expressions to !-graphs with neighbourhood orders is defined recursively:

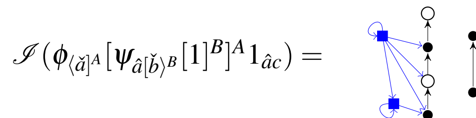
- $\mathscr{I}(1) := \{\}$
- $\mathscr{I}(1_{\hat{b}\check{a}}) := \{a \to b\}$
- $\mathscr{I}(\phi_e = N_j) := \{a \to j : \check{a} \in e\} \cup \{j \to a : \hat{a} \in e\} \cup \{B \to a : B \in \mathrm{ectx}_e(a)\}$
- $\mathscr{I}(GH) := \mathscr{I}(G) \cup \mathscr{I}(H)$
- $\mathscr{I}([G]^B) := \mathscr{I}(G) \cup \{B \to x : x \in U(\mathscr{I}(G))\} \cup \{B \to B' : B' \leq B\}$

The typing function on vertices maps $j \to \phi$ where $N_j = \phi_e$, !-boxes to ! and edge names to their edge type. The neighbourhood order is $nhd(j) := e$ for each $N_j = \phi_e$.

So, for example, from the tensor expression $\phi_{\langle\check{a}]^A}[\psi_{\hat{a}[\check{b}\rangle^B}[1]^B]^A 1_{\hat{d}\check{c}}$ if we index the nodes from left to right using $\{1,2,3,4\}$ then we get the following after $\mathscr{I}$:



Going through the recursive definition these combine to the !-graph we would expect:



3.2 $\mathscr{I}$ is Well-Behaved

We now check $\mathscr{I}$ behaves as desired so we can use this mapping to encode !-tensors in Quantomatic.

Theorem 3.6. *Given $G \in \mathscr{T}_\Sigma$, $\mathscr{I}(G)$ is a $\mathscr{G}_{\mathscr{I}(\Sigma)!}$-typed !-graph with simple overlap and a neighbourhood order.*

Proof. Let J index the nodes in G.

- We first go through the four conditions of Definition 1.8 to check we have a $\mathscr{G}_{\mathscr{I}(\Sigma)!}$-typed !-graph:
  - BG1. Take (ϕ, w) a morphism in $\mathscr{I}(\Sigma)$. The typegraph $\mathscr{G}_{\mathscr{I}(\Sigma)}$ contains a node ϕ with $\bullet$-edges to/from the $\bullet$-tagged edges in w. Given $\phi_e = N_j$, the $\bullet$-edge neighbourhood of j is the finite arity edges in e which are the same as $\bullet$-tagged edges in w. Hence we have a bijection. From the definition of $\mathscr{I}$, wire vertices come directly from edge names. The unique (possible) occurrence of $\hat{a}$ results in a wire $_ \to a$ and the unique (possible) occurrence of $\check{a}$ results in a wire $a \to _$. Hence each a must have at most one incoming and one outgoing edge.
  - BG2. The full subgraph $!(G)$ is the reflexive, transitive closure of the 'child of' relation on !-boxes. Hence it is reflexive, transitive and antisymmetric (ensuring 'child of' is cycle-free).
  - BG3. Take $B \in !(G)$ and write X for $U(C(B))$, so we need to show that X is an open subgraph of $U(G)$. We first show that any wire-vertex adjacent to a vertex in X is in X. From the definition of $\mathscr{I}$, the only time an edge is added from a box-vertex to a node-vertex j, there are also edges added to each neighbour of j. Any two adjacent wire-vertices must come from $\mathscr{I}(1_{\hat{a}\check{b}})$, so that $B \to a \Leftrightarrow B \to b$. Incident edges can only come from node-vertices not in X with adjacent wire-vertices in X. Hence, the wire comes from a directed edge with B in its edge context which means it is ∞-tagged in the typegraph.
  - BG4. An edge between box-vertices, $A \to B$ must be added by $\mathscr{I}([H]^A)$ during the recursive definition, where B and $C(B)$ are already in $\mathscr{I}(H)$. Hence we get edges $A \to x$ for all $x \in C(B)$ meaning $C(B) \subseteq C(B')$.
- Next we check that any overlap of non-nested !-boxes B, B' is simple.

  Suppose j is a node-vertex such that $B \to j$ and $B' \to j$. From the definition of $\mathscr{I}$ we see that B and B' must appear in the node context of N_j and so one is nested inside the other. This contradiction proves that $C(B) \cap C(B')$ contains only wire vertices

  Suppose a is a wire-vertex such that $B \to a$ and $B' \to a$. If a is adjacent to another wire-vertex b then this edge must have come from $1_{\hat{b}\check{a}}$ or $1_{\hat{a}\check{b}}$ so a and b are nested inside the same !-boxes. Hence the intersection $C(B) \cap C(B')$ contains only the interiors of zero or more wires.

  Suppose both endpoints are wire-vertices. Hence the edges have come from identity wires $1_{\hat{b}\check{a}}$ which must have B and B' in their node context which would require them to be nested.

  Suppose a wire-vertex $a \in C(B) \cap C(B')$ is adjacent to the node-vertex j. If j was not in $C(B)$ nor $C(B')$ then both B and B' must occur in the edge context of a and hence would be nested. We conclude that j must occur in exactly one of $C(B)$ or $C(B')$.
- Finally we wish to check that nhd is a neighbourhood order on $\mathscr{I}(G)$. For $j \in N(\mathscr{I}(G))$ the incoming edges a to j come from inputs $\check{a}$ and the outgoing edges b come from outputs $\hat{b}$ as required. Also, for the node $N_j = \phi_e$, !-boxes with edges to $a \in \mathrm{ectx}\, e$ but not to j are exactly those which appear in $\mathrm{ectx}_e(a) = \mathrm{ectx}_{\mathrm{nhd}(\phi)}(a)$.

$\square$

Hence $\mathscr{I}$ takes a !-tensor expression and returns a correctly typed !-graph with a neighbourhood order. By the following theorem, the definition of $\mathscr{I}$ can be lifted from specific !-tensor expressions to !-tensors (equivalence classes of !-tensor expressions).

Theorem 3.7. *$\forall G, H \in \mathscr{T}_\Sigma$, $G \equiv H$ if and only if $\mathscr{I}(G)$ and $\mathscr{I}(H)$ are equivalent up to renaming and wire homeomorphisms.*

Proof. We prove first that all !-tensor equivalences from Definitions 2.3 and 3.3 are preserved through $\mathscr{I}$. In both formalisms bound variables can be renamed freely, so we will not worry about names here.

- Since edgeterms are copied directly from !-tensors to !-graphs, the edgeterm equivalences are still preserved. The associativity, commutativity and identity equivalences on !-tensor expressions have no affect on the graphical formalism so these are also preserved. The only thing left to check are the two equivalences involving inserting identity wires $1_{\hat{b}\check{a}}$. These two conditions come down to wire homeomorphism in the !-graph framework. First we are given that $\check{b}$ exists in G and (for some !-box B) we look at $\mathrm{Ins}_{B \ni 1_{\hat{b}\check{a}}}(G)$ under $\mathscr{I}$. This becomes a graph with edges $a \to b \to x$ for some x and since a and b are both wire-vertices this is wire homeomorphic to $G[\check{b} \mapsto \check{a}]$. The other case is similar but with arrows reversed.

- For the other direction suppose $\mathscr{I}(G)$ and $\mathscr{I}(H)$ differ only by a single wire homeomorphism, so there exist wire-vertices a, b and a vertex c in $\mathscr{I}(G)$ with $a \to b \to c$, and $\mathscr{I}(H)$ is $\mathscr{I}(G)$ but with $a \to b \to c$ replaced with $a \to c$. From the definition of $\mathscr{I}$ we see that there exists some G' and !-box B such that $G = \mathrm{Ins}_{B \ni 1_{\hat{b}\check{a}}}(G')$ (this is the only way two wire vertices can be connected) and also $\check{b}$ must exist in G. H must be the same as G' except the edge $\check{b}$ is replaced by the edge $\check{a}$.

$$\begin{aligned} G &\equiv \mathrm{Ins}_{B \ni 1_{\hat{b}\check{a}}}(G') \\ &\equiv G'[\check{b} \mapsto \check{a}] \\ &\equiv H \end{aligned}$$

□

We have shown $\mathscr{I}$ is injective on !-tensors and hence is a bijection onto its image. We can hence take any !-tensor G and work with it in Quantomatic in the form of $\mathscr{I}(G)$. To work as an encoding we would hope that !-box operations are equivalent in each formalism. Applying the !-box operation Op_B to $\mathscr{I}(G)$ and then returning to the !-tensor formalism should result in a !-tensor equivalent to $\mathrm{Op}_B(G)$. By the previous theorem we need only check that $\mathrm{Op}_B \circ \mathscr{I} = \mathscr{I} \circ \mathrm{Op}_B$ and it then follows that $\mathscr{I}^{-1} \circ \mathrm{Op}_B \circ \mathscr{I} = \mathrm{Op}_B$.

Theorem 3.8. $\mathrm{Op}_B(\mathscr{I}(G)) = \mathscr{I}(\mathrm{Op}_B(G))$ *for any !-box operation* Op_B *and* $G \in \mathscr{T}_\Sigma$.

Proof. This can be shown by case analysis on the recursive definition of $\mathscr{T}_\Sigma$ going through each operation Copy_B, Drop_B, Kill_B. Most cases are trivial, the interesting case is showing $\mathrm{Op}_B(\mathscr{I}([G]^B)) = \mathscr{I}(\mathrm{Op}_B([G]^B))$ for each operation:
$\mathrm{Copy}_B(\mathscr{I}([G]^B))$ is defined by a pushout of the inclusion $1 \hookrightarrow \mathscr{I}([G]^B)$ with itself, so equals the disjoint

union of two copies of $\mathscr{I}([G]^B)$. Hence (for **rn** a renaming function):

$$\begin{aligned}
\text{Copy}_B(\mathscr{I}([G]^B)) &= \mathscr{I}([G]^B) \cup \mathscr{I}([\mathbf{rn}(G)]^{\mathbf{rn}(B)}) \\
&= \mathscr{I}([G]^B[\mathbf{rn}(G)]^{\mathbf{rn}(B)}) \\
&= \mathscr{I}(\text{Copy}_B([G]^B)) \\
\text{Drop}_B(\mathscr{I}([G]^B)) &= \text{Drop}_B(\mathscr{I}(G) \cup \{B \to x : x \in U(\mathscr{I}(G))\} \cup \{B \to B' : B' \leq B\}) \\
&= \mathscr{I}(G) \\
&= \mathscr{I}(\text{Drop}_B([G]^B)) \\
\text{Kill}_B(\mathscr{I}([G]^B)) &= \text{Kill}_B(\mathscr{I}(G) \cup \{B \to x : x \in U(\mathscr{I}(G))\} \cup \{B \to B' : B' \leq B\}) \\
&= \{\} \\
&= \mathscr{I}(1) \\
&= \mathscr{I}(\text{Kill}_B([G]^B))
\end{aligned}$$

□

Corollary 3.9. *The concrete instances of $\mathscr{I}(G)$ are the concrete instances of G with $\mathscr{I}$ applied to them.*

4 Future work

4.1 Theory

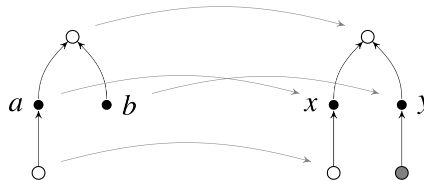
This paper demonstrates work in progress and there are a few things which need to be added to the theory. Firstly, we would like to show that $\mathscr{I}$ is a bijection on to the set of !-graphs with simple overlap and neighbourhood order. This would allow us to replace Theorem 3.6 with an if and only if statement:

Conjecture 4.1. $\exists G \in \mathscr{T}_\Sigma$ such that $\mathscr{I}(G) = (X, \text{nhd})$ if and only if all of the following hold:

- X is a $\mathscr{G}_{\mathscr{I}(\Sigma)!}$-typed !-graph
- X has simple overlap
- nhd is a neighbourhood order on X

Related to this, we would like to explore the differences between simple overlap and the stricter property of trivial overlap of !-boxes [14].

Matching (finding occurrences of one !-graph inside another) and substitution (replacing matches with equivalent graphs) are the key parts of diagrammatic reasoning with !-graphs. Matching a string graph G with a string graph K means finding a monomorphism from G in to K showing that G is isomorphic to a subgraph of K. For example, in a commutative theory the following is a string graph matching:



This may not be the case if we are working with non-commutative nodes. If the top node-vertex in G has edges ordered ab and the top node-vertex in K has edges ordered yx then we can not find a matching. This can be seen from their !-tensor equivalents:

Hence we hope to define a matching from (G, nhd_G) to (K, nhd_K) to be a !-graph matching $i : G \to K$ satisfying $i(\text{nhd}_G(x)) = \text{nhd}_K(i(x))\ \forall x \in N(G)$. This is currently being looked into by the author, along with checking that substitution works without any changes.

Once matching and substitution are formalised we can start comparing the differences between rewriting using substitution in !-graphs with neighbourhood orders and !-logic as defined in [13]. Ideally we would like to demonstrate that any case of proof by substitution of !-graphs (with neighbourhood order) can be shown in !-logic for !-tensors and conversely that each rule in !-logic can be implemented using substitution.

4.2 Implementation

The key contribution of this paper is to demonstrate that !-tensors could be quickly implemented in a program such as Quantomatic by adding neighbourhood order data to node vertices. Hence an obvious piece of further work would be to actually implement this in Quantomatic. There are three key areas to work on here. First is to code the new neighbourhood data on the !-graph datatype. Secondly we should update the matching and substitution algorithms to respect neighbourhood orders. Finally the user needs to be able to interact with !-graphs in a new way allowing neighbourhood orders to be chosen.

We add neighbourhood orders by recursively defining an edgeterm datatype (using the set of edge and !-box names) and then adding a node-vertex indexed table of edgeterms to the !-graph datatype. Code can be written to enforce the restrictions of Definition 2.4 whenever a !-graph is updated. The functions acting on !-graphs can then be extended to also work on the individual edgeterms. For example !-box operations should be defined to act on edgeterms as described in Definition 2.5.

Once the theory has been checked, the matching and substitution algorithms can be updated to test orders on neighbourhoods are preserved. A naive approach to matching would be to look for all matches ignoring edgeterms and then to filter out those in which edgeterms do not match up. This would be easy to implement but not very efficient. More work is needed to find a better method, checking edgeterms as the matching algorithm searches. As for substitution, the requirement that two !-tensors in a !-tensor equation have compatible boundaries should make sure that substitution does not create any issues.

We present a planned graphical user interface for !-tensors in Quantomatic in Appendix A.

Since non-commutativity is added to each node-vertex individually it may be possible for Quantomatic to have commutative and non-commutative nodes interacting in one diagram. This is something which needs to be explored further.

References

[1] John C. Baez & Jason Erbele (2014): *Categories in Control*. Technical Report, arXiv:1405.6881.

[2] F. Bonchi, P. Sobocinski & F. Zanasi (2014): *A categorical semantics of signal flow graphs*. In: *CONCUR'14: Concurrency Theory.*, *Lecture Notes in Computer Science* 8704, Springer, pp. 435–450.

[3] B. Coecke (2009): *Quantum Picturalism*. *Contemporary Physics* 51, pp. 59–83. arXiv:0908.1787.

[4] B. Coecke & R. Duncan (2008): *Interacting quantum observables*. In: *Proceedings of the 37th International Colloquium on Automata, Languages and Programming (ICALP)*, Lecture Notes in Computer Science.

[5] B. Coecke, R. Duncan, A. Kissinger & Q. Wang (2012): *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. ArXiv:1203.4988.

[6] Lucas Dixon & Ross Duncan (2009): *Graphical Reasoning in Compact Closed Categories for Quantum Computation*. *AMAI* 56(1), p. 20, doi:10.1017/S0305004100074338.

[7] Andre Joyal & Ross Street (1991): *The geometry of tensor calculus I*. *Advances in Mathematics* 88, pp. 55–113, doi:10.1016/0001-8708(91)90003-P.

[8] D. Kartsaklis (2014): *Compositional Distributional Semantics with Compact Closed Categories and Frobenius Algebras*. Ph.D. thesis, University of Oxford.

[9] Aleks Kissinger (2014): *Abstract Tensor Systems as Monoidal Categories*. In C Casadio, B Coecke, M Moortgat & P Scott, editors: *Categories and Types in Logic, Language, and Physics: Festschrift on the occasion of Jim Lambek's 90th birthday*, *Lecture Notes in Computer Science* 8222, Springer, doi:10.1007/978-3-642-54789-8_13. arXiv:1308.3586 [math.CT].

[10] Aleks Kissinger, Alex Merry & Matvey Soloviev (2012): *Pattern Graph Rewrite Systems*. In: *Proceedings of DCM 2012*, *EPTCS* 143, doi:10.4204/EPTCS.143.5. arXiv:1204.6695 [math.CT].

[11] Aleks Kissinger, Alexander Merry, Lucas Dixon, Ross Duncan, Matvey Soloviev & Benjamin Frot (2011): *Quantomatic*. https://sites.google.com/site/quantomatic/.

[12] Aleks Kissinger & David Quick (2014): *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.*, pp. 56–67, doi:10.4204/EPTCS.172.5. arXiv:1412.8552 [cs.LO].

[13] Aleks Kissinger & David Quick (2015): *A first-order logic for string diagrams*. arXiv:1505.00343 [math.CT].

[14] Aleks Kissinger & Vladimir Zamdzhiev (2015): *!-Graphs with Trivial Overlap are Context-Free*. doi:10.4204/EPTCS.181.2. ArXiv:1501.06059.

[15] Alexander Merry (2014): *Reasoning with !-Graphs*. Ph.D. thesis, University of Oxford.

[16] R. Penrose (1971): *Applications of negative dimensional tensors*. In: *Combinatorial Mathematics and its Applications*, Academic Press, pp. 221–244.

A Graphical user interface

Implementing the gui could require a large amount of work. The current method for drawing edges simply draws a line directly between nodes. This will need to be changed for non-commutative nodes where it becomes important where edges enter/leave a node. New edges will need to be curves defined by their source and target nodes and angles. We will add handles to each edge controlling which direction the edge should go. Figure 2 demonstrates what this may look like. We envisage Quantomatic choosing the angles initially and only letting the user drag one edge handle over another edge to reorder them. If the user wished to have more control, double clicking the handle would allow them to control the angle exactly. The user should also be able to choose the angle of the tick.

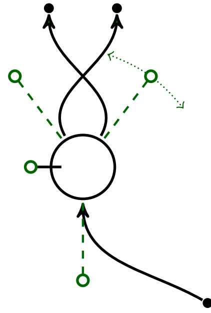


Figure 2: Reordering edges

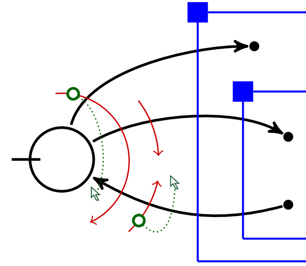


Figure 3: Manipulating arcs

Figure 3 demonstrates the second major interface modification. We need to draw and let the user interact with arcs over edges entering a !-box. Whenever the user draws an edge between a non-commutative node and a !-box, Quantomatic will automatically add an arc over the edge (grouping multiple edges if possible). To change the arcs the user should be able to click on any part of an arc and drag. This would start a new arc containing every edge the user drags over in the direction of dragging. Hence arcs over multiple edges can be split apart by drawing each single arc in its required direction.

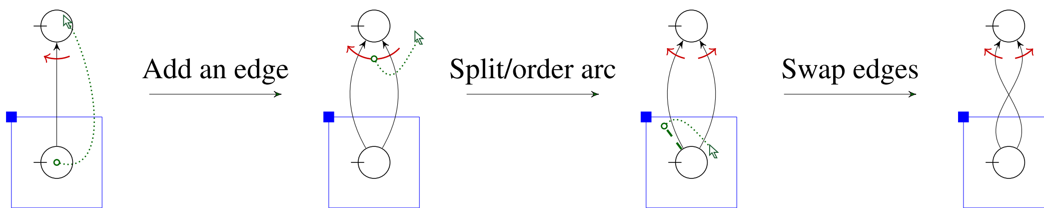


Figure 4: An example workflow in Quantomatic