Computer representation of string diagrams

Jules Hedges   Jelle Herold

June 28, 2019

1 Introduction

The importance of string diagrams can be expressed formally by the fact (better known as the Joyal-Street coherence theorem [JS91]) that they are a presentation of *free monoidal categories*. More precisely, the free monoidal category on a collection of generating objects and a collection of generating morphisms, is equivalent to a category whose morphisms are isotopy classes of string diagrams with string labelled by generating objects and nodes labelled by generating morphisms. The universal property of freeness states that if each generating object and generating morphism is associated to an object and morphism in any target 'semantic' monoidal category, then any diagram induces a canonical morphism, its 'semantics', which is isotopy-invariant.

(Side note: In general, freeness is a property of *syntax*, and string diagrams are a 2-dimensional syntax. Decategorifying, the free *monoid* on a set of generating symbols is the monoid of *strings* of symbols (also known as lists), which are used as syntax in ordinary (1-dimensional) algebra.)

By instantiating with an example, this becomes much less abstract. Suppose that our target category is $\mathbf{Mat}_{\mathbb{R}}$, the category whose objects are natural numbers and whose morphisms $m \to n$ are $m \times n$ matrices, with matrix multiplication as composition. Suppose we take a single generating object 1, and take each $m \times n$ matrix as a generating morphism of type $1^{\otimes m} \to 1^{\otimes n}$. The Joyal-Street coherence theorem then states that any string diagram in which every node is labelled by a matrix (whose dimensions match the node's input and output degree) determines a matrix, which is isotopy-invariant.

There is moreover a simple procedure to calculate this matrix. We recursively cut the diagram horizontally and vertically, and stop when each region is indivisible, containing either a single node, or a single string, or a single string crossing. Then we associate each indivisible region to the associated semantics (respectively either the labelling matrix, or the 1×1 identity matrix (1), or the swap matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$), and combine them using matrix multiplication and Kronecker product according to the horizontal and vertical cuts we made. The Joyal-Street theorem states that the resulting matrix is independent of our choices of cuts. Moreover, although we illustrated the procedure with matrices, the procedure is the same in any monoidal category.

***

There are, broadly, two ways of presenting free monoidal categories. The first is categories of string diagrams modulo isotopy, and the second is as categories of linear logic proof trees modulo rewrites. The former is far more useful for calculating with, but the second is finitistic and much easier to implement in a computer. We propose a bridge between the two via *kd-trees*, a data structure from computer graphics that can be used to represent string diagrams in a way that coincides with proof trees.

1.1 Computer implementation of string diagrams

The need for computer assistance has been widely discussed in the fledgeling Applied Category Theory community, for example at the first ACT conference in Leiden in 2018, and on the ACT Google Groups mailing list.

There are several existing tools for working with string diagrams that are classified as *proof assistants*. Quantomatic and pyZX are tools for simplifying diagrams in the language of ZX calculus, which internally use a graphical representation (i.e. based on graphs) and are essentially specialised graph rewrite engines. By using a graphical representation, these tools are essentially specialised to compact closed categories, if not to ZX calculus specifically.

Homotopy.io and its predecessor Globular are tools for constructing morphisms in higher categories using their associated diagrams.

What is conspicuously absent is any tool that can be classified as a *compiler*, for working with the semantics rather than the syntax of string diagrams. Consider, for example, the prop $\mathbf{Mat}_{\mathbb{R}}$ whose objects are natural numbers and whose morphisms $m \to n$ are $m \times n$ real matrices. With such a tool, the user would create a diagram using a graphical user interface in which every atomic block is labelled by a matrix of the appropriate dimensions. The user will then push a button and, if the diagram is well-formed, the software will compute the matrix represented by the diagram (or generate code that will do so, for example in numPy).

In general, (applied) category theory can make connections between disparate application areas by finding commonalities between their graphical languages. A common front end user interface that is able to support only a small number of topological flavours of string diagrams (for example monoidal, symmetric monoidal, traced monoidal, compact closed) together with customisable generating morphisms will be applicable to many different domain-specific backends. In each case, the backend must supply a 'meaning' to a morphisms as a representation in the computer, that supports the operations needed to interpret the relevant class of string diagrams (chiefly, sequential and parallel composition operators). For example, in the case of matrix algebra, each morphism represents a matrix, and sequential and parallel composition correspond semantically to matrix multiplication and Kronecker product.

A diagram can generally be 'compiled' in many different ways, corresponding to different ways of 'carving up' the diagram using horizontal and vertical cuts, and also by doing topological moves on the diagram. The axioms of a monoidal category (for example) state precisely that all of these different ways will produce the same end result.

However, they may be vastly different from a computational perspective. Taking the example of matrices again, Kronecker product is exponentially more expensive than matrix multiplication, and so a good heuristic will be to manipulate the diagram in order to minimise the number of parallel compositions.

In the end, any general-purpose software will need to be scriptable to allow domain-specific heuristics. As a starting point, however, any compilation method will be suitable for small diagrams in every application domain.

2 Graphs

Probably the most obvious way to represent a string diagram is as a graph, which we will call a *graphical* representation. Of course, the representation of graphs in a computer is extremely well-studied with efficient implementations available in every serious programming language. Probably the most common representation is *adjacency lists*, where the graph is represented by a list of vertices, where each vertex is associated to a list of adjacent vertices. (Note however that representation of graphs in purely functional programming languages is notoriously tricky, with the Haskell's Algebraic Graph Library [Mok17] being a recent development.)

Graphical representations are used by the Quantomatic [KZ15] and PyZX [KvdW] proof assistants for ZX calculus.

Graphs are certainly reasonable for use-cases that are specialised to compact closed categories (as the above examples are), since any node may be connected to any other. However, it is also easy to generalise from compact closed to symmetric monoidal categories, by restricting to directed acyclic graphs.

Certainly there are many examples of compact closed categories of importance in applied category theory. Examples include the ZX calculus used in categorical quantum mechanics [CK17], the category of finite dimensional vector spaces used in distributional semantics [CSC10], and categories of decorated cospans [Fon15] and corelations [Fon18].

However, while compact closure is a 'gold standard' for categorical structure, there are also important monoidal categories that are not compact closed, for example those used in dataflow [Del], classical computing [Pav13] (including the Statebox visual programming language [Tea19]), and compositional game theory [GHWZ18]. We note however that all of these examples are symmetric monoidal categories. (Non-symmetric monoidal categories find application for example in knot theory and parts of theoretical physics, but these are certainly the exception.)

The formal definition of string diagrams [JS91] is actually via *topological graphs*, which are Hausdorff spaces that are 1-dimensional manifolds plus singular points representing the nodes (which is equivalently a 1-dimensional simplicial complex [Hat02, section 1.A]), equipped with an embedding in the plane. (Notice that an embedding is by definition injective, which forces planarity.)

An important point to keep in mind is that when drawing string diagrams in practice we nearly always abuse notation, and draw the singular points as regions (of various shapes, which can themselves carry information [CK17, section 4.3.2]). For example,

instead of the graph
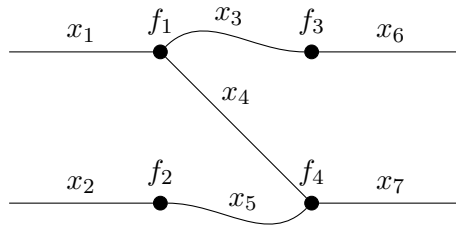
we would typically draw
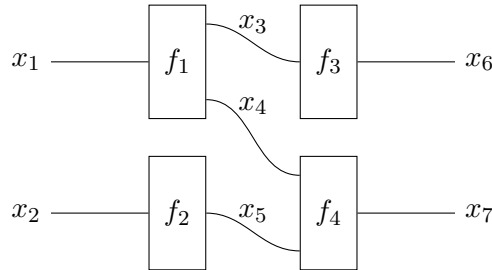
This can be thought of as infinitesimal information. To our knowledge, this has not been formalised using differential topology.

3 Planar graphs

Given that most of the categories we care about are symmetric, why should we care about planar graphs? The answer is that the symmetry of a monoidal category can be computationally nontrivial, and we may want finer control over it. So, even if our target category is symmetric, we may want to represent its diagrams as planar graphs together with a special type of node representing a string crossing.

- [Tam96]

- [HR17]

- [Kar18] (maybe not useful, focus on edge contraction)

# 4 Polyhedral complexes

5 Pasting diagrams

The Poincaré dual of a string diagram is a *pasting diagram*. There are different flavours of pasting diagram, corresponding to different flavours of 2-category.

If we view our strict monoidal category as a 2-category with 1 object, we can also represent the same morphism by a *globular pasting diagram*:
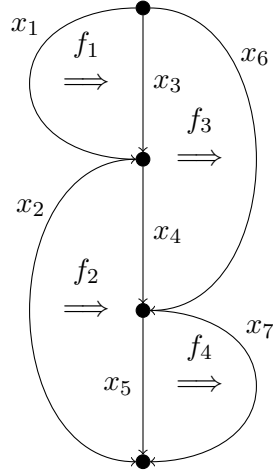
Globular pasting diagrams are unwieldy to work with for any example much larger than this one, and in the authors' opinions they are also less intuitive in practice, in spite of the fact that it seems more intuitive that 2-cells go between 1-cells just as morphisms go between objects. Indeed, the introduction of string diagrams over the older pasting diagrams is intimately connected with the emergence of modern applied category theory.

However, a strict monoidal category can also be viewed not only as a 2-category with 1 object, but also as a double category with 1 0-cell and only the identity horizontal 1-cell. These *cubical pasting diagrams* overcome the practical shortcomings of globular pasting diagrams with respect to actually drawing them on a page:
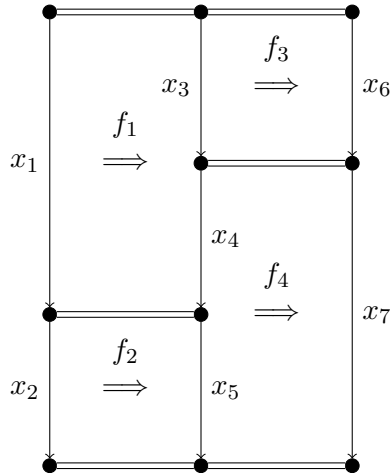
As a further piece of evidence in favour of cubical diagrams being more intuitive, we note that Joyal and Street's original coherence proof for monoidal categories [JS91] proceeds by division of the plane into rectangular regions, even though globular regions would have sufficed.

6 Tilings

We make some minor notational changes to the previous cubical pasting diagram, to make it visually cleaner:
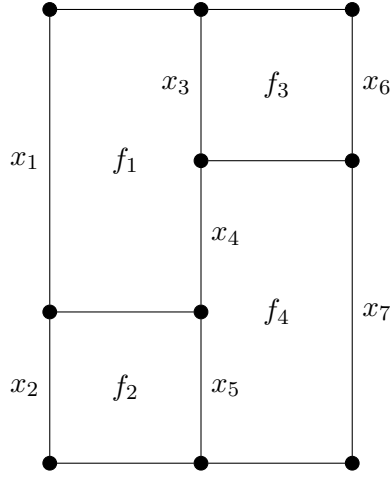
Geometrically, this is a *rectangular tiling* of the plane.

7 K-d trees

8 Other representations

9 String diagrams, pasting diagrams and brick diagrams

An issue with this representation is that we need to treat the monoidal unit as a special case, as a labelled edge that can always be introduced or cancelled (or give up the idea of using rectangular tilings, which will lead to new difficulties). We propose the following notation, in which we take an additional Poincaré dual of only the vertical boundaries:
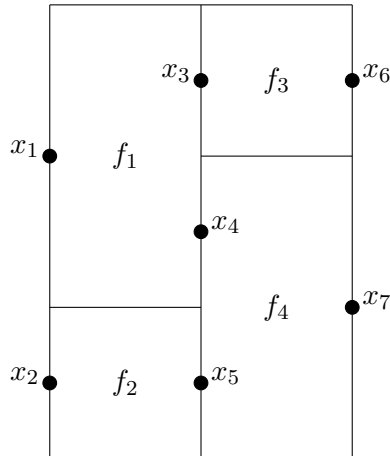
We refer to this as a *brick diagram*.

10 Tilings

A *tile diagram* is a division of the square $[0,1]^2$ into rectangles. We restrict to tile diagrams that do not contain a *pinwheel*. In [Daw95] it is shown that this restriction is equivalent to being *binarily composable*, which is to say, it can be described by an expression that builds it from atomic parts using sequential and parallel composition. (Such an expression is, of course, usually not unique.)

We work in the context of a monoidal signature, i.e. a set $Ob(\Sigma)$ of *object symbols*, a set $Mor(\Sigma)$ of *morphism symbols*, and for every morphism symbol f a pair of words $s(f), t(f)$ over $Ob(\Sigma)$, the *source* and *target* of f.

A *brick diagram* is a tile diagram together with:

- To each minimal rectangle, an assigned morphism symbol, or the special symbol *id*

- To each minimal vertical edge, an assigned word of object symbols

- Such that for each minimal rectangle, concatenating together all the words along its entire left (right) edge yields its source (target)

Each minimal rectangle in a brick diagram is called a *brick*, and should be thought of like a 2-dimensional Lego brick: its top and bottom edges have a pattern of studs and holes specified by its source and target, whereas its side edges are smooth. This means that brick diagrams are interpreted in monoidal categories rather than double categories; however, it should be straightforward to generalise to double categories.

11 Proof trees and kd-trees

Let Σ be a monoidal signature. We consider the noncommutative linear logic of $\otimes$ only, with elements of $Ob(\Sigma)$ as atomic propositions, and the following proof rules:

$$(id)\frac{}{x \vdash x}(x \in Ob(\Sigma)) \qquad\qquad (f-ax)\frac{}{s(f) \vdash t(f)}(f \in Mor(\Sigma))$$

$$(\otimes - intro)\frac{a \vdash b \qquad a' \vdash b'}{aa' \vdash bb'} \qquad\qquad (cut)\frac{a \vdash b \qquad b \vdash c}{a \vdash c}$$

The free monoidal category on Σ can be presented as the category whose objects are words over $Ob(\Sigma)$ and whose morphisms $a \to b$ are particular equivalence classes of proof trees with conclusion $a \vdash b$. The equivalence relation that must be imposed is essentially generated by the axioms required of a monoidal category, known in proof theory as *commuting conversions*.
***

As a bridge between proof trees and diagrams, we consider *binary space partition trees*, a common data structure in computational geometry. A binary space partition

tree is a recursive partitioning of a space by dividing hyperplanes. Specifically, we focus on *kd-trees*, the special case in which the separating hyperplanes are all aligned to the coordinate axes. We also focus on the 2-dimensional case.

We use a flavour of kd-trees in which leaves are labelled by regions, and nodes are labelled by partitions between regions. (This is a hybrid of kd-trees with nodes labelled by partitions [dBvKOS97] and those with leaves labelled by regions [Hou87].)

K-d trees are not unique, and it seems obvious to conjecture that equivalences of k-d trees (which are well studied for the purpose of balancing) are the same as equivalences of proof trees. (The same idea may also work in higher dimensions, but it is less obvious that nothing goes wrong, since higher dimensional diagrams are generally interpreted in semi-strict rather than strict *n*-categories.)

12 Diagrams with cursors

13 Rendering string diagrams

We do not expect many people (if any) to convert from working with string diagrams to tile diagrams day-to-day, and a software interface that forces tile diagrams on the user may put off many potential users. Fortunately, we have the option to treat tiling diagrams only as a mental model and a data structure, and render a string diagram to the screen. This can be done relatively easily by recursion on the kd-tree, performing a coordinate transformation at each recursive step. This means, essentially, that the renderer is performing a Poincaré dualisation.

At the base case of the recursion, rendering a region is done by drawing a node in its centre and then connecting it to its ports with strings. Regions labelled by non-generating morphisms, such as identities, will have special rendering code.

14 Interpretation in non-strict monoidal categories

References

[CK17]   Bob Coecke and Aleks Kissinger. *Picturing quantum processes*. Cambridge University Press, 2017.

[CSC10]   Bob Coecke, Mehrnoosh Sadrzadeh, and Stephen Clark. Mathematical foundations for a compositional distributional model of meaning. *Linguistic Analysis*, 36(1):345–384, 2010.

[Daw95]   Robert Dawson. A forbidden-suborder characterisation of binarily-composable diagrams in double categories. *Theory and applications of categories*, 1(7):146–153, 1995.

[dBvKOS97] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Orthogonal range searching. In *Computational geometry: Algorithms and applications*. Springer, 1997.

[Del] Antonin Delpeuch. A complete language for faceted dataflow programs. Forthcoming in proceedings of Applied Category Theory 2019.

[Fon15] Brendan Fong. Decorated cospans. *Theory and applications of categories*, 30(33):1096–1120, 2015.

[Fon18] Brendan Fong. Decorated corelations. *Theory and applications of categories*, 33(22):608–643, 2018.

[GHWZ18] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of Logic in Computer Science (LiCS) 2018*, pages 472–481. ACM, 2018.

[Hat02] Allen Hatcher. *Algebraic topology*. Cambridge University Press, 2002.

[Hou87] Piet Houthuys. Box sort, a multidimensional binary sorting method for rectangular boxes, used for quick range searching. *The visual computer*, 3(4):236–249, 1987.

[HR17] Jacob Holm and Eva Rotenberg. Dynamic planar embeddings of dynamic graphs. *Theory of computing systems*, 61:1054–1083, 2017.

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

[Kar18] Adam Karczmarz. *Data structures and dynamic algorithms for planar graphs*. PhD thesis, University of Warsaw, 2018.

[KvdW] Aleks Kissinger and John van de Wetering. PyZX: Large-scale automated diagrammatic reasoning. Forthcoming in proceedings of Quantum Physics and Logic 2019.

[KZ15] Aleks Kissinger and Vladimir Zamdzhiev. Quantomatic: A proof assistant for diagrammatic reasoning. In *Automated Deduction - CADE-25*, volume 9195 of *Lecture Notes in Computer Science*, pages 326–336, 2015.

[Mok17] Andrey Mokhov. Algebraic graphs with class (functional pearl). In *Proceedings of the 10th ACM SIGPLAN International Symposium on Haskell*. ACM, 2017.

[Pav13] Dusko Pavlovic. Monoidal computer I: Basic computability by string diagrams. *Information and Computation*, 226:94–116, 2013.

[Tam96] Roberto Tamassia. On-line planar graph embedding. *Journal of algorithms*, 21:201–239, 1996.

[Tea19]   Statebox Team. The mathematical specification of the statebox language. arXiv:1906.07629, 2019.