A Category of Surface-Embedded Graphs

Malin Altenmüller [1] Ross Duncan[1,2]
malin.altenmuller@strath.ac.uk ross.duncan@strath.ac.uk
[1]University of Strathclyde
[2]Cambridge Quantum Computing Ltd

We introduce a categorical formalism for rewriting surface-embedded graphs. Such graphs can represent string diagrams in a non-symmetric setting where we guarantee that the wires do not intersect each other. The main technical novelty is a new formulation of double pushout rewriting on graphs which explicitly records the boundary of the rewrite. Using this boundary structure we can augment these graphs with a rotation system, allowing the surface topology to be incorporated.

1 Introduction

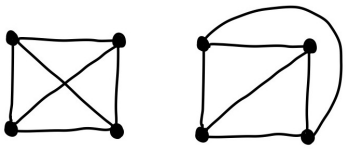
String diagrams [17] are a graphical formalism to reason about monoidal categories. Equational reasoning in *symmetric* string diagrams can be implemented as graph or hyper-graph rewriting subject to various side conditions to capture the precise flavour of the monoidal category intended [5, 6, 7, 14, 2, 1]. We want to use string diagrammatic reasoning for monoidal categories which are not necessarily symmetric. Informally, the lack of symmetry is often stated as "the wires cannot cross" – but what does that mean when the string diagram is a graph or other combinatorial object? Where is this "crossing" taking place? To make sense of this we must move beyond the situation where only the connectivity matters and add some topological information.

In this paper we make two steps in that direction. Firstly we borrow a tool from topological graph theory – rotation systems – and use it to define a category of graphs which are embedded in some surface. Secondly, we introduce a new refinement of double pushout rewriting [9] which is adapted to this category. This refinement was motivated by the desire to do rewriting on rotation systems, however it works on conventional directed graphs equally well, and removes many annoyances encountered when using standard techniques from algebraic graph rewriting in string diagrams. This is an important step towards formalising non-symmetric string diagrams and their rewriting theory.

Our motivation is also twofold. From the abstract point of view, non-symmetric string diagrams can capture a larger class of theories, including both the symmetric case and the braided monoidal one. A more practical motivation comes from the area of quantum computing, where string diagrams are often used to model quantum circuits [3], their connectivity restrictions imposed by the qubit architecture [4] require a theory without implicit SWAP gates, and can involve circuits defined on quite complex surfaces.

Curiously, Joyal and Street's original work [12] formalised monoidal categories as plane embedded diagrams, and used the plane to carry the categorical structure. Our work goes in the opposite direction: to recover the topology from the combinatorial structure.

Graph Embeddings Graph embeddings are drawings of graphs on surfaces. A graph can have multiple different embeddings on the same surface, as shown below.



We will only consider *plane* embeddings, like the one on the right, where the graph does not intersect itself. A graph need not be embedded in the plane; different surfaces can permit different embeddings. Indeed it may not be possible to embed a graph in the plane at all!

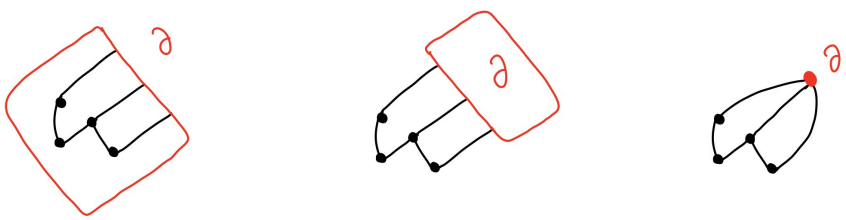
Dealing with lines and points as submanifolds of some surface (up to homeomorphism) is quite unwieldy, so we use a combinatorial representation of graph embeddings called *rotation systems*. A rotation system imposes an order on the edges incident at vertex (called a *rotation*). The rotation information at each vertex is enough to fix the embedding of the graph into some surface, as it defines the faces of the embedding uniquely. This is a well studied topic in graph theory and we refer to the literature for more details [10].

Theorem 1.1. *A rotation system determines the embedding of a connected graph into a minimal surface up to homeomorphism [11, 8, 10].*

Note that different rotation systems for the same graph may have different minimal surfaces, which need not be the plane.

Boundary Graphs and Partitioning Spans When using string diagrams, graphs as usually defined are not the most natural object; rather, we often think about *open* graphs which have "half-edges" or "dangling wires" which represent the domain and codomain of the morphism in question. The half-edges therefore provide the interface along which morphisms compose, and also where substitutions can be made in rewriting. Unfortunately half-edges don't work particularly well with double pushout rewriting, necessitating various workarounds encoding the "wires" as special vertices in a graph [7] or hypergraph [1]. This in turn leads to its own complications when we consider the identity morphism, and other natural transformations which are naturally "all string"; equations which should be trivial are no longer so. Surface embedded graphs suggest a different approach to this question.

Naively, when picturing a rewrite on a surface embedded graph, we picture a disc-like region of the surface which is removed and replaced. The edges which cross the boundary of this region define the interface and we naturally require that the removed disc and its replacement should have the same interface. From the outside, this disc is homeomorphic to a point, so it can be treated as if it was a vertex equipped with a rotation system. However, the perspective from inside and outside the region are completely equivalent, so we can dually view the rest of the graph as a single vertex connected to the interior of the disc. We think of and draw a graph with boundary in three different ways:



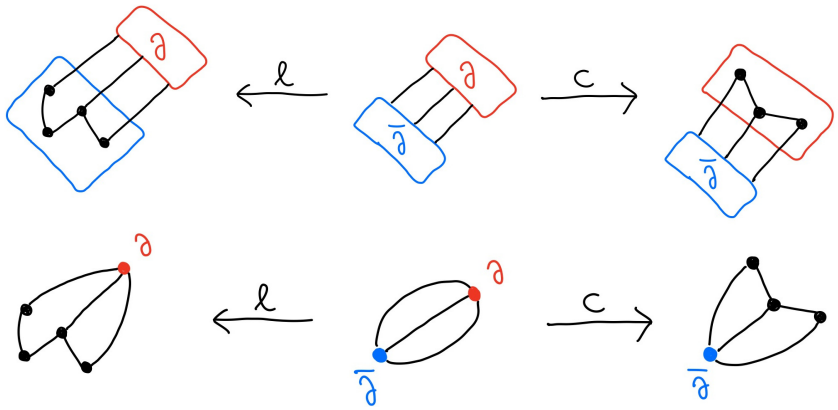


Figure 1: Example of a partitioning span, drawn in a region- and a vertex-style (edge directions are omitted for readability)

On the left, the graph is depicted as a region of the surface with its outside being the rest of the surface. In the middle, graph and its surrounding are both regions of the surface, and on the right we have drawn the boundary as a vertex with the interconnecting edges attached.

This leads naturally to our notion of *boundary graph*: we contract both subgraphs on either side of the boundary to points, leaving a two-vertex graph whose edges specify the connections crossing the boundary. Boundary graphs form the vertex of *partitioning spans*, which specify the whole graph as the two parts, as shown in Figure 1; the pushout of a partitioning span is the original graph.

This formalism allow us to use a simple definition of graph, although our morphisms are now built from partial functions, which introduces some complications around the required injectivity properties to preserve the type of the vertices, which is essential if these graphs are to be interpreted as string diagrams.

Limitations The astute reader will have noted that Theorem 1.1 applies only to connected graphs. To specify an embedding of a disconnected graph a rotation system does not suffice. We would also need to take into account the relationship between components and faces of the graph. We have made no attempt to do so here.

2 A Suitable Category of Graphs

In this section we will introduce a category of directed graphs without reference to any topological structure. The main difficulty here is arriving at the correct notion of graph morphism: our intent here is that the graphs represent terms in some monoidal category – i.e. string diagrams – and the morphisms represent *embeddings* of subterms. This implies that certain structures should be preserved which conventional graph rewriting does not worry about. Our choices here are also influenced by the variant of double-pushout rewriting we will define in the next section. In later sections we will show how to incorporate the plane topology by adding rotation systems.

A total graph is a functor $G : (\bullet \rightrightarrows \bullet) \to \mathbf{Set}$. Concretely, such a graph is a pair of sets V and E, of *vertices* and *edges* respectively, and a pair of functions s and t assigning *source* and

target vertices to each edge.

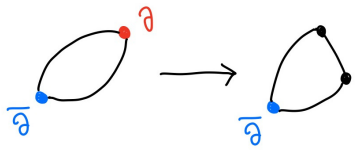
$$E \underset{t}{\overset{s}{\rightrightarrows}} V.$$

In the functor category $[\bullet \rightrightarrows \bullet, \mathbf{Set}]$, a morphism of graphs is a pair of functions $f_V : V \to V', f_E : E \to E'$, such that the following squares commute:

$$\begin{array}{ccc} E & \xrightarrow{f_E} & E' \\ {\scriptstyle s}\downarrow & & \downarrow{\scriptstyle s'} \\ V & \xrightarrow[f_V]{} & V' \end{array} \qquad \begin{array}{ccc} E & \xrightarrow{f_E} & E' \\ {\scriptstyle t}\downarrow & & \downarrow{\scriptstyle t'} \\ V & \xrightarrow[f_V]{} & V' \end{array} \tag{1}$$

Sadly for us, this simple and elegant definition will not suffice.

We want to consider graph morphisms which can replace vertices with subgraphs, and therefore forget these vertices, as shown below:



To achieve this we could operate in a subcategory of $[\bullet \rightrightarrows \bullet, \mathbf{Pfn}]$, the category of partial graphs and maps, with only the total graphs as objects. However this is not quite enough. Commutation of the naturality squares (1) in this category is strict, meaning it includes equality of the domains of definition. Therefore if a morphism forgets a vertex it must also forget all the incident edges at that vertex. This is no use. We address this issue by using the poset enrichment of **Pfn**, and work in the category $[\bullet \rightrightarrows \bullet, \mathbf{Pfn}]_{\leq}$ of functors and *lax* natural transformations:

$$\begin{array}{ccc} E & \xrightarrow{f_E} & E' \\ {\scriptstyle s}\downarrow & \leq & \downarrow{\scriptstyle s'} \\ V & \xrightarrow[f_V]{} & V' \end{array} \qquad \begin{array}{ccc} E & \xrightarrow{f_E} & E' \\ {\scriptstyle t}\downarrow & \leq & \downarrow{\scriptstyle t'} \\ V & \xrightarrow[f_V]{} & V' \end{array} \tag{2}$$

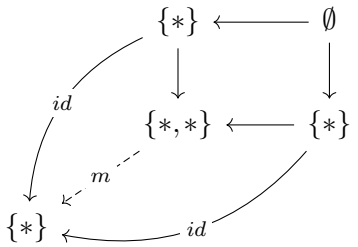
The lax commutation allows the vertex component of a morphism to be undefined at some vertex v while its incident edges may be preserved. However, if an edge is "forgotten" then its source and target vertices must also be so. We'll need more, but let's take $[\bullet \rightrightarrows \bullet, \mathbf{Pfn}]_{\leq}$ as our ambient category for now.

Proposition 2.1. *The category* **Pfn** *of sets and partial functions has pushouts.*

Proof. Given a span $L \xleftarrow{l} B \xrightarrow{c} C$, the elements of the pushout are the same as in for **Set**, but restricted to a subset $B' \subseteq B$, with both $l(b)$ and $c(b)$ defined for $b \in B'$. This is the only way the square commutes for elements in B', and the universal property of the pushout can be derived from **Set**. □

Proposition 2.2. *The category* **Inj** *of sets and injective functions does not have pushouts.*

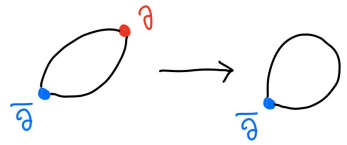
Proof. If pushouts in **Inj** exist, they have to coincide with those of **Set**. Consider the span $\{*\} \leftarrow \emptyset \rightarrow \{*\}$, and commuting squares:



In the square all morphisms are injective, but the mediating map out of the pushout $m : \{*,*\} \to \{*\}$ is not. □

We would like to be able to accommodate two further properties in our notion of graph morphism: Firstly, since vertices represent morphisms of a monoidal category, their type should be preserved. Secondly, we want to specify when a morphism is a graph *embedding*, which requires an injectivity property. Merely asking for injectivity of the vertex and edge component is not enough though, our setup requires the edge component to be non-injective, i.e. to represent the identity morphism (or similar circumstances):

Example 2.3. A graph morphism with a non-injective edge component:



Both of the above requirements turn out to be properties of the connection points between vertices and their incident edges, called *flags*:

Definition 2.4. Given a graph (V, E, s, t) its set of *flags* is defined

$$F = \{(e, s(e)) : e \in E\} + \{(e, t(e)) : e \in E\}$$

Given a graph morphism $f : G \to G'$ there is an induced *flag map*, $f_F : F \to F'$,

$$f_F = (f_E \times f_V) + (f_E \times f_V)$$

Note that the flag map is in general a partial map: it is undefined on (e, v), whenever f_V is undefined on v. Whenever f_F is injective we say that f is *flag injective*.

Flag injectivity allows edges to be combined but prevents a morphism from decreasing a vertex degree in the process. However, nothing said so far forbids a morphism from increasing the degree of a vertex: we require a notion of *flag surjectivity*. Given $f : G \to G'$, it doesn't suffice to require the flag map f_F to be surjective, since in general G' will contain more vertices than G, and hence more flags. The resulting definition is unfortunately unintuitive.

Definition 2.5. Let $f : G \to G'$ be a morphism between two total graphs; we say that f is *flag surjective* if the two diagrams below commute laxly,

$$\begin{array}{ccc} V & \xrightarrow{f_V} & V' \\ {\scriptstyle s^{-1}}\downarrow & \geq & \downarrow{\scriptstyle s'^{-1}} \\ P(E) & \xrightarrow[P(f_E)]{} & P(E') \end{array} \qquad \begin{array}{ccc} V & \xrightarrow{f_V} & V' \\ {\scriptstyle t^{-1}}\downarrow & \geq & \downarrow{\scriptstyle t'^{-1}} \\ P(E) & \xrightarrow[P(f_E)]{} & P(E') \end{array} \tag{3}$$

where s^{-1} and t^{-1} are the preimage maps of s and t respectively, and P is the powerset functor.

If a flag surjective morphism f is defined on a vertex v, it will ensure that all edges attached to $v' = f_V(v)$ are in the image of f_E, thus no additional edges can be attached to v' in the process. An example of a morphism which is not flag surjective can be found in Figure 10 in Appendix B. We'll call a morphism which is both flag injective and flag surjective a *flag bijection*. This is quite a strong property; it's almost enough to make the vertex map injective, but not quite.

Lemma 2.6. *Let $f : G \to G'$ be a flag bijection, and suppose that $f_V(v_1) = f_V(v_2)$ and both are defined; then $\deg v_1 = \deg v_2 = 0$.*

Proof. Let $v' = f_V(v_1) = f_V(v_2)$; since f is flag injective, the set of flags at v' must contain (the image of) the disjoint union of the flags at v_1 and v_2; hence $\deg v' \geq \deg v_1 + \deg v_2$. Since (by (2)) f_E is defined on all the flags at v_1, flag surjectivity implies that $\deg v_1 \geq \deg v'$, and similarly for v_2. Hence $\deg v' = \deg v_1 = \deg v_2 = 0$. □

Lemma 2.7. *Let G and G' be total graphs, and let $f : G \to G'$ be a flag bijection. For all $v \in V$, if $f_V(v)$ is defined, then f_E defines a bijection between the flags at v and those incident at $f_V(v)$; in consequence $\deg v = \deg f_V(v)$.*

Proof. Let $v' = f_V(v)$. The edges incident at v are given by the disjoint union of $s^{-1}(v)$ and $t^{-1}(v)$, and likewise at v'. Since f is flag injective, f_E is injective on the subset of flags defined by v. Since f is flag surjective all the flags at v' are in the image of $f_E(s^{-1}(v)) + f_E(t^{-1}(v))$. Note that since $f_V(v)$ is defined then f_E is defined for all $e \in s^{-1}(v)$ and all $e \in t^{-1}(v)$ by Eq. (2). Hence we have a bijection as required. □

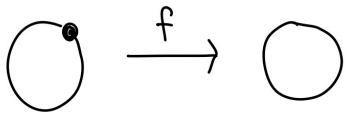
Lemma 2.8. *Let $f : G \to H$ and $g : H \to J$ be flag bijections; then $g \circ f$ is a flag bijection.*

Proof. See Appendix A. □

By the preceding lemma, and by observing that the identity is a flag bijection, we may conclude that the flag bijections define a wide subcategory of $[\bullet \rightrightarrows \bullet, \mathbf{Pfn}]_{\leq}$, which we will call $\mathcal{B}$.

Example 2.3 suggests a confounding special case: the vertex of a self loop can be forgotten. Here is another one:

Example 2.9. Let G be the (unique) total graph with one vertex and one edge; let G' be the (unique) partial graph with no vertices and a single edge. Define $f : G \to G'$ by $f_V = \emptyset$ and $f_E = \mathrm{id}_{\mathbf{1}}$. This is a valid flag bijection in $\mathcal{B}$.



While it is tempting to restrict to the subcategory defined by the total graphs, and ban such monsters by fiat, they do occur quite naturally in the rewrite theory we propose, albeit in quite restricted circumstances. So they must be tamed. To do so, we extend the definition of graph with *circles*: closed edges which have neither a source nor a target vertex[1]. Unfortunately the definition of graph morphism will get more complex and the resulting category is no longer a functor category, as we shall now see.

[1]This notion of graph has a long history; see, for example, the work of Kelly and Laplaza on compact closed categories [13].

Definition 2.10. A *graph with circles* is a 5-tuple $G = (V, E, O, s, t)$ where (V, E, s, t) is a total graph and O is a set of *circles*. For notational convenience we define the set of *arcs* as the disjoint union $A = E + O$.

A morphism $f : G \to G'$ between two graphs with circles consists of two (partial) functions $f_V : V \rightharpoonup V'$ as above, and $f_A : A \to A'$, satisfying the conditions listed below. Note that any such f_A factors as four maps,

$$\begin{array}{cc} f_E : E \to E' & f_{EO} : E \to O' \\ f_{OE} : O \to E' & f_O : O \to O' \end{array}$$

The following conditions must be satisfied:

1. $f_A : A \to A'$ is total;
2. the component $f_{OE} : O \to E'$ is the empty function;
3. the pair (f_V, f_E) forms a flag surjection between the underlying graphs in $\mathcal{B}$.

If, additionally, the following three conditions are satisfied, we call the morphism an *embedding*:

4. $f_V : V \rightharpoonup V'$ is injective;
5. the component f_O is injective;
6. the pair (f_V, f_E) forms a flag bijection between the underlying graphs.

It's worth noticing that if some f_A maps an edge e to a circle, then $f_E(e)$ is undefined, but $f_{EO}(e)$ is defined. This, by the lax naturality property, implies that f_V is undefined on both $s(e)$ and $t(e)$. Various examples and non-examples of morphisms and embeddings of graphs with circles can be found in Appendix B.

Lemma 2.11. *Defining composition point-wise, the composite of two morphisms of graphs with circles is again such a morphism. Additionally, if both morphisms are embeddings, their composition is an embedding as well.*

Proof. See Appendix A. □

We finally have introduced all the necessary structure to define our suitable category of graphs.

Definition 2.12. Let $\mathcal{G}$ be the category whose objects are graphs with circles, and whose arrows are morphisms as per Definition 2.10.

There is an obvious and close relationship between $\mathcal{G}$ and the category of partial graphs and flag bijections, $\mathcal{B}$. We can make this precise.

Definition 2.13. We define a forgetful functor $U : \mathcal{G} \to \mathcal{B}$ by

$$\begin{array}{ccc} U : (V, E, O, s, t) & \longmapsto & (V, E, s, t) \\ {\scriptstyle U:(f_V, f_A)}\downarrow & \mapsto & \downarrow{\scriptstyle (f_V, f_E)} \\ U : (V', E', O', s', t') & \longmapsto & (V', E', s', t') \end{array}$$

Example 2.14. Returning to Example 2.9, we see how this degenerate case fits in to the framework. We start with G, the unique total graph with a single vertex and a single edge (and no circles). There a single valid way to erase the vertex in $\mathcal{G}$.

Firstly observe that $G' = (\emptyset, \{e\}, \emptyset, \emptyset, \emptyset)$ as in the earlier example is not an object in $\mathcal{G}$. However $G'' = (\emptyset, \emptyset, \{e\}, \emptyset, \emptyset)$ is a valid graph, and the map $f : G \to G''$ which is undefined on the vertex and sends the edge to the circle is a valid morphism, indeed the only one.

Finally observe that the image of UG'' is the empty graph and Uf is the empty function.

The term "graph with circles" is unacceptably cumbersome, so henceforth we will simply say "graph" and refer to $\mathcal{G}$ as the category of graphs. In practice the circles are rarely important, although we will devote a disappointingly large amount of this paper to them.

3 DPO Rewriting in the Suitable Category

Double pushout rewriting [9] is an approach to formalising equational theories over graphs by rewriting. Each equation is formalised as a rewrite rule $L \Rightarrow R$, and the substitution $G[R/L]$ is computed via a double pushout as shown below.

$$\begin{array}{ccccc} L & \xleftarrow{l} & B & \xrightarrow{r} & R \\ {\scriptstyle m}\downarrow & \urcorner & \downarrow & \ulcorner & \downarrow \\ G & \xleftarrow{} & C & \xrightarrow{} & H \end{array}$$

The upper span embeds a *boundary graph* B into both L and R; ensures that both graphs have the same connectivity, and hence that R can validly replace L. The map $m : L \to G$ is the *match*, an embedding of L into G, which shows where the rewrite will occur. The first pushout square is completed by C, the *context graph*; it is basically G with L removed. In the DPO approach, C is computed as a *pushout complement*. Finally the graph $H = G[R/L]$ is the graph resulting from performing the rewrite $L \Rightarrow R$ in G; it is computed as a pushout.

In the algebraic graph literature the notion of adhesive category [15, 16] is commonly used, as DPO rewriting behaves well in such categories. However, adhesivity is not suitable for our purposes, since the monomorphisms of $\mathcal{G}$ don't play any special role in our formalism. We will instead consider a specific case of maps in the DPO diagram only, and in that context show the existence of pushouts and the existence and uniqueness of pushout complements, which are similar properties to those of an adhesive categories. The key to this approach is to recognise that B and C are in some sense partial graphs, as to a lesser extent are L and R; our handling of this partiality is one of the main novelties of this paper.

Notation 3.1. Almost every map in this section is an embedding of a small object into a larger one. Wherever unambiguous to do so, we will treat these embeddings as actual inclusions so, for example, we may write $m_E(e) = e$ despite the fact that the domain and codomain of the map are different graphs.

In our approach the graphs L and R that make up a rewrite rule have an additional distinguished vertex, the *boundary vertex* ∂, which represents the rest of the world, from the perspective of L (or R). The incident edges at ∂ represent the interface between L and the rest of the graph it occurs in. The context graph C also has a distinguished vertex, the dual boundary $\bar{\partial}$ which represents its interface. In our formalism, the graph B in the middle exists only to say that these interfaces must be compatible.

Definition 3.2. A *boundary graph* is a graph with exactly two vertices, ∂ and $\bar{\partial}$ (called respectively the *boundary* and *dual boundary* vertices), where $s(e) \neq t(e)$ for all its edges e, and there are no circles.

Definition 3.3. A *partitioning span* is a span $L \xleftarrow{l} B \xrightarrow{c} C$ in $\mathcal{G}$, where B is a boundary graph, the vertex component l_V is defined on ∂ and undefined on $\bar{\partial}$ and, dually, c_V is undefined on ∂ and defined on $\bar{\partial}$. Further, we require l and c to be embeddings.

An example of a partitioning span and its pushout in $\mathcal{G}$ is depicted in Figure 2. The name *partitioning span* arises from the fact that each of the maps out of the boundary graph replaces one half of it. Hence each graph has two regions, connected via the edges present in the boundary graph.

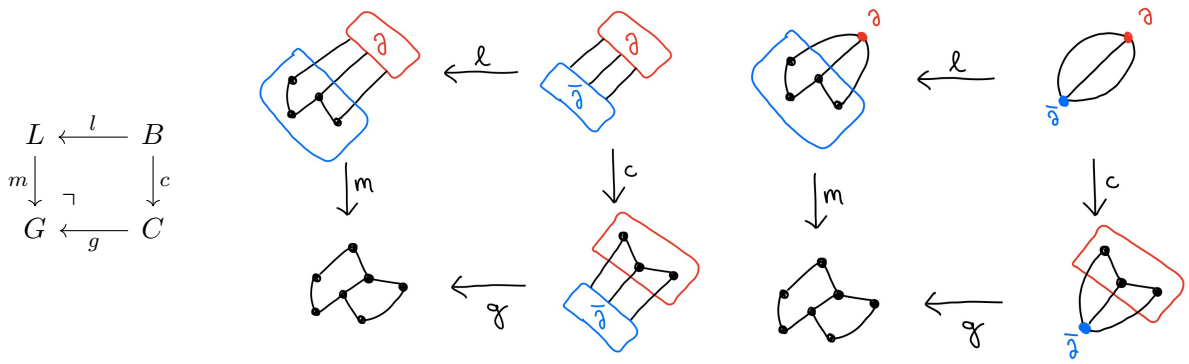


Figure 2: Pushout of the partitioning span from Figure 1, drawn two different ways

Lemma 3.4. *Let $L \xleftarrow{l} B \xrightarrow{c} C$ be a partitioning span and suppose that $e = l_E(e_1) = l_E(e_2)$ in L for distinct e_1 and e_2 in E_B. Then e is a self-loop at ∂ in L and for all other $e_3 \neq e_1 \neq e_2$ we have $e \neq l_E(e_3)$. The same holds mutatis mutandis for C.*

Proof. By flag bijectivity all the flags at ∂ must be preserved, including distinct flags for $l_E(e_1)$ and $l_E(e_2)$. By hypothesis these two edges are identified so necessarily $s_B(e_1) = \partial$ and $s_B(e_2) = \bar{\partial}$ or vice versa. Hence e is a self loop. Suppose further that $l_E(e_3) = e$; then l is not flag bijective, which is a contradiction. □

Theorem 3.5. *In $\mathcal{G}$, pushouts of partitioning spans exist. Further, the maps into the pushout are embeddings.*

Proof. See Appendix A. □

Since pushouts of partitioning spans are the basis of the rewrite theory we wish to pursue, for the rest of the paper the term "pushout" should be understood to imply "of partitioning span".

Implicit in the proof of Theorem 3.5 there is a structure which will be important later, when understanding when the edges in the boundary graph will be identified in a pushout.

Lemma 3.6. *Let G be the pushout of $L \xleftarrow{l} B \xrightarrow{c} C$ then the preimage of any arc in A_G defines a (possibly empty)* path *in B.*

Proof. Consider an arc $e \in A_G$ and $e_1 \in (m_E \circ l_E)^{-1}(e)$ (if there is no such e_1 we're done – the path is empty.) Wlog say that $s_B(e_1) = \bar{\partial}$ and $t_B(e_1) = \partial$. Look at the image of e_1 in L; if $(s_L \circ l_E)(e_1) \neq \partial$ then this is the first edge of the path; otherwise $l_E(e_1)$ is a self loop in L, so (cf Lemma 3.4) there exists $e_2 \in E_B$ such that $l_E(e_1) = l_E(e_2)$ and $s_B(e_2) = \partial$. Add e_2 to beginning

of the path. Now look at the image of e_2 in C and perform the same procedure. We continue in this way, alternating sides of the span until we either reach the start of the path (in which case $e \in E_G$) or we reach e_1 again (in which case $e \in O_G$). Having found the start of the path we repeat the trick with the target map to find the end. □

Another way to view this is a *pairing graph* $\mathcal{P}$ where the vertices are the edges of B, and the edges are the self loops of L and C.

Definition 3.7. The *pairing graph* for a partitioning span $L \xleftarrow{l} B \xrightarrow{c} C$ is a directed graph whose vertices are E_B; each vertex receives a *polarity*: $+$ if $s_B(e) = \partial$, $-$ if $s_B(e) = \bar{\partial}$. We draw a *blue* edge between e_1 and e_2 if $l_E(e_1) = l_E(e_2)$ i.e. if e_1 and e_2 form self-loop in L; similarly we draw a *red* edge between e_1 and e_2 if they form a self-loop in C. Blue edges are directed from positive to negative polarity; red edges the reverse. (Note that if two edges form a self-loop they must have opposite polarity.)

An example of a pairing graph is shown in Figure 3.

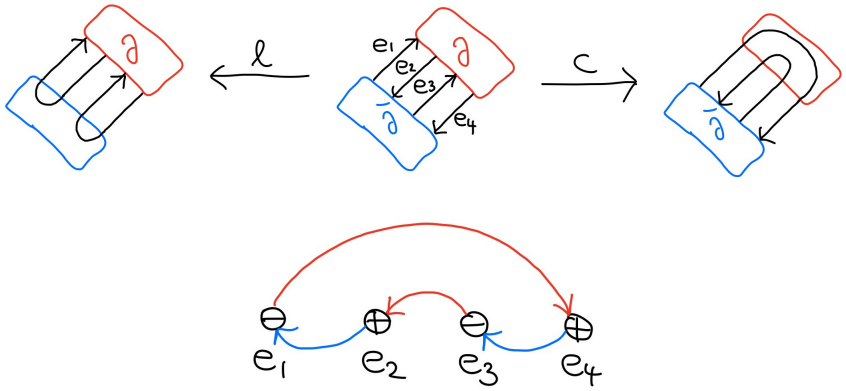


Figure 3: An examples of a partitioning span with its pairing graph

Corollary 3.8. *Let $\mathcal{P}$ be the pairing graph of $L \xleftarrow{l} B \xrightarrow{c} C$, and let G be its pushout.*

- *Each connected component of $\mathcal{P}$ determines an edge-disjoint path on B.*
- *If e and e' are in the same connected component of G then $(m_E \circ l_E)(e) = (m_E \circ l_E)(e')$.*

We now move on to the other required ingredient for DPO rewriting: pushout complements. Just as we did with partitioning spans and pushouts, we will introduce a specific kind of embedding for which the complement must exist.

Definition 3.9. A *boundary embedding* is a pair of maps $B \xrightarrow{l} L \xrightarrow{m} G$ in $\mathcal{G}$, where B is a boundary graph, where : (i) $l_V(\partial)$ is defined but $l_V(\bar{\partial})$ is undefined; and (ii) $(m_V \circ l_V)(\partial)$ is undefined. Further, L has to be a connected graph, and m an embedding.

Definition 3.10. Given a boundary embedding $B \xrightarrow{l} L \xrightarrow{m} G$ we can immediately construct half a pairing graph $\mathcal{P}$, consisting of only the blue edges using the mapping $l : B \to L$. The *re-pairing problem* is to construct the other half (the red edges) so that the connected components map to the edges of G (cf. Corollary 3.8). See Figure 5 for examples.

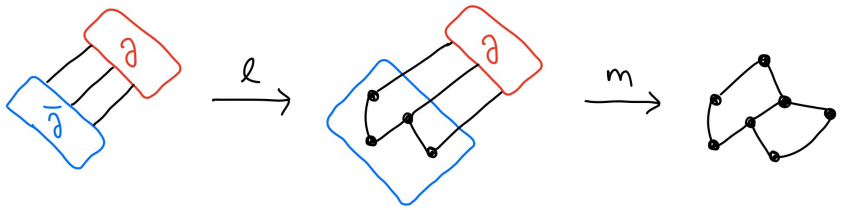


Figure 4: Example of a boundary embedding

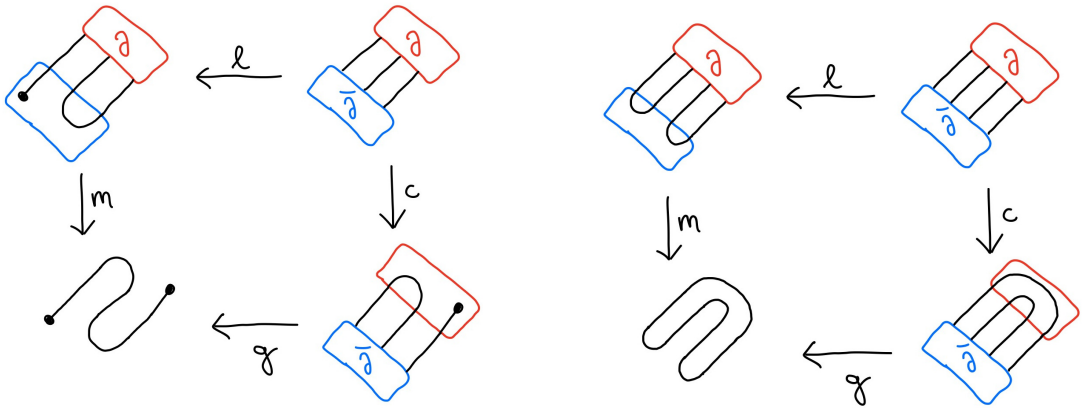
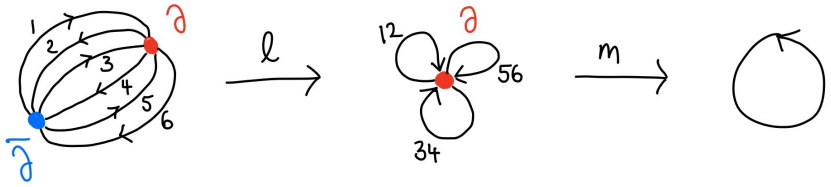


Figure 5: Two different examples of the re-pairing problem, connecting one component in the pushout.

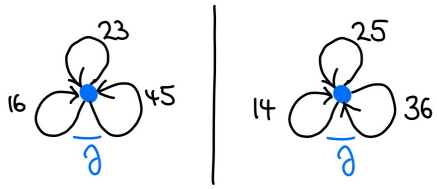
Lemma 3.11. *Given a boundary embedding* $B \xrightarrow{l} L \xrightarrow{m} G$ *a solution to the re-pairing problem always exists, but it is not necessarily unique.*

Proof. See Appendix A. □

Example 3.12. Consider the following boundary embedding:



This embedding has two different solutions to the re-pairing problem:



Theorem 3.13. *In* $\mathcal{G}$*, pushout complements of boundary embeddings exist, and give rise to partitioning spans.*

Proof. We'll use the boundary embedding $B \xrightarrow{l} L \xrightarrow{m} G$ to construct the complement C such that $L \xleftarrow{l} B \xrightarrow{c} C$ is a partitioning span, and show that G is indeed the pushout of this span.

Let C have vertex set $V_C = (V_G \setminus V_L) + \{\bar{\partial}\}$. We'll construct the edge set, and the source and target maps, in three steps.

1. Let E_C contain all the edges of the induced subgraph of G defined by the vertices V_C, and define the source and target maps on those edges correspondingly.
2. Let O_C contain $O_G \setminus m_O^{-1}(O_G)$.
3. Finally we add the edges between $\bar{\partial}$ and the rest of the graph, and simultaneously define the map $c : B \to C$. Let $\mathcal{P}$ be a solution to the re-pairing problem given by $B \xrightarrow{l} L \xrightarrow{m} G$. If in $\mathcal{P}$ there is a red edge between e_1 and e_2 in create a self-loop e at $\bar{\partial}$ and set $c(e_1) = c(e_2) = e$. If there is any vertex e in $\mathcal{P}$ which has no incident red edge, add e to E_C; if its polarity is positive set
$$s_C(e) = (s_G \circ m_E \circ l_E)(e) \qquad t_C(e) = \bar{\partial}$$
and if the polarity is negative, the source and target are reversed. We define $c_E(e) = e$.

The resulting span $L \xleftarrow{l} B \xrightarrow{c} C$ is evidently partitioning, and by construction has G as its pushout, as a consequence of Corollary 3.8 □

Theorem 3.14. *In $\mathcal{G}$, pushout complements are unique up to the solution of the re-pairing problem.*

Proof. Suppose that both $B \xrightarrow{c} C \xrightarrow{g} G$ and $B \xrightarrow{c'} C' \xrightarrow{g'} G$ are pushout complements for the boundary embedding $B \xrightarrow{l} L \xrightarrow{m} G$. Observe that given the boundary embedding, a solution to the re-pairing problem determines the map $c : B \to C$ and vice versa. Let's assume for now a that $\mathsf{im}(c) = \mathsf{im}(c')$ and hence they both correspond to the same pairing graph.

Since m is an embedding, it follows that every part of C not in $\mathsf{im}(c)$ is preserved isomorphically in G, and similarly for C'. Since we have assumed $\mathsf{im}(c) = \mathsf{im}(c')$ this implies that $C \simeq C'$.

Further, observe that different solution of the re-pairing also have the same number of edges, and hence produce the same number of self loops at $\bar{\partial}$. Hence the difference between different solutions is just the labels on the edges incident at $\bar{\partial}$. □

4 A Category of Rotation Systems

Despite some suggestive illustrations, up to this point we have operated in a purely combinatorial setting, but now we introduce some topological information in the form of rotation systems. A rotation system for a graph determines an embedding of the graph into a surface by fixing a cyclic order of the incident edges, or more precisely the flags, at every vertex.

We augment our category of graphs with this extra structure, in the form of *cyclic lists of flags* for each vertex, and strengthen the property of flag surjectivity (Equation 3). The requisite categorical properties for DPO rewriting will follow more or less immediately from those of the underlying category of directed graphs.

Definition 4.1. Let $\mathsf{CList} : \mathbf{Set} \to \mathbf{Set}$ be the functor where $\mathsf{CList}\, X$ is the set of circular lists whose elements are drawn from X.

Definition 4.2. A *rotation system* R for a graph with circles (V, E, O, s, t) is a total function $\mathsf{inc} : V \to \mathsf{CList}\, E$ such that :

- $e \in \mathsf{inc}(s(e))$
- $e \in \mathsf{inc}(t(e))$
- $t^{-1}(v) + s^{-1}(v) \cong \mathsf{inc}(v)$ (when considering $\mathsf{inc}(v)$ as a set)

We call $\mathsf{inc}(v)$ the *rotation* at v.

Note that $\mathsf{inc}(v)$ is actually a cyclic ordering on the set of flags at v.

Definition 4.3. A homomorphism of rotation systems $f : R \to R'$ is a $\mathcal{G}$-morphism (f_A, f_V) between the underlying graphs, satisfying the following additional condition.

$$\begin{array}{ccc} V & \xrightarrow{\quad f_V \quad} & V' \\ {\scriptstyle \mathsf{inc}}\downarrow & \geq & \downarrow{\scriptstyle \mathsf{inc}'} \\ \mathsf{CList}\, E & \xrightarrow[\mathsf{CList}\, f_E]{} & \mathsf{CList}\, E' \end{array} \tag{4}$$

This condition requires the preservation of the edges ordering on vertices where f_V is defined; it implies flag surjectivity (Equation 3). Morphisms therefore either preserve a vertex with its rotation exactly, or forget about it.

Definition 4.4. Let $\mathcal{R}$ be the category whose objects are tuples $(V, E, O, s, t, \mathsf{inc})$ where (V, E, O, s, t) is an object of the category of graphs, $\mathcal{G}$ (see Def. 2.10) and inc is a rotation system for this graph. The morphisms of $\mathcal{R}$ are homomorphisms of rotation systems.

There is an evident forgetful functor $U' : \mathcal{R} \to \mathcal{G}$; this is especially clean since the morphisms of $\mathcal{R}$ are just $\mathcal{G}$-morphisms which satisfy an additional condition. Further, since we demand require the inc structure to be preserved exactly, pushouts and complements are very easily defined here.

Definition 4.5. In $\mathcal{R}$, objects B, spans $L \xleftarrow{l} B \xrightarrow{c} C$ and composites $B \xrightarrow{l} L \xrightarrow{m} G$ are respectively *boundary graphs*, *partitioning spans*, and *boundary embeddings* if their underlying graphs in $\mathcal{G}$ satisfy those definitions (respectively Definitions 3.2, 3.3, and 3.9).

Lemma 4.6. *In $\mathcal{R}$ pushouts of partitioning spans exist.*

Proof. The pushout candidate is the one in the underlying category (see Theorem 3.5), together with the rotation system:

$$\mathsf{inc}_G(v) = \begin{cases} \mathsf{inc}_C(v), & \text{if } v \in V_C \\ \mathsf{inc}_L(v), & \text{if } v \in V_L \end{cases}$$

The vertex set of the pushout is the disjoint union of vertices from both input graphs, $V_G = (V_L + V_C) \setminus V_B$. Therefore, by the mediating map from Theorem 3.5, inc_G is indeed the pushout of the rotations. □

Lemma 4.7. *In $\mathcal{R}$ pushout complements of boundary embeddings exist, and are unique up to the solution of the re-pairing problem.*

Proof. This follows from the underlying construction in $\mathcal{G}$; see Theorem 3.14. Note that the rotation for every vertex of C is determined by either those of G or of B, so there is no choice about the additional structure. □

Remark 4.8. We must sound a cautionary note about the "up to" in the preceding statement. While in $\mathcal{G}$ pushout complements that arise from different pairing graphs are essentially the same, this is not so in $\mathcal{R}$. Since the rotation around $\bar{\partial}$ is preserved exactly by $c : B \to C$, different choices for which edges to merge as self loops will result in different local topology at $\bar{\partial}$. In particular it can happen that a re-pairing problem can have planar and non-planar solutions; see Figure 6 for an example.

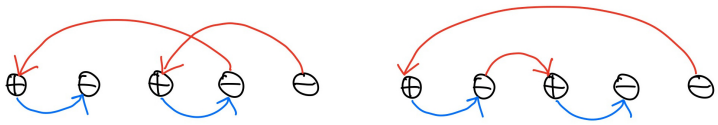


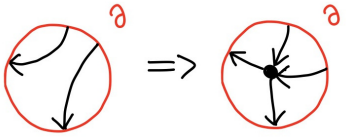
Figure 6: Topologically different solutions (in red) of the same re-pairing problem (in blue).

With that caveat noted, since $\mathcal{R}$ has pushouts and their complements, specialised to the setting where the rewrite rules explicitly encode the connectivity at their boundary, we can use it as a setting for DPO rewriting of surface embedded graphs.

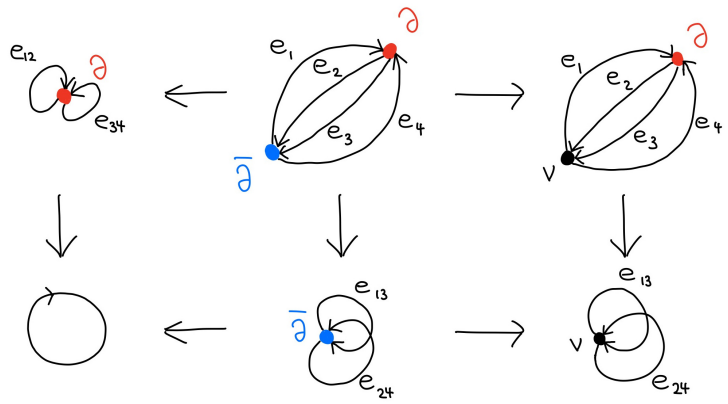
Remark 4.9. As illustrated in Figure 6, we require the pairing graph to conform to a certain shape when reasoning topologically: vertices have to be ordered according to the edges in the boundary graph, with red edges being drawn to one side of the vertices and blue edges to the other. Though a restriction, this regulation admits encoding of surface information in the pairing graph.

5 On Planarity

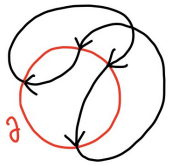
The way we construct the category of graphs makes adding rotation information to vertices convenient. Because rotation systems determine graph embeddings, this adds topological information to a graph, and considers its embeddings only. When working with embedded graphs on a specific type of surface, one may be interested in a surface invariant: applying a rewrite rule to a graph embedded on a specific surface should result in a graph embeddable into the same surface again. Encoding embedding information in our setting is possible, but preservation of topological properties may require some additional restrictions. As an example, consider the case of plane graphs, i.e. graphs embedded into the surface of the sphere (or, equivalently, the plane). A rewrite theory would ideally be able to provide rewrites which are guaranteed to result in a plane graph again. Consider a rewrite rule:



This is a legitimate rewrite rule for plane graphs, and expanding it into a span for the top of a double pushout diagram makes sense.



In this example we match the left hand side of the rule to the graph with one edge and no vertices, and compute the complement and the result of substituting the right hand side into the context graph.
When computing the pushout complement of this boundary embedding, we notice that there is only one solution to the re-pairing problem, and that this solution is not plane. In a setting where all embeddings are plane this is an unwanted case. The non-plane structure of the pushout complement graph provides some very useful information though: the circle at the bottom left is seemingly plane, but with the flags in L fixed, there is no way this circle can be drawn on the plane without edges crossing.



Because rotation systems are orderings of edges around vertices, they do not store information about the embedding of circles, because they are not attached to any vertex. But by observing the structure of the solution to the re-pairing problem, we are able to distinguish plane and non-plane embeddings of circles. Such rewrites should not allowed in a planar setting, and hence will have to be prohibited. This can either be done by restricting the format of rewrite rules, or allow rejection of a rewrite if the pushout complement shows that topological properties are not preserved.

References

[1] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski & Fabio Zanasi (2016): *Rewriting modulo symmetric monoidal structure*. In: *LiCS'16 Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pp. 707–719, doi:10.1145/2933575.2935316.

[2] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski & Fabio Zanasi (2017): *Confluence of Graph Rewriting with Interfaces*. In: *ESOP'17*.

[3] Bob Coecke, Ross Duncan, Aleks Kissinger & Quanlong Wang (2015): *Generalised Compositional Theories and Diagrammatic Reasoning*. In G. Chirabella & R. Spekkens, editors: *Quantum Theory: Informational Foundations and Foils*, Springer, doi:10.1007/978-94-017-7303-4_10.

[4] Alexander Cowtan, Silas Dilkes, Ross Duncan, Alexandre Krajenbrink, Will Simmons & Seyon Sivarajah (2019): *On the qubit routing problem*. In Wim van Dam & Laura Mancinska, editors: *14th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2019)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 135, pp. 5:1–5:32, doi:10.4230/LIPIcs.TQC.2019.5. Available at `http://drops.dagstuhl.de/opus/volltexte/2019/10397`.

[5] Lucas Dixon & Ross Duncan (2009): *Graphical Reasoning in Compact Closed Categories for Quantum Computation*. *Annals of Mathematics and Artificial Intelligence* 56(1), pp. 23–42, doi:10.1007/s10472-009-9141-x.

[6] Lucas Dixon, Ross Duncan & Aleks Kissinger (2010): *Open Graphs and Computational Reasoning*. In: *Proceedings DCM 2010*, *Electronic Proceedings in Theoretical Computer Science* 26, pp. 169–180, doi:10.4204/EPTCS.26.16.

[7] Lucas Dixon & Aleks Kissinger (2013): *Open Graphs and Monoidal Theories*. *Math. Structures in Comp. Sci.* 23(2), pp. 308–359, doi:10.1017/S0960129512000138.

[8] J. R. Edmonds (1960): *A combinatorial representation for polyhedral surfaces*. *Notices of the American Mathematical Society* 7(A646).

[9] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange & Gabriele Taentzer (2006): *Fundamentals of Algebraic Graph Transformation*. Monographs in Theoretical Computer Science, Springer Berlin Heidelberg, doi:10.1007/3-540-31188-2.

[10] Jonathan L. Gross & Thomas W. Tucker (2001): *Topological Graph Theory*. Dover.

[11] Lothar Heffter (1891): *Über das Problem der Nachbargebiete*. *Mathematische Annalen* 38(4), pp. 477–508, doi:10.1007/BF01203357.

[12] A. Joyal & R. Street (1991): *The Geometry of Tensor Categories I*. *Advances in Mathematics* 88, pp. 55–113.

[13] G.M. Kelly & M.L. Laplaza (1980): *Coherence for Compact Closed Categories*. *Journal of Pure and Applied Algebra* 19, pp. 193–213, doi:10.1016/0022-4049(80)90101-2.

[14] Aleks Kissinger & Vladimir Zamdzhiev (2015): *Equational Reasoning with Context-Free Families of String Diagrams*. In Francesco Parisi-Presicce & Bernhard Westfechtel, editors: *Graph Transformation*, *Lecture Notes in Computer Science* 9151, Springer International Publishing, pp. 138–154, doi:10.1007/978-3-319-21145-9_9. Available at `http://dx.doi.org/10.1007/978-3-319-21145-9_9`.

[15] S. Lack & P. Sobocinski (2003): *Adhesive categories*. Technical Report BRICS RS-03-31, BRICS, Department of Computer Science, University of Aarhus.

[16] Stephen Lack & Pawel Sobocinski (2005): *Adhesive and quasiadhesive categories*. *Theoretical Informatics and Applications* 39(3), pp. 511 – 545, doi:10.1051/ita:2005028.

[17] Peter Selinger (2011): *A survey of graphical languages for monoidal categories*. In Bob Coecke, editor: *New structures for physics*, *Lecture Notes in Physics* 813, Springer, pp. 289–355, doi:10.1007/978-3-642-12821-9_4.

A Proofs

From Section 2

Lemma 2.8 Let $f : G \to H$ and $g : H \to J$ be flag bijections; then $g \circ f$ is a flag bijection.

Proof. For flag injectivity, we assume injectivity of the flag maps induced by f and g. If f_V is undefined, so is the flag map. Consider flags (e,v) and (e',v') where $(f_E \times f_V)$ is defined, $v = s(e)$, $v' = s(e')$, and assume $g_F(f_F(e,v)) = g_F(f_F(e',v'))$. Because f is a flag surjection and defined on the given flags, Equation 3 holds strictly on v and v'. Therefore we get: $f_E(e) = s_H(f_V(v))$ and $f_E(e') = s_H(f_V(v'))$. This lets us apply flag injectivity of g to get $f_F(e,v) = f_F(e',v')$, and flag injectivity of f to reach $(e,v) = (e',v')$. The argument applies equally to the target map.

For flag surjectivity we assume lax commutation of Equation 3 for f and g and show that the composite diagram also commutes laxly.

$$\begin{array}{ccccc} V_G & \xrightarrow{f_V} & V_H & \xrightarrow{g_V} & V_J \\ {\scriptstyle s_G^{-1}}\downarrow & \geq & {\scriptstyle s_H^{-1}}\downarrow & \geq & \downarrow{\scriptstyle s_J^{-1}} \\ P(E_G) & \xrightarrow[P(f_E)]{} & P(E_H) & \xrightarrow[P(g_E)]{} & P(E_J) \end{array} \qquad \begin{array}{ccccc} V_G & \xrightarrow{f_V} & V_H & \xrightarrow{g_V} & V_J \\ {\scriptstyle s_G^{-1}}\downarrow & & \geq & & \downarrow{\scriptstyle s_J^{-1}} \\ P(E_G) & \xrightarrow[P(f_E)]{} & P(E_H) & \xrightarrow[P(g_E)]{} & P(E_J) \end{array}$$

In the case of either f_V or g_V being undefined, the composite $(g_V \circ f_V)$ is also undefined and the diagram commutes laxly immediately. If both f_V and g_V are defined, both their diagrams commute strictly, and by diagram gluing, their composite does as well. □

Lemma 2.11 Defining composition point-wise, the composite of two morphisms of graphs with circles is again such a morphism. Additionally, if both morphisms are embeddings, their composition is an embedding as well.

Proof. Let $f : G \to G'$ and $g : G' \to G''$ be two morphisms; then $g \circ f = ((g_{V'} \circ f_V), (g_{A'} \circ f_A))$; since composition of partial functions is associative, we need only check that the four properties of Definition 2.10 are preserved.

Conditions 1 and 4 follow from the properties of partial functions, and condition 6 (which includes condition 3) follows from Lemma 2.8. Observe that

$$\begin{aligned} (g \circ f)_O &= [g_{EO}, g_O] \circ (f_{OE} + f_O) \\ &= (g_{EO} \circ f_{OE}) + (g_O \circ f_O) \\ &= (g_{EO} \circ \emptyset) + (g_O \circ f_O) \\ &= g_O \circ f_O \end{aligned}$$

hence $(g \circ f)_O$ is injective since f_O and g_O are, satisfying condition 5. Finally, by a similar argument we have

$$\begin{aligned} (g \circ f)_{EO} &= [g_{EO}, g_O] \circ (f_E + f_{EO}) \\ &= (g_{EO} \circ f_E) + (g_O \circ f_{EO}) \\ &= (\emptyset \circ f_E) + (g_O \circ \emptyset) \\ &= \emptyset \end{aligned}$$

hence the remaining condition 2 is satisfied. □

From Section 3

Theorem 3.5 In $\mathcal{G}$, pushouts of partitioning spans exist. Further, the maps into the pushout are embeddings.

Proof. We explicitly construct the pushout graph G and show that it has the required universal property in $\mathcal{G}$. We first construct the underlying sets and functions by pushout in **Pfn**,

$$\begin{array}{ccc} V_L & \xleftarrow{l_V} & \{\partial, \bar{\partial}\} \\ {\scriptstyle m_V}\downarrow & \urcorner & \downarrow{\scriptstyle c_V} \\ V_G & \xleftarrow[g_V]{} & V_C \end{array} \qquad \begin{array}{ccc} A_L & \xleftarrow{l_A} & E_B \\ {\scriptstyle m_A}\downarrow & \urcorner & \downarrow{\scriptstyle c_A} \\ A_G & \xleftarrow[g_A]{} & A_C \end{array} \tag{5}$$

so explicitly we have

$$V_G = (V_C + V_L) \setminus V_B \qquad A_G = (A_L + A_C)/\!\sim$$

where $\sim$ is the least equivalence relation such that $l_E(e) = c_E(e)$ for $e \in E_B$. Next we define the source map by

$$s_G(e') = \begin{cases} s_L(e), \text{ if } e' = m_A(e) \text{ and } s_L(e) \text{ is defined and } s_L(e) \neq \partial \\ s_C(e), \text{ if } e' = g_A(e) \text{ and } s_C(e) \text{ is defined and } s_C(e) \neq \bar{\partial} \\ \text{undefined otherwise.} \end{cases} \tag{6}$$

for all $e' \in A_G$. The target map t_G is defined similarly. (Strictly speaking we have defined s and t on all of A; they will be restricted to E when we have defined that.) Finally we divide the arcs into edges and circles by setting

$$E_G = \{e \in A_G : \text{ both } s_G(e) \text{ and } t_G(e) \text{ are defined } \} \tag{7}$$

$$O_G = A_G \setminus E_G \tag{8}$$

Now we prove that G so defined is a valid graph.

We start by showing that s is well defined. Suppose that in L we have distinct e_1, e_2 such that $m_A(e_1) = m_A(e_2)$ and $s_L(e_1) \neq \partial$. Since they are distinct in L and identified in G, we must have distinct $e_1, e_2 \in B$ such that $c_A(e_1) = c_A(e_2)$ in C. By Lemma 3.4 this gives a self-loop at $\bar{\partial}$ in C, which in turn implies that $s_L(e_2) = \partial$. Hence L provides at most one candidate source vertex for every edge in G, and a similar argument can be made for C.

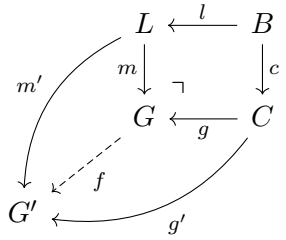
Now suppose $m_A(e_1) = g_A(e_2)$, and that $s_L(e_1) \neq \partial$ and $s_C(e_2) \neq \bar{\partial}$. Since the edges are identified in G they are both present in B. Since $s_L(e_1) \neq \partial$ we have $s_B(e_1) = \bar{\partial}$, from which $s_C(e_1) = \bar{\partial}$. Since $s_C(e_2) \neq \bar{\partial}$, e_1 and e_2 are distinct in C. Therefore we must have e_1 and e_2 identified in L; therefore, by Lemma 3.4, e_1 must be a self-loop at ∂ which contradicts our original assumption. Therefore there is at most one candidate source vertex and the map s_G is well defined in (6). The same argument applies to the target map t_G.

Next we show that every arc is either an edge (with two ends) or a circle (with none). Suppose, wlog, that for some edge $e \in A_G$ we have $s_G(e) = m_V(v)$; we want to show that $t_G(e)$ is also defined. If $t_L(e) \neq \partial$ then this is candidate, and by the preceding argument, the only one; hence $t_G(e)$ is defined. Otherwise, $t_L(e) = \partial$, which implies that $e \in E_B$, and its image in C either has $t_C(c_E(e)) \neq \bar{\partial}$ – in which case we are done – or it is a self loop. If it is a self loop we have another edge e' in B such that $c_E(e) = c_E(e')$; we can repeat this analysis with the image of e' in

L. Since the number of edges in B is finite this will eventually produce a candidate. Hence $s_G(e)$ is defined if and only if $t_G(e)$ is defined. Therefore the division of A_G into edges and circles is correct and G is indeed a valid graph.

Next we show that the map m is an embedding in $\mathcal{G}$. Note that Properties 4 and 1 are automatic from the underlying pushouts in **Pfn**. Since the graph B has no circles, the m_O component is injective by construction (Property 5) and since no arc gets a source or target in G unless its preimage had one, the component m_{OE} is empty as required (Property 2). Finally we have to show that the induced map (m_V, m_E) is a flag bijection. First note that if $m_E(e)$ is undefined then e is necessarily a self-loop at ∂, and $m_V(\partial)$ is always undefined, so the squares (2) commute. Otherwise if $(f_V \circ s_L)(e)$ is defined then the square commutes directly by the definition of s_G above, and similarly for t_G. Finally for all $v \neq \partial \in V_L$, we have that $m_V(v)$ is defined. By the definition of s_G and t_G, e is a flag at v if and only if $m_E(e)$ is a flag at $m_V(v)$. Flag injectivity and flag surjectivity follow immediately. Hence m is an embedding in $\mathcal{G}$. A symmetric argument will do the same for g.

Finally we must show that the cospan $L \xrightarrow{m} G \xleftarrow{g} C$ has the required universal property.



Since the underlying sets and functions are constructed via pushout the required mediating map $f = (f_V, f_A)$ exists; we need to show that it is a morphism of $\mathcal{G}$. Property 1 follows from m' and g' satisfying it as well. For the f_{OE} to be empty (Property 2), use the fact that m'_{OE} and g'_{OE} are empty for circles in L and C, because they are morphisms in $\mathcal{G}$. The remaining case for a circle to appear in G is as the pushout of some edges in B being identified in one instance of the re-pairing problem. In this case, because the outer square has to commute for the edge component, these edges have to be identified, and hence form a circle, in G', too. This makes f_{OE} empty. For flag surjectivity between the underlying graphs (Property 3), observe that the vertex set V_G is the disjoint union of vertex sets V_L and V_C. Because m' and g' are valid morphisms in $\mathcal{G}$, they are flag surjective, and therefore so is f. □

Lemma 3.11 Given a boundary embedding $B \xrightarrow{l} L \xrightarrow{m} G$ a solution to the re-pairing problem always exists, but it is not necessarily unique.

Proof. Note that any half-pairing graph has connected components of at most two vertices, linked by a (blue) edge from a positive vertex to a negative one. Define the component of an arc by

$$k(a) = (m_A \circ l_A)^{-1}(a) \text{ for all } a \in A_G$$

Note that this defines a partition of the set $E_B \simeq \sum_{a \in A_G} k(a)$, and each (non-empty) $k(a)$ determines a connected component of the solution to the re-pairing problem. We'll abuse notation and use $k(a)$ to also denote the subgraph of the half-pairing graph whose vertices are $k(a)$. There are two cases depending whether a is a circle or an edge.

1. Suppose $a \in O_G$; we form a closed loop involving all $e \in k(a)$, by adding red edges as follows. Pick a degree-one positive vertex p follow the incident blue edge to the negative vertex n; now pick another a degree-one positive vertex p' which is not connected to n. Add a red edge from n to p'. Repeat the process starting from p'. When no more vertices remain, close the loop by adding a red edge from the final negative vertex back to p. Since a is a circle, $k(a)$ necessarily contains an even number of vertices, so closing the loop is always possible.
2. The case when a is an edge is slightly more complex because edges have end points; $k(a)$ may contain zero, one, or two degree-zero vertices depending how many of its end points are defined by vertices in L. We will connect the vertices as previously, but in a line, rather than a loop. Since we can only add red edges, and only one at each vertex, the degree-zero vertices will necessarily be the end points of this line.

□

B Examples

This collection captures some of the corner cases we do or do not want to allow as morphisms in the category of graphs with circles as described in Definition 2.10.

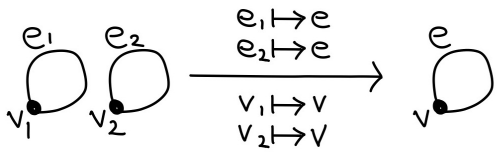


Figure 7: A flag surjective, but not flag bijective map. This is a valid of $\mathcal{G}$.

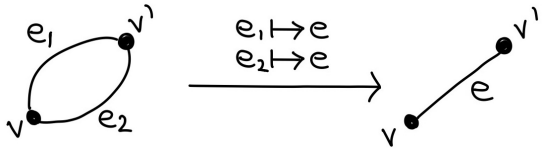


Figure 8: An example of a flag surjective but not flag bijective morphism of $\mathcal{G}$.

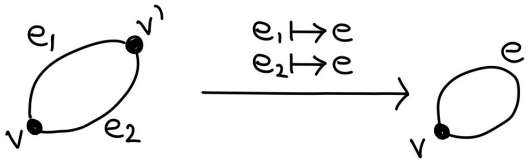


Figure 9: An example of an embedding (hence a flag bijective morphism) in $\mathcal{G}$.

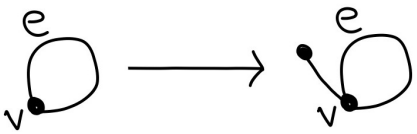


Figure 10: This map is not flag surjective and therefore not a valid morphism in $\mathcal{G}$.

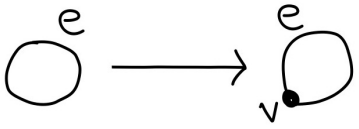


Figure 11: This example is not a valid morphism in $\mathcal{G}$ because it does not respect Condition 2.