A Category of Surface-Embedded Graphs

Malin Altenmüller, Ross Duncan

ACT 2022

Why Graphs?

- string diagrams syntaxes for monoidal categories
- want combinatorial representation to implement rewriting
- use (some form of) graphs and their morphisms

Why Graphs?

- string diagrams syntaxes for monoidal categories
- want combinatorial representation to implement rewriting
- use (some form of) graphs and their morphisms

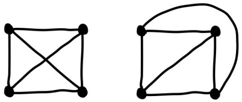
here: vertices represent generators, edges represent wires

Why Surface-Embedded Graphs?

- monoidal theories may require non-trivial topology of graphs
  in particular non-symmetric theories
- example: string diagrams for quantum processes

Why Surface-Embedded Graphs?

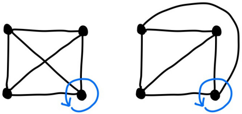
- monoidal theories may require non-trivial topology of graphs in particular non-symmetric theories
- example: string diagrams for quantum processes
- easiest case: plane graph embeddings



- working at the level of the *embedding*

Representing Graph Embeddings

Rotation Systems fix order of edges around vertices

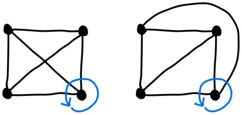


Theorem

Rotation systems uniquely determine a graph embedding.

Representing Graph Embeddings

Rotation Systems fix order of edges around vertices



Theorem

Rotation systems uniquely determine a graph embedding.

plan: construct a category of graphs, then add rotation information

Diagram Rewriting

$$
\begin{tikzcd}
L \arrow[d] & B \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G \arrow[ur, phantom, "\urcorner", very near start] & G \setminus L \arrow[l] \arrow[r] & G[R/L] \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}
$$

- rewrite rules are $L \Rightarrow R$, with common boundary B
- double-pushout diagram, all maps are embeddings
- $C = G \setminus L$: context with a hole

Diagram Rewriting

$$
\begin{tikzcd}
G \setminus C \arrow[d] & B \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G \arrow[ur, phantom, "\urcorner", very near start] & C \arrow[l] \arrow[r] & G[R/L] \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}
$$

- rewrite rules are $L \Rightarrow R$, with common boundary B
- double-pushout diagram, all maps are embeddings
- $C = G \setminus L$: context with a hole
- $L = G \setminus C$: LHS with a "hole"

Diagram Rewriting

$$
\begin{tikzcd}
G \setminus C \arrow[d] & B \arrow[l] \arrow[r] \arrow[d] & R \arrow[d] \\
G \arrow[ur, phantom, "\urcorner", very near start] & C \arrow[l] \arrow[r] & G[R/L] \arrow[ul, phantom, "\ulcorner", very near start]
\end{tikzcd}
$$

- rewrite rules are $L \Rightarrow R$, with common boundary B
- double-pushout diagram, all maps are embeddings
- $C = G \setminus L$: context with a hole
- $L = G \setminus C$: LHS with a "hole"
- need: pushouts, pushout complements, notion of embedding

Category of Graphs

We start from the usual category of graphs:

- graphs are $E \underset{t}{\overset{s}{\rightrightarrows}} V$
- morphisms are pairs of edge map f_E and vertex map f_V s.t.

$$\begin{tikzcd} E \arrow[r, "f_E"] \arrow[d, "s"'] & E' \arrow[d, "s'"] \\ V \arrow[r, "f_V"'] & V' \end{tikzcd} \qquad \begin{tikzcd} E \arrow[r, "f_E"] \arrow[d, "t"'] & E' \arrow[d, "t'"] \\ V \arrow[r, "f_V"'] & V' \end{tikzcd}$$

Category of Graphs

We start from the usual category of graphs:

- graphs are $E \underset{t}{\overset{s}{\rightrightarrows}} V$
- morphisms are pairs of edge map f_E and vertex map f_V s.t.

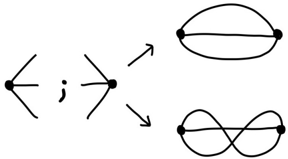
$$\begin{tikzcd} E \arrow[r, "f_E"] \arrow[d, "s"'] & E' \arrow[d, "s'"] \\ V \arrow[r, "f_V"'] & V' \end{tikzcd} \qquad \begin{tikzcd} E \arrow[r, "f_E"] \arrow[d, "t"'] & E' \arrow[d, "t'"] \\ V \arrow[r, "f_V"'] & V' \end{tikzcd}$$

Disclaimer

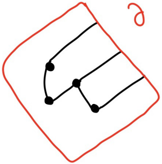
(Almost) all graphs are drawn undirected in this presentation.

Open Graphs

- have to encode inputs and outputs of the diagrams
- different approaches: open graphs, representative vertices, cospans
- morphisms for open graphs don't preserve the surface:

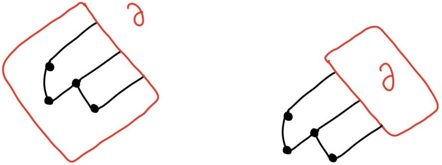


Boundary Vertices



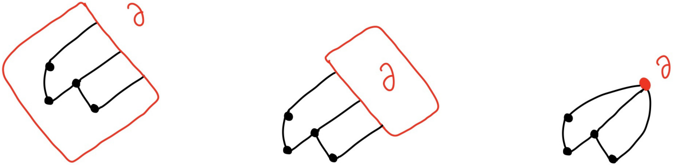
- identify the "outside" of a graph
- attach input and output edges to this region

Boundary Vertices



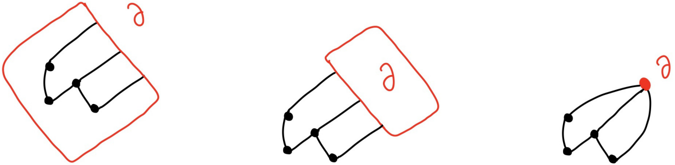
- identify the "outside" of a graph
- attach input and output edges to this region

Boundary Vertices



- identify the “outside” of a graph
- attach input and output edges to this region
- replace the outside with a vertex

Boundary Vertices

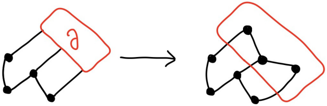


- identify the "outside" of a graph
- attach input and output edges to this region
- replace the outside with a vertex

This provides:

- total graphs
- strategy to deal with the outside, and any holes in a graph

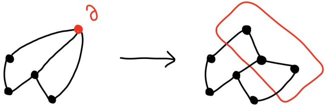
Requirements for Graph Morphisms(1)



What are embeddings?

Requirements for Graph Morphisms(1)

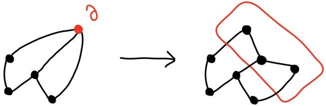
- vertex map needs to be partial



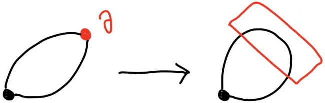
What are embeddings?

Requirements for Graph Morphisms(1)

- vertex map needs to be partial



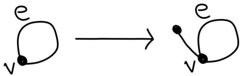
- cannot be injective on edges



What are embeddings?

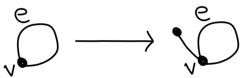
Requirements for Graph Morphisms(2)

- vertices must not change their arity



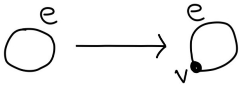
Requirements for Graph Morphisms(2)

- vertices must not change their arity

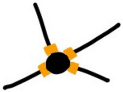


- morphisms from edges to loops are allowed

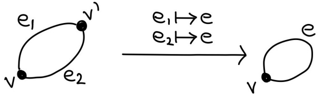
but the other way is not



Flags



- connection points between vertices and their incident edges, pairs (v, e)
- flag map (f_E, f_V) *partial* map induced by graph map



- characterize morphisms/embeddings by properties of the flag map

Graphs with Circles

Objects are total graphs, as defined above
Morphisms are (f_E, f_V) where

- the flag map is surjective
  (no increase in flags at a vertex)
- $+$ other conditions

Further graph, embeddings are

- flag injective (no decrease in flags at a vertex)
- $+$ other conditions

Graphs with Circles

Objects are total graphs, as defined above
Morphisms are (f_E, f_V) where

- the flag map is surjective
  (no increase in flags at a vertex)
- $+$ other conditions

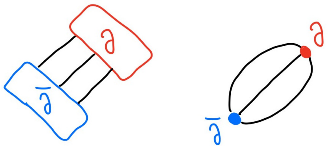
Further graph, embeddings are

- flag injective (no decrease in flags at a vertex)
- $+$ other conditions

It's a category!

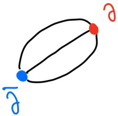
Rewriting for Graphs with Circles

specify the cases for applying a rewrite rule
Boundary graph:



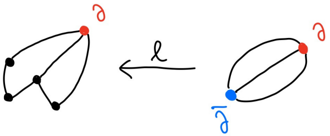
Partitioning Spans

partition a graph into two (connected) parts: context and subgraph



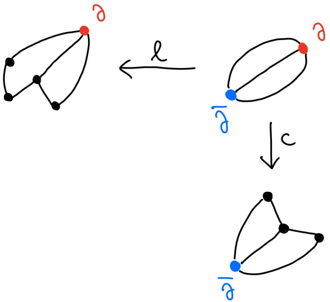
Partitioning Spans

partition a graph into two (connected) parts: context and subgraph



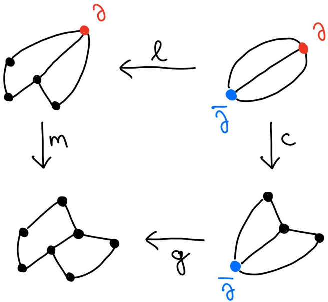
Partitioning Spans

partition a graph into two (connected) parts: context and subgraph



Partitioning Spans

partition a graph into two (connected) parts: context and subgraph

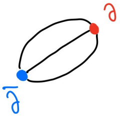


Theorem

Pushouts of partitioning spans exist, and all morphisms in the pushout square are embeddings.

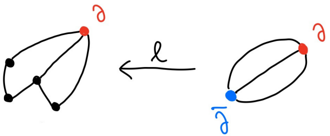
Boundary Embeddings

for constructing pushout complements which give rise to partitioning spans



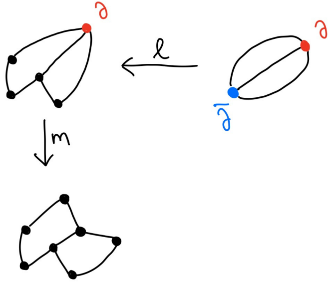
Boundary Embeddings

for constructing pushout complements which give rise to partitioning spans



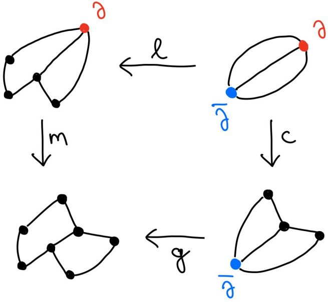
Boundary Embeddings

for constructing pushout complements which give rise to partitioning spans



Boundary Embeddings

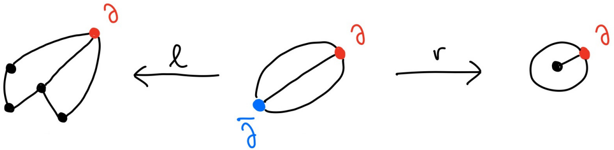
for constructing pushout complements which give rise to partitioning spans



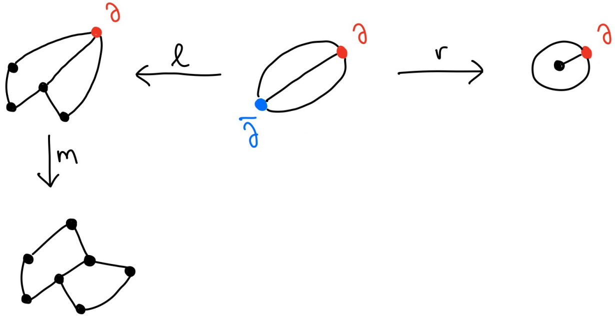
Theorem

Pushout complements of boundary embeddings exist and are unique (up to degeneracies).

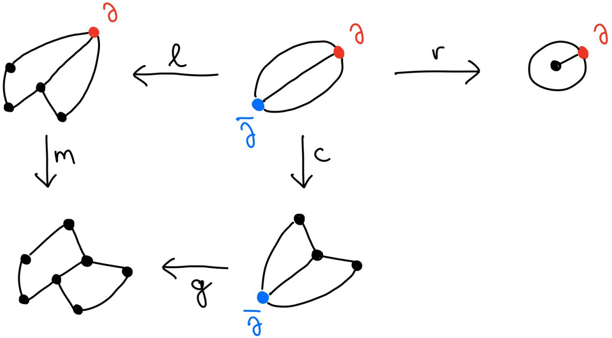
An Example of Applying a Rewrite Rule



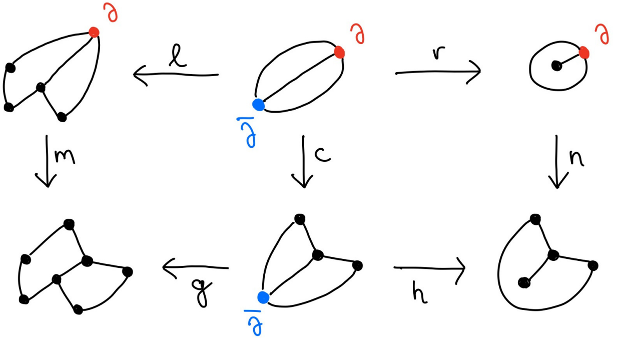
An Example of Applying a Rewrite Rule



An Example of Applying a Rewrite Rule



An Example of Applying a Rewrite Rule



Category of Rotation Systems

so far, edges around vertices were *sets*...

- objects are rotation systems: assign to a cyclic ordering of flags to all vertices
- morphisms are morphisms in the underlying category of graphs, plus an order preservation condition

Category of Rotation Systems

so far, edges around vertices were *sets*...

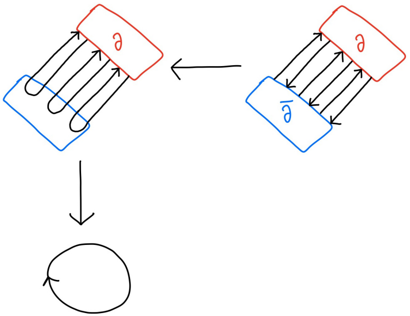
- objects are rotation systems: assign to a cyclic ordering of flags to all vertices
- morphisms are morphisms in the underlying category of graphs, plus an order preservation condition

Theorem

Pushouts and pushout complements are the same as in the underlying category.

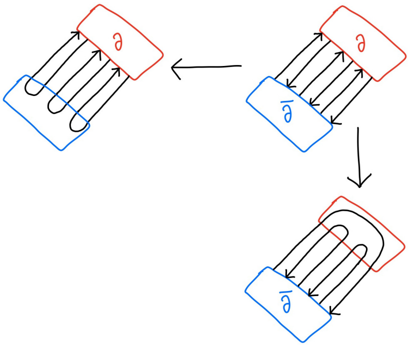
Let's talk about Loops!

problem: construct a pushout complement of a loop



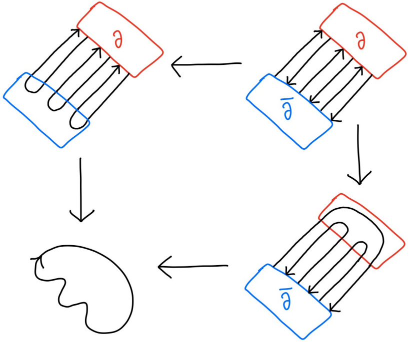
Let's talk about Loops!

problem: construct a pushout complement of a loop



Let's talk about Loops!

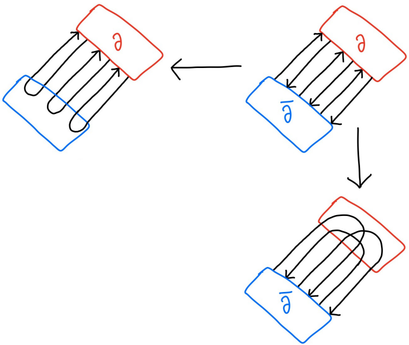
problem: construct a pushout complement of a loop



has a plane solution

Let's talk about Loops!

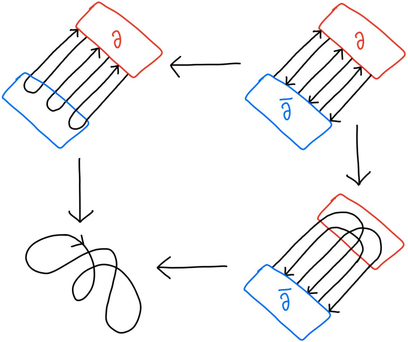
problem: construct a pushout complement of a loop



has a plane solution

Let's talk about Loops!

problem: construct a pushout complement of a loop



has a plane solution
and a non-plane solution

Summary

- fix inputs and outputs to control topology
- restrict your rewrite rules to meaningful cases
- category of graphs with circles extendable to rotation systems

Summary

- fix inputs and outputs to control topology
- restrict your rewrite rules to meaningful cases
- category of graphs with circles extendable to rotation systems

Future Thoughts

- How about surface-embedded loops?
- How about multiple boundary vertices?

Summary

- fix inputs and outputs to control topology
- restrict your rewrite rules to meaningful cases
- category of graphs with circles extendable to rotation systems

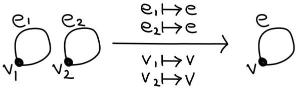
Future Thoughts

- How about surface-embedded loops?
- How about multiple boundary vertices?

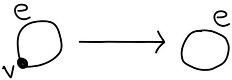
Thank You for Your Attention!

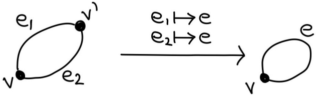
Appendix: Examples

Valid morphisms:

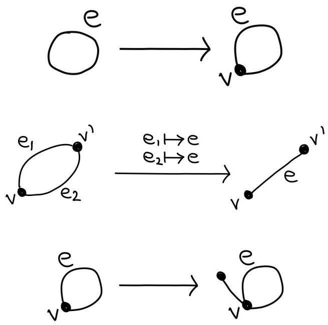


Embeddings:





Appendix: Non-Examples



Appendix: Graphs with Circles

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:

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

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

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

Appendix: Flag Surjectivity

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,

$$\begin{tikzcd} V \arrow[r, harpoon, "f_V"] \arrow[d, "s^{-1}"'] \arrow[dr, phantom, "\geq"] & V' \arrow[d, "s'^{-1}"] \\ P(E) \arrow[r, "P(f_E)"'] & P(E') \end{tikzcd} \qquad \begin{tikzcd} V \arrow[r, harpoon, "f_V"] \arrow[d, "t^{-1}"'] \arrow[dr, phantom, "\geq"] & V' \arrow[d, "t'^{-1}"] \\ P(E) \arrow[r, "P(f_E)"'] & P(E') \end{tikzcd}$$