Combinatorial Notions of Plane String Diagrams

Malin Altenmüller

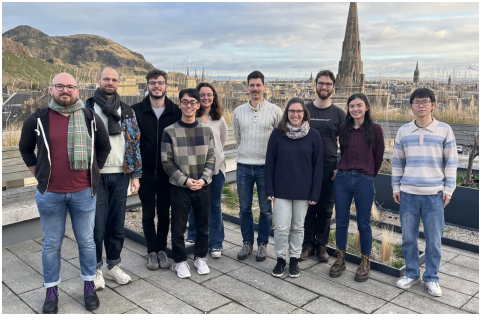
Theoretical Computer Science Seminar, Birmingham 26th July 2026

THE UNIVERSITY of EDINBURGH
informatics

Hello, I'm Malin!

I'm a postdoc at the University of Edinburgh, working on quantum programming languages and their type systems.

I did my PhD at Strathclyde University in Glasgow, supervised by Conor McBride and Ross Duncan, on plane string diagrams.



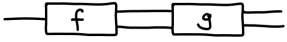
Research interests: graphical languages, type theory, proof assistants, programming languages and their semantics.

Monoidal categories

Formalism to express systems that can be composed both in sequence and in parallel.

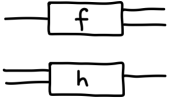
- sequential composition of maps $f : A \to B$ and $g : B \to C$ by function composition:

$$f \fatsemi g : A \to C$$



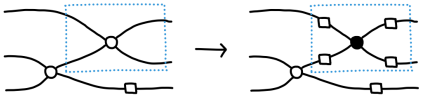
- parallel composition of maps $f : A \to B$ and $h : C \to D$ by tensor product:

$$f \otimes h : A \otimes C \to B \otimes D$$



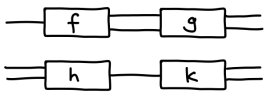
String diagrams

- String diagrams provide a graphical syntax for morphisms of monoidal categories.
- Reasoning about morphisms by rewriting.



- Coherence: Any valid morphism in the category is expressible by a valid diagram, and nothing else.
  Example Interchange Law:

$$(f \otimes h) \fatsemi (g \otimes k) = (f \fatsemi g) \otimes (h \fatsemi k)$$



Combinatorial presentations of string diagrams

- Elements of the diagrammatic language as mathematical objects.
- Especially important: modelling rewriting theories for string diagrams.
- Properties of the category have to be matched by its diagrams.

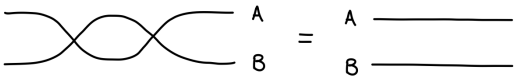
Definition

A graph G is a tuple (V, E, s, t) with a set of vertices V, a set of edges E, source and target functions $s, t : E \to V$.

A graph morphism $f : G \to H$ consists of maps $f_V : V_G \to V_H$ and $f_E : E_G \to E_H$ which preserve source and target functions.

- Diagram rewriting implemented by graph rewriting.
- Translation: wires → edges, boxes → vertices. Any map has to preserve a vertex' arity.

Example: symmetric monoidal categories (SMC)



- Have a braiding operation $\sigma : A \otimes B \to B \otimes A$. such that $\sigma_{B,A} \fatsemi \sigma_{A,B} = \mathrm{id}\, A \otimes B$.
- Placement of edges does not matter, only the connections at their ends.

Only connectivity matters.

- Combinatorial presentation: graph $G = (V, E, s, t)$ is sufficient.

Non-symmetric monoidal categories

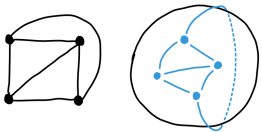
- Express theories do not contain a symmetry operation.
- Quantum circuits: expensive SWAP operations must be recorded explicitly.
- Abstraction of braided and symmetric monoidal categories.
- Have to represent the structure of wires in the theory.

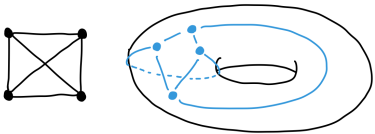
Connectivity *and topology* matter.

- Combinatorial presentation: plane graph embeddings.

Surface-embeddings of graphs

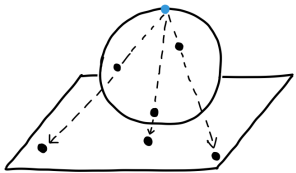
A graph's surface-embedding is characterised by the arrangement of its faces.





Definition

A graph is *planar* if it can be embedded into the surface of a sphere. The embedding itself is called a *plane* graph.



Plan for this talk

Two different combinatorial representation of graph embeddings.

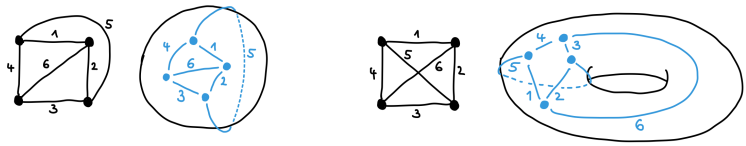
1. A category of surface-embedded graphs.
   - Based on rotation systems.
   - Introduce notion of boundary vertex.
   - Implement rewriting by double pushout.

2. A data type of plane graphs.
   - Based on spanning trees of graphs.
   - Intrinsic notion of planarity.

A category of of surface-embedded graphs

Representing surface-embeddings of graphs — rotation systems

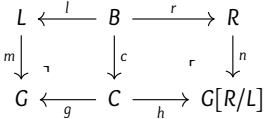
Rotation systems specify the order of edges around vertices.

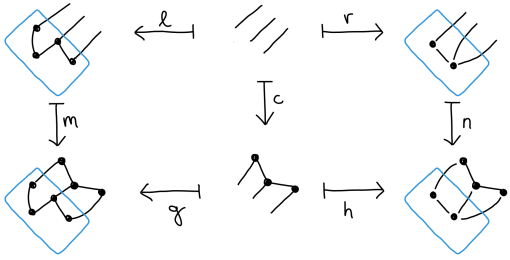


- Can compute the faces from the rotations.
- Potential extensions for non-orientable surfaces.
- Can use existing work on standard graph rewriting.

Goal: Develop category of graphs which admits the addition of rotation systems.

Double-pushout rewriting

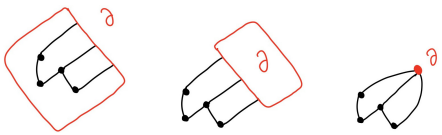




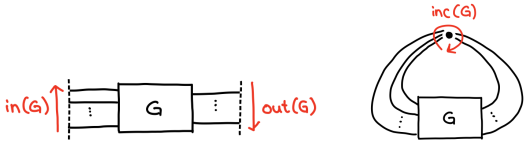
- Rewrite rule $L \Rightarrow R$, applied to graph G.
- Remove L from G, then insert R into the hole.
- B is a *boundary graph*, C is a *context graph* with a hole of size B.
- In adhesive categories, DPO rewriting behaves well.

Challenge 1: Open graphs

A graph may have *interface* edges, which are not connected to a vertex.



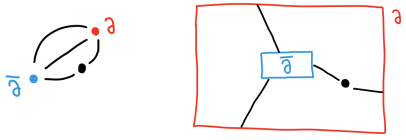
- Boundary vertex treated as part of the graph. All graphs are total.
- Can add rotation information to a graph's interface edges.



Outer and inner boundaries

Represent both interfaces as well as holes with boundary vertices.

A graph with an outer and an inner boundary:



Challenge 2: Partial graph morphisms

Embedding a graph into a context means replacing the boundary vertex with a graph:

Therefore, graph morphisms have to be *partial on vertices*.

A morphism is sometimes not even injective on edges. Consider the identity graph:

What is the correct notion of graph embedding?

Flags and flag maps

Flags and flat maps are a derived notion:

Definition

The flags of a graph G are pairs $(v, e) \in V \times E$ with $s(e) = v$ or $t(e) = v$.
Every graph morphism f induces a flag map, $f_E \times f_V$.

- Flag map is a partial map (because f_V is).
- No increase of flags at a vertex: flag surjectivity.
- No decrease of flags at a vertex: flag injectivity.

Category of graphs with circles

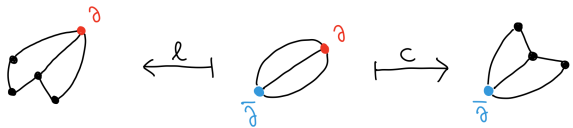
- Objects: total graphs.
- Morphisms have a partial vertex component.
- Induced *flag map* records the end points of edges.
- No increase of flags at a vertex: flag surjectivity.
- No decrease of flags at a vertex: flag injectivity.

Challenge 3: Rewriting

- The category of graphs with circles is not adhesive.
- But we can specify which pushouts we care about:

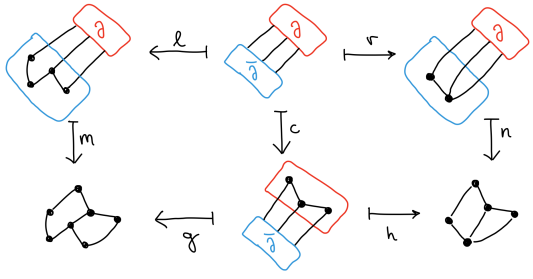
Definition

A partitioning span is a span of maps $l : B \to L$, $c : B \to C$, where l is undefined on $\bar{\partial}$ but defined otherwise, and c is undefined on ∂ but defined otherwise.



- In our category of graphs, pushouts of partitioning spans exist.
- Pushout complements of partitioning spans exist and are unique*.

Example of a rewriting step, using the notion of partitioning span:

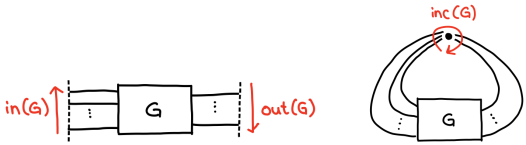


Adding rotation systems

- As all graphs are total, we can easily add rotation information to all its vertices.
- Instead of a set of edges around each vertex, store a *cyclic list*, preserved by a morphism.
- All constructions translate: partitioning spans, pushouts, pushout complements.

PRO of plane graphs:

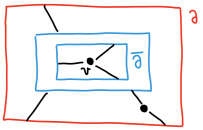
- objects: natural numbers n, m.
- morphisms $n \to m$: graphs with a boundary vertex with rotation of size $n + m$.

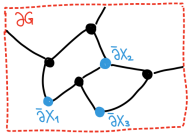


- We can prove that composition and tensor product preserve planarity.

What else?

What about a more complex boundary structure? Multiple holes? Multiple boundary vertices? Boundary graph?





Multicategories: A graph is a map in a multicategory from the type of its holes to the type of its outer boundary. Composition is substitution of graphs.
(Ask me about pattern matching later, if you're curious.)

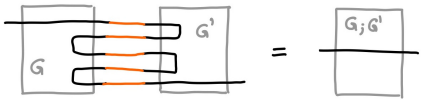
Plane graphs in Agda

Plane graphs as a data type?

Goal: implementation of plane graphs and their rewriting theory in Agda

Various challenges:

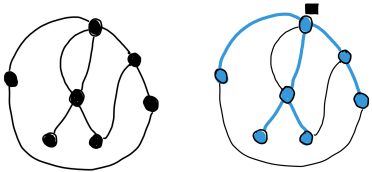
- Composition is really nice on paper, but not in a term based tool:



- Graphs are cyclic, but we would like an inductive type.
- How to enforce the planarity?
- How to implement rewriting?

Spanning trees to the rescue

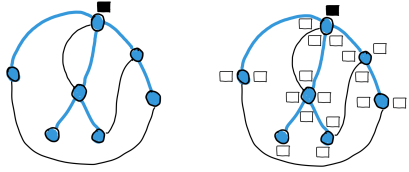
A largest cycle-free subgraph of a graph is one of its spanning trees.



graph = spanning tree (incl. root) + looping edges

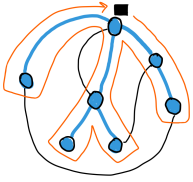
An inductive data type

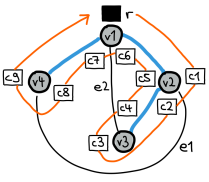
graph = spanning tree (incl. root) + looping edges + corners



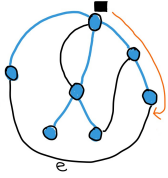
An ordered data type

- We represent a graph as the clockwise (depth-first) traversal of its spanning tree.
- Follow along the spanning edges, while storing the looping eges alongside.

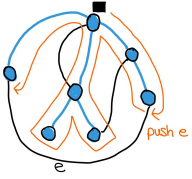




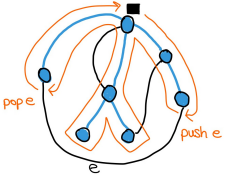
A stack of looping edges



[]



[e…]



[]

The Data type of Plane Graphs

A graph is encoded by a clockwise traversal of its spanning tree, together with a stack of its looping edges.
Potential traversal steps:

- Follow a spanning tree edge.
- Record a corner.
- Record a looping edge, either push or pop it.

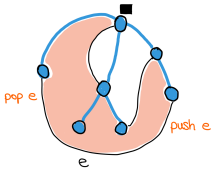
In Agda, we index the type of steps by the *change of edge stack* it induces.

Rewriting a subgraph

- Using a zipper for the graph's spanning tree.
- *Separate* the subgraph and its context graph (cf. DPO rewriting).
- Have to preserve the stack discipline at all times.

Stack structure determines faces

Every non-tree edges closes a face of the graph embedding:



We can calculate the faces of the embedding by observing the changes of the edge stack.

Theorem

A stack of non-tree edges ensures planarity of a graph.

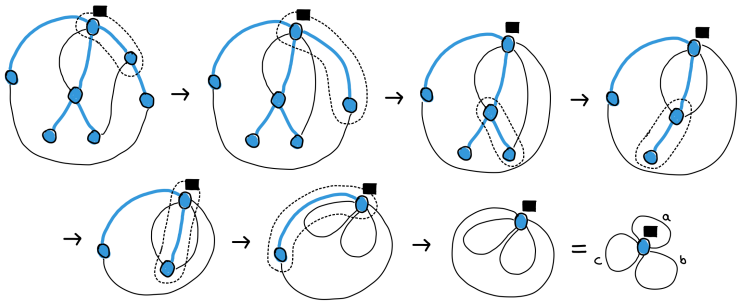
To prove this, we use the following fact:

Lemma

Contracting a plane subgraph does not change the genus of a graph's embedding.

- Plan: contract the entire spanning tree of a graph.
- All the surface information is stored in the non-tree edges of a graph.

Contracting the spanning tree



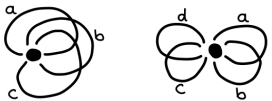
Looping edges form a well bracketed word `abbcca`.
(cf. context-free grammars, Dyck language,…)

Higher-genus graph embeddings?

Open question: Which structure of looping edges represents higher-genus surface embeddings?

Forbidden minor for the sphere:

Two stacks are not enough for the torus, and and the same time too much:



Summary

1. Categories of surface-embedded graphs.
   - Category of graphs that can be equipped with rotation systems.
   - Introduce notion of boundary vertex to connect interface edges.
   - Notion of partioning spans for DPO rewriting.
2. A data type for plane graphs.
   - Separate a graph into its spanning tree plus looping edges: cycle-free, inductive representation.
   - Stack of looping edges guarantees plane graph embedding intrinsically.
   - My favourite open problem: How to encode higher-genus graph embeddings?

Thank you for your attention!

maltenmuller.github.io