A Combinatorial Presentation of the Operad of Plane Graphs

Malin Altenmüller[1]   Ross Duncan[1,2]

STRINGS 3 in Birmingham, 4 September 2019

1 University of Strathclyde Glasgow

2 CAMBRIDGE QUANTUM COMPUTING

Diagrams for Monoidal Categories

- string diagrams graphical language for monoidal categories
- represented by graphs (Selinger, 2011)
  - SMC: directed acyclic graphs
  - traced: can contain cycles
  - autonomous: wires can go the other way
- diagram equality → graph isomorphism
- equational reasoning via rewrite rules
  → double pushout graph rewriting

Diagrams for non-symmetric Monoidal Categories

Non-symmetric case:

- printing circuits: crossings not possible
- quantum circuits: crossing not for free
- most general case: can add structure on top

Representation:

- no crossing wires
- represented by plane graphs

Implementation:

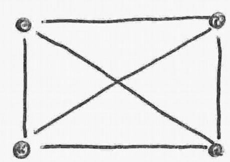
This project is being implemented in Agda

Plane Graphs

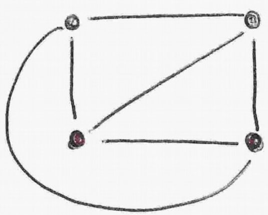
Definition:

- graph $G = (V, E)$ consisting of vertices and edges
- embedding of G: drawing of G on a surface S
- G planar if there exists an embedding into the plane without crossing edges. The embedding is called plane.

planar graph G:



plane embedding of G:

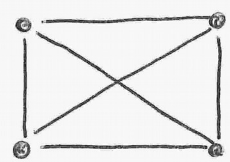


Plane Graphs

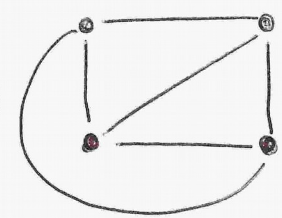
Definition:

- graph $G = (V, E)$ consisting of vertices and edges
- embedding of G : drawing of G on a surface S
- G planar if there exists an embedding into the plane without crossing edges. The embedding is called plane.

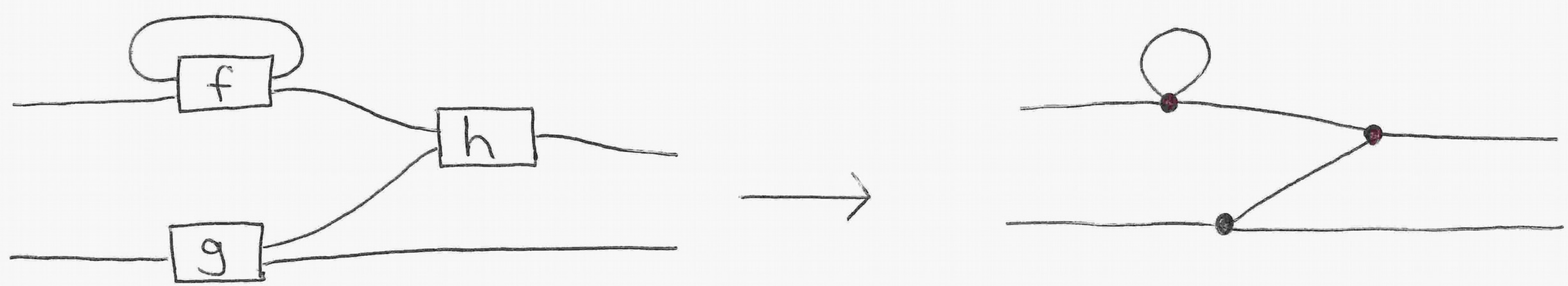
planar graph G :



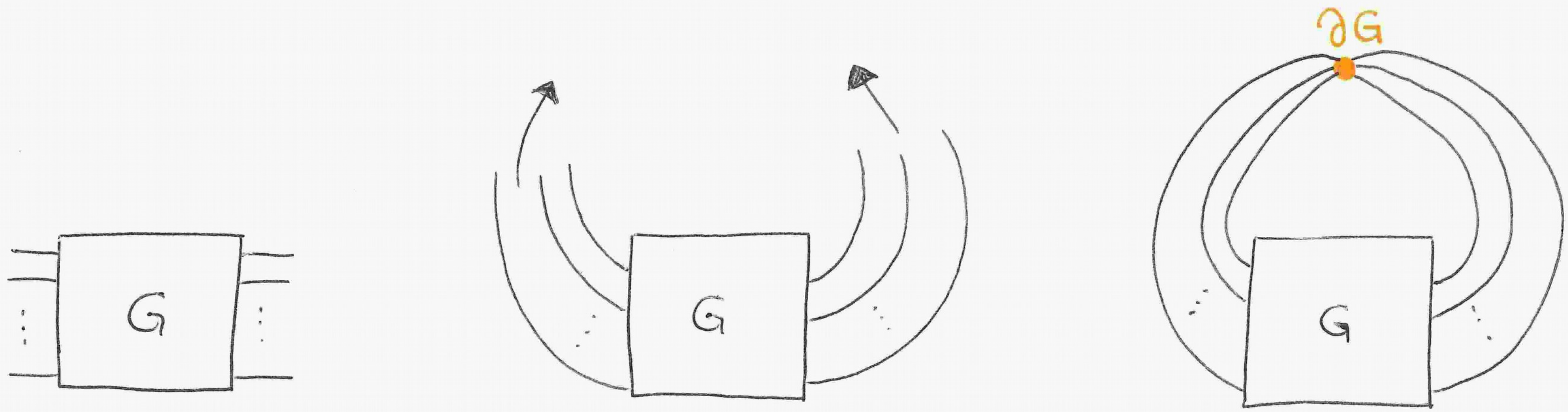
plane embedding of G :



A PRO can be represented as open plane graph:



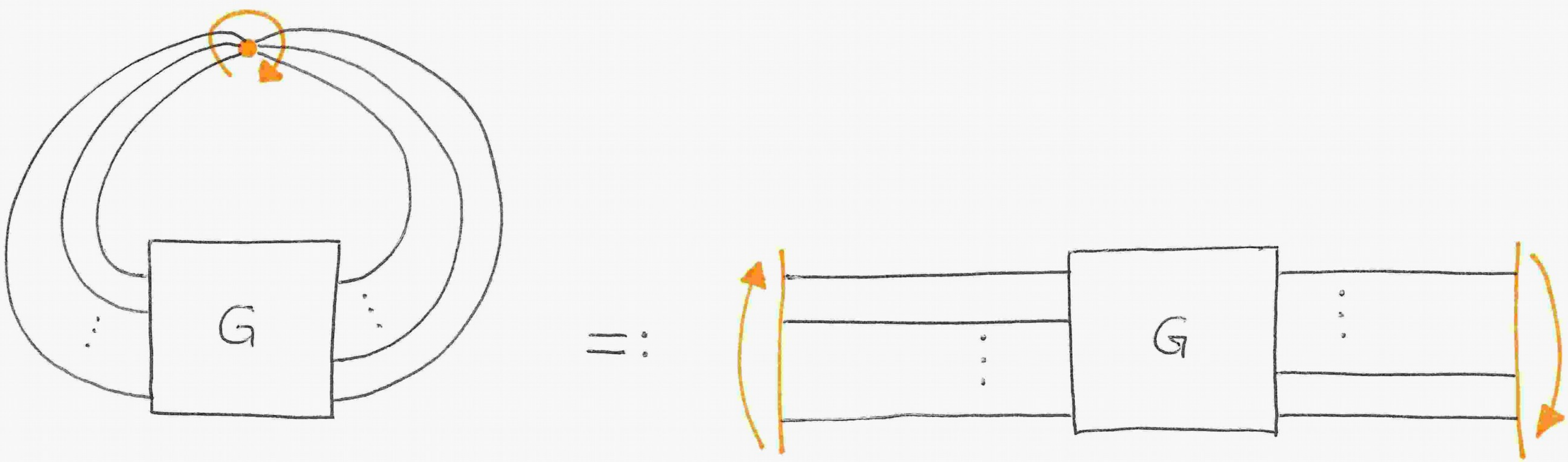
Plane Graphs with a Boundary Vertex (1)



Encoding the dangling input and output wires:

- introduce a new vertex ∂G, the boundary vertex
- making the boundary part of the graph

Plane Graphs with a Boundary Vertex (2)



- boundary vertex part of the plane graph
- connecting parallel graphs
- nice way to represent plane graphs combinatorically

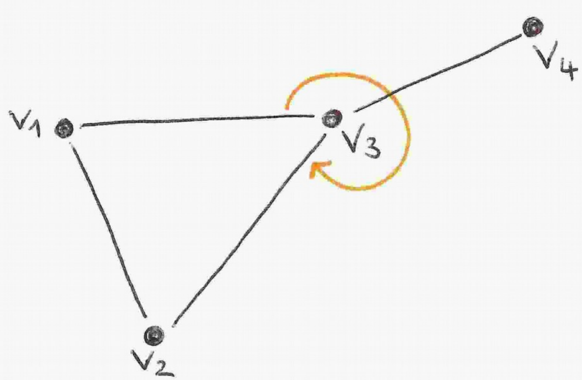
Rotation Systems (1)

Definition:

- rotation of a vertex $v \in V$:

  cyclic ordered list of adjacent vertices
- rotation system: rotation for all vertices in the graph

Here: rotation in clockwise direction



$v_1 : v_2, v_3$

$v_2 : v_1, v_3$

$v_3 : v_1, v_4, v_2$

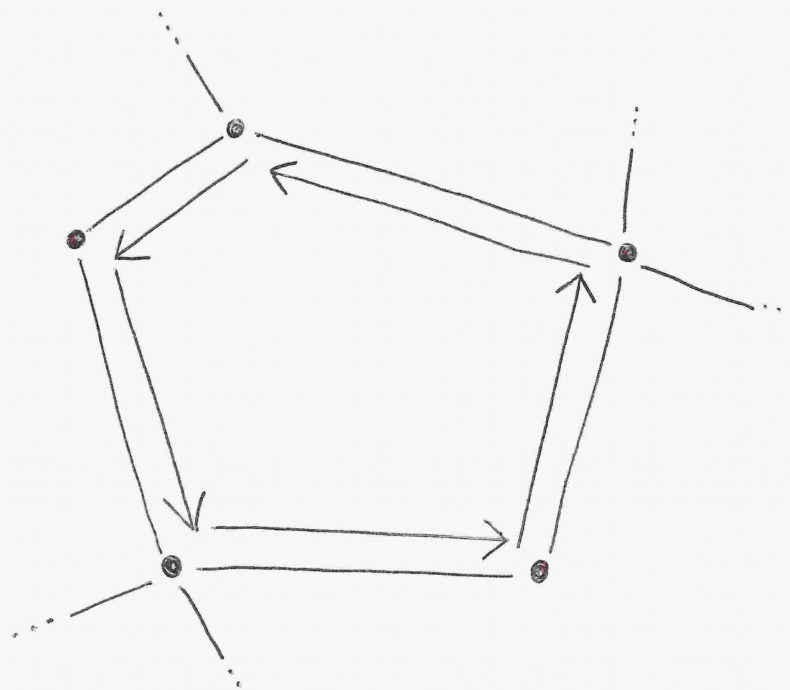
$v_4 : v_3$

Rotation Systems (2)

Lemma:

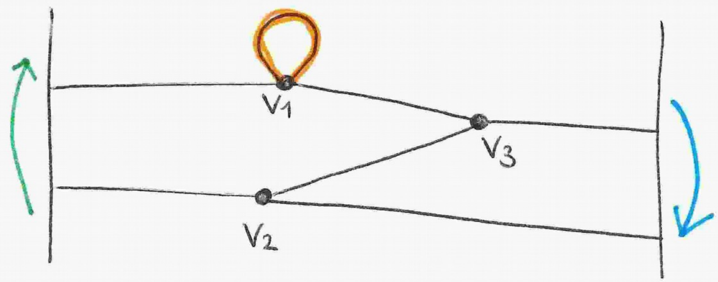
A rotation system uniquely defines a (cellular) embedding of a graph (Youngs, 1963)

Proof:



Rotation Systems (3)

Example:



v_1 : in, v_1, v_1, v_3

v_2 : in, v_3, out

v_3 : v_1, out, v_2

in : v_2, v_1

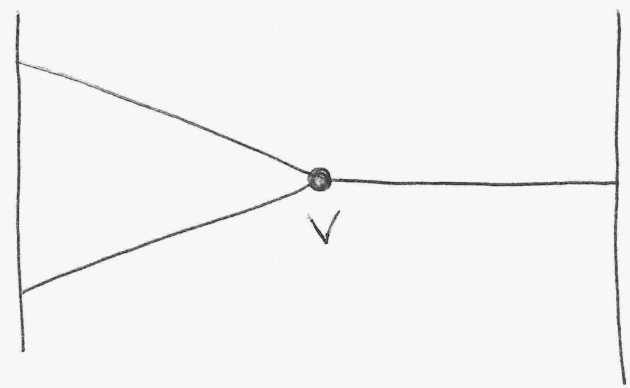
out : v_3, v_2

- categorically: inputs and outputs non-cyclic ordered lists
- combinatorically: boundary vertex as cyclic ordered list
- special case: multiple self loops (later, maybe)

boundary and inner vertices

Building Graphs – Base Cases (1)

Single vertex:



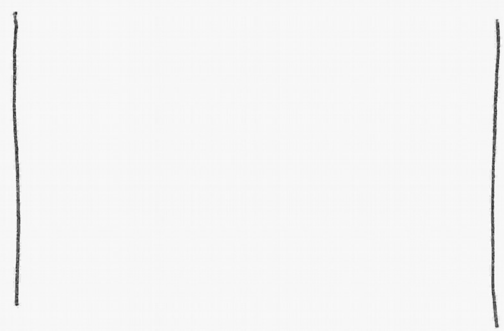
v : in, in, out

in : v, v

out : v

Building Graphs – Base Cases (2)

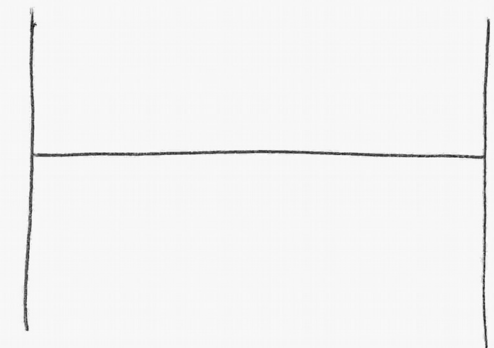
empty graph:



in: []

out: []

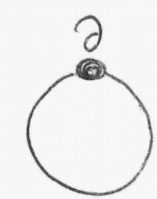
identity:



in: out

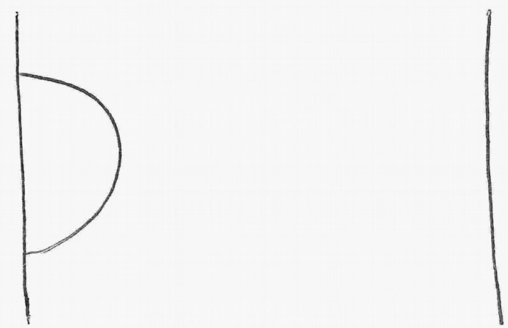
out: in

=



Building Graphs – Base Cases (3)

cap:

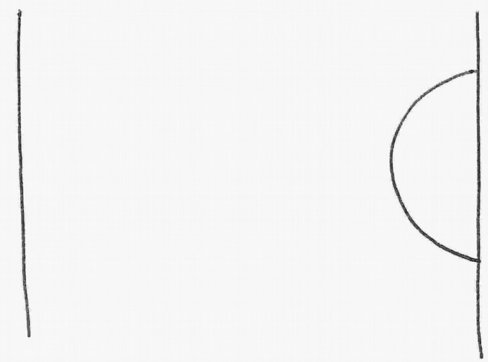


in: in, in

out: []

= 

cup:



in: []

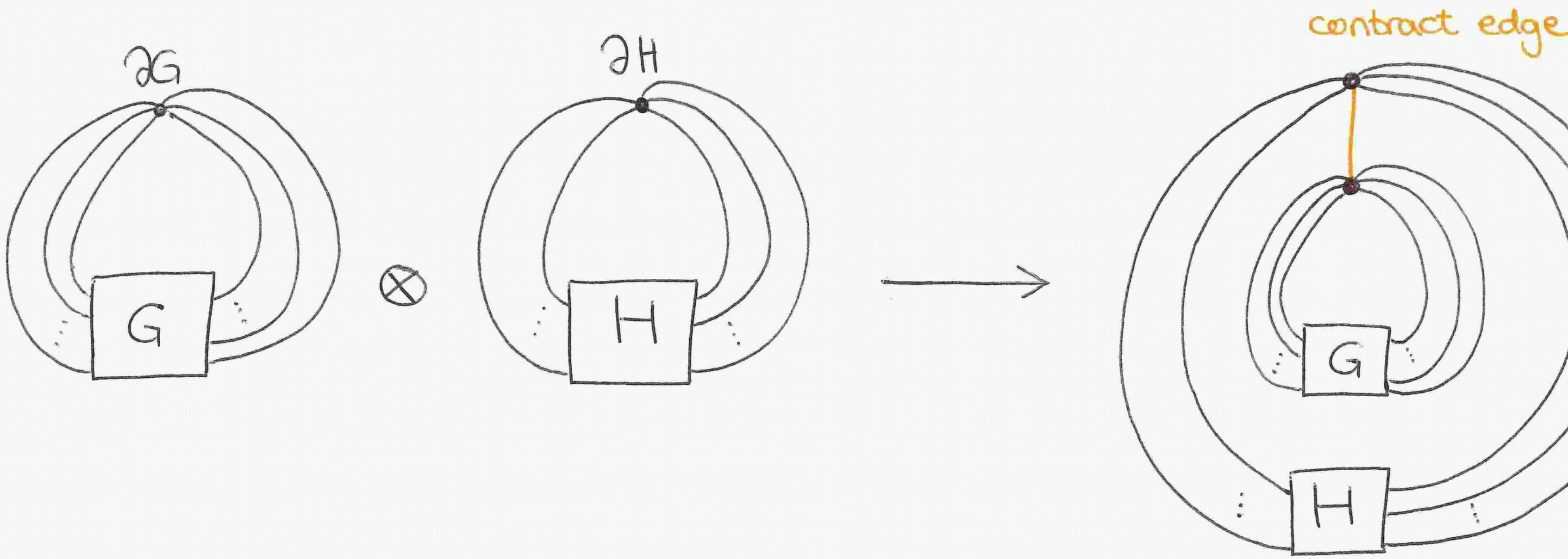
out: out, out

= 

cap and cup are self loops at the boundary vertex

(so is the identity!)

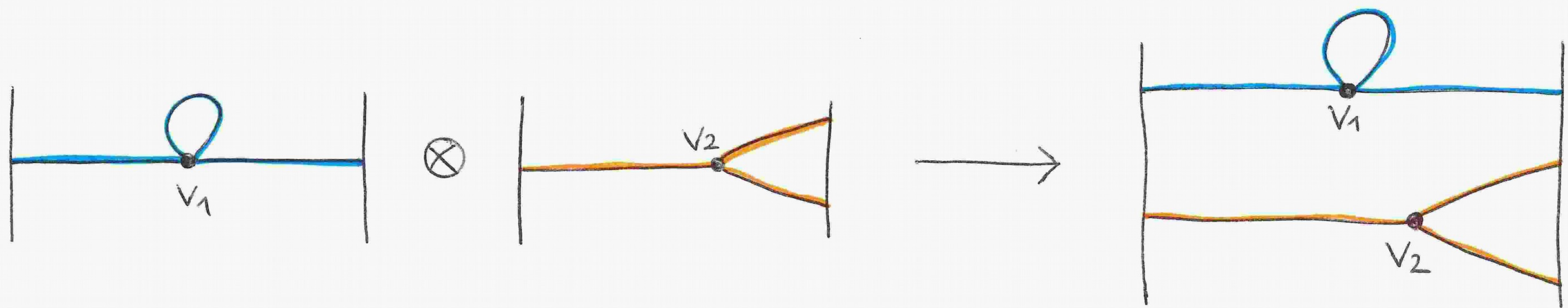
Building Graphs – Parallel Composition (1)



- make names of vertices disjoint
- new rotation system: union of both rotation systems
- new boundary vertex: draw extra edge and contract it

Building Graphs – Parallel Composition (2)

Example:



v_1 : in, v_1, v_1, out

in : v_1

out: v_1

v_2 : in, out, out

in : v_2

out: v_2, v_2

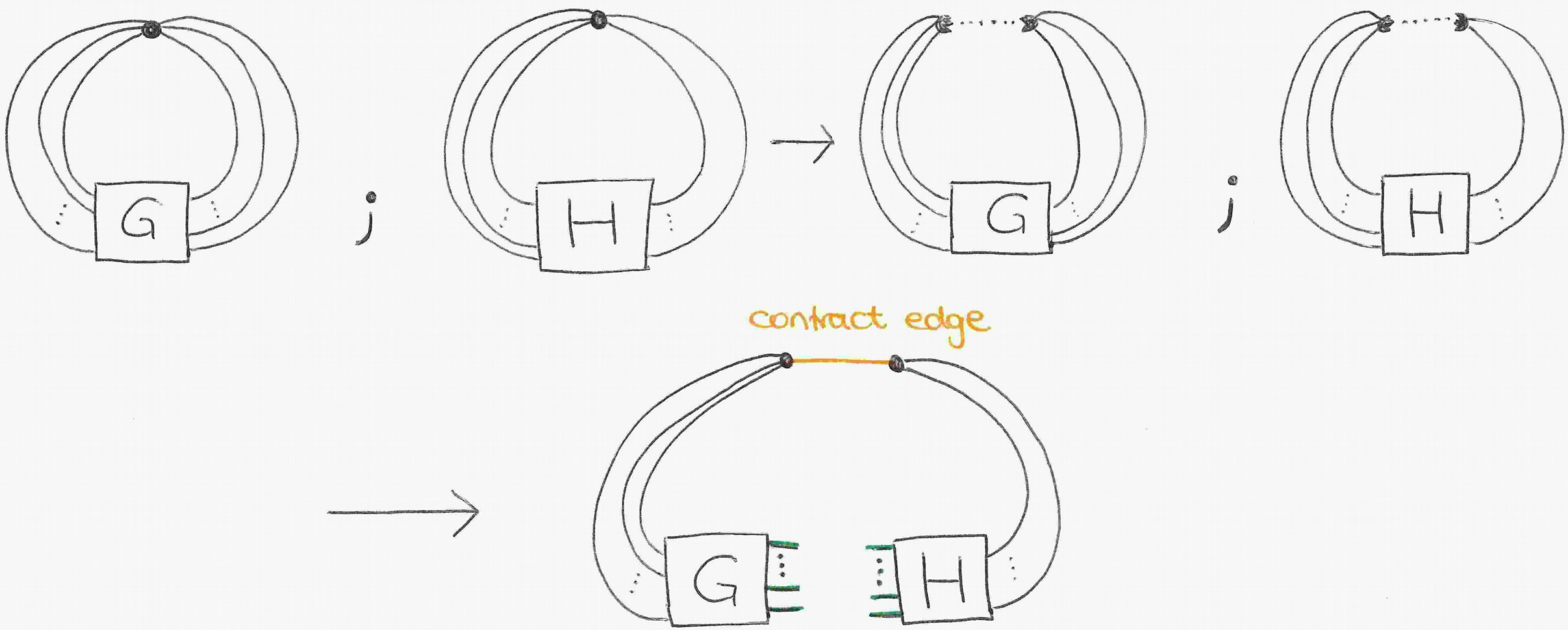
v_1 : in, v_1, v_1, out

v_2 : in, out, out

in : v_2, v_1

out: v_1, v_2, v_2

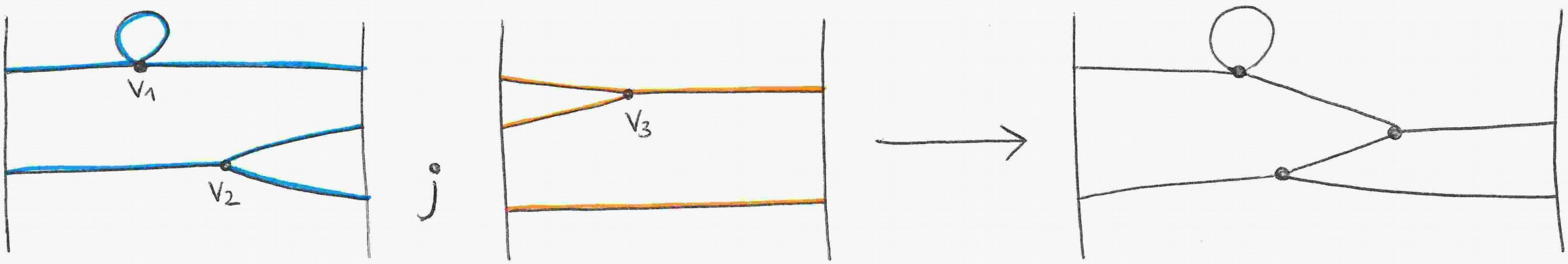
Building Graphs – Sequential Composition (1)



- identify edges at the composition boundary
- update rotation systems on both sides
- new boundary vertex: inputs from the left
  outputs from the right

Building Graphs – Sequential Composition (2)

Example:



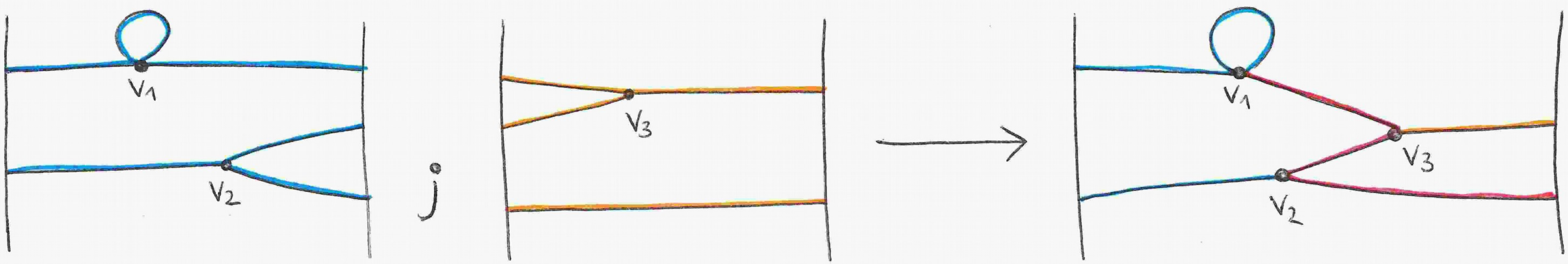
v_1: in, v_1, v_1, out
v_2: in, out, out

in: v_2, v_1
out: v_1, v_2, v_2

v_3: in, in, out
in: out, v_3, v_3
out: v_3, in

Building Graphs - Sequential Composition (2)

Example :



v_1: in, v_1, v_1, out
v_2: in, out, out

in: v_2, v_1
out: v_1, v_2, v_2

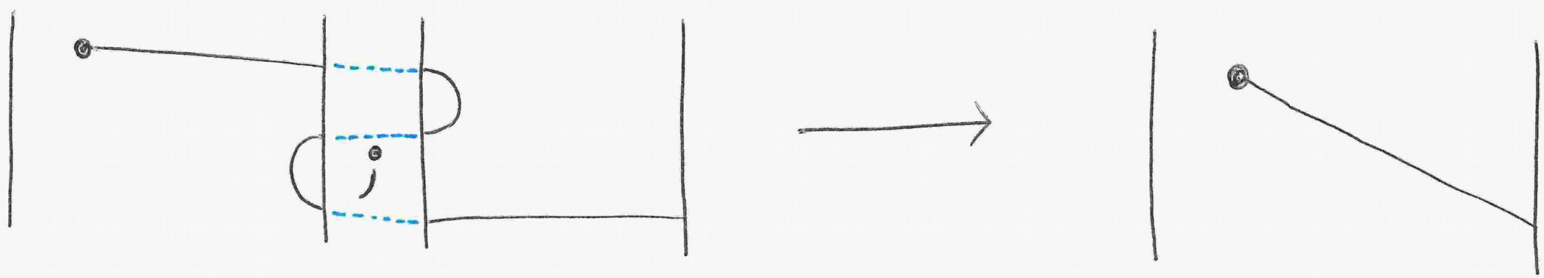
v_3: in, in, out
in: out, v_3, v_3
out: v_3, in

v_1: in, v_1, v_1
v_2: in, v_3, out
v_3: v_2, v_1, out
in: v_2, v_1
out: v_3, v_2

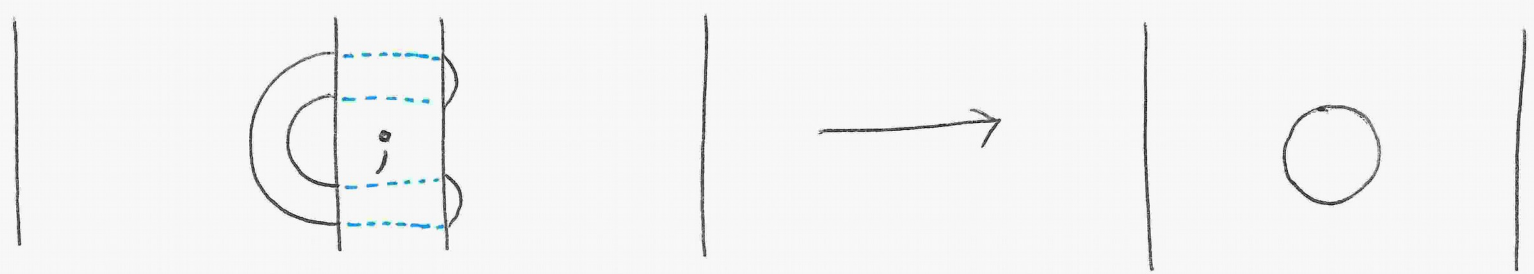
Building Graphs – Sequential Composition (3)

Special cases for sequential composition:

- longer paths:



- cycles:



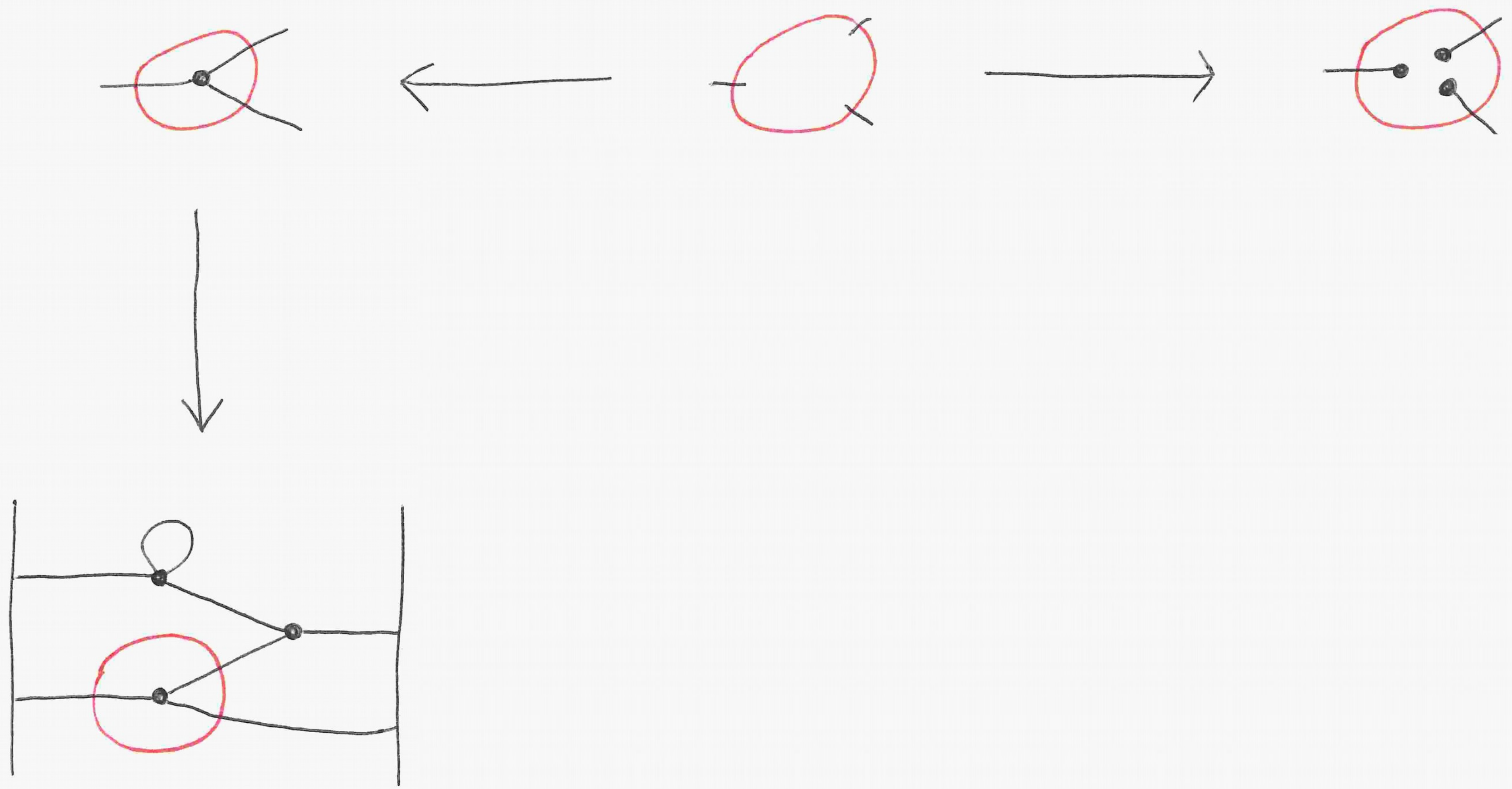
Plane Graphs with a Boundary Vertex

This representation of plane graphs with a boundary vertex defines a strict monoidal category, where

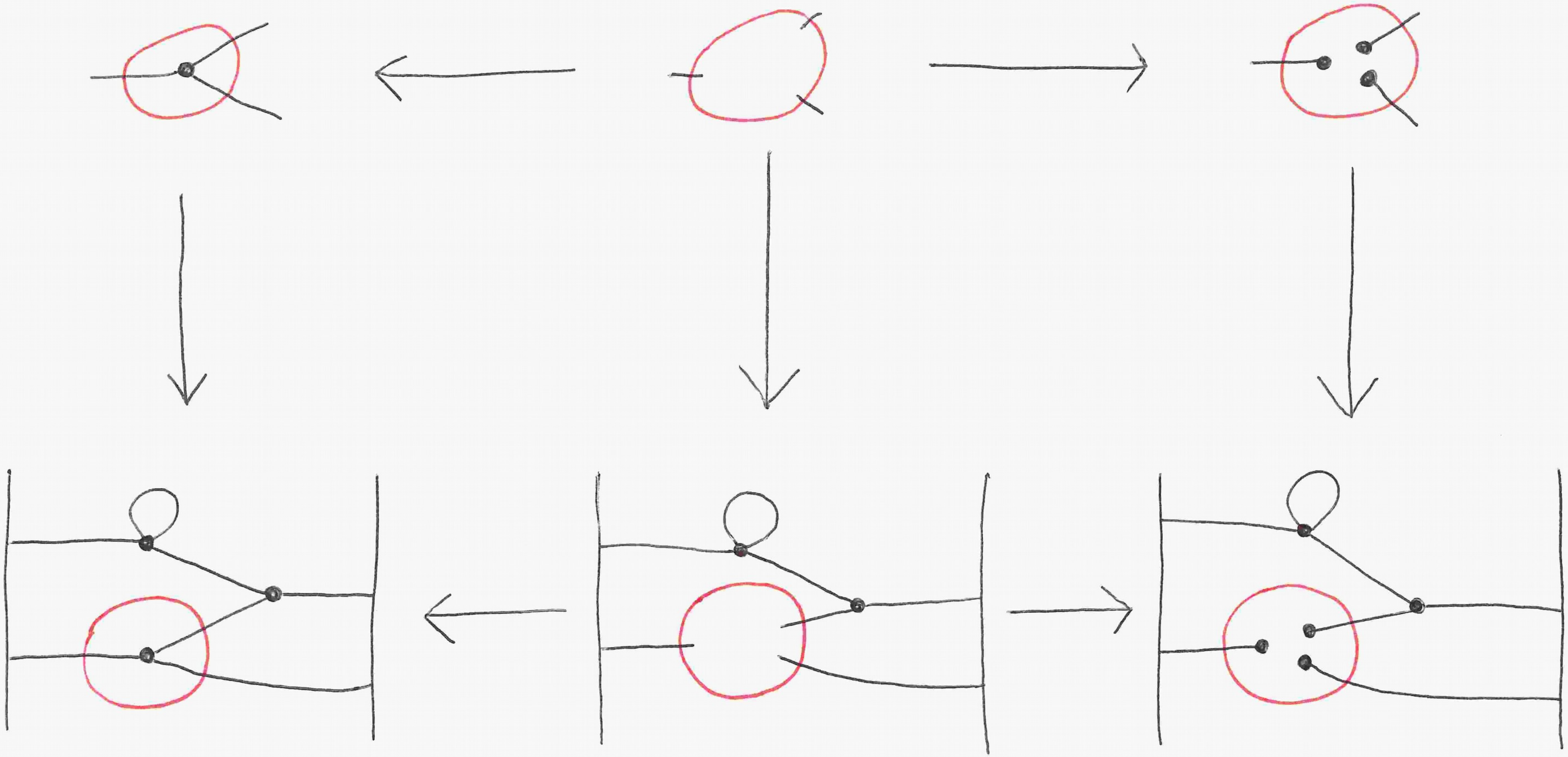
- the objects are lists of types of wires
- the morphisms are graphs
- parallel and sequential composition as defined above

Now: How does rewriting work?

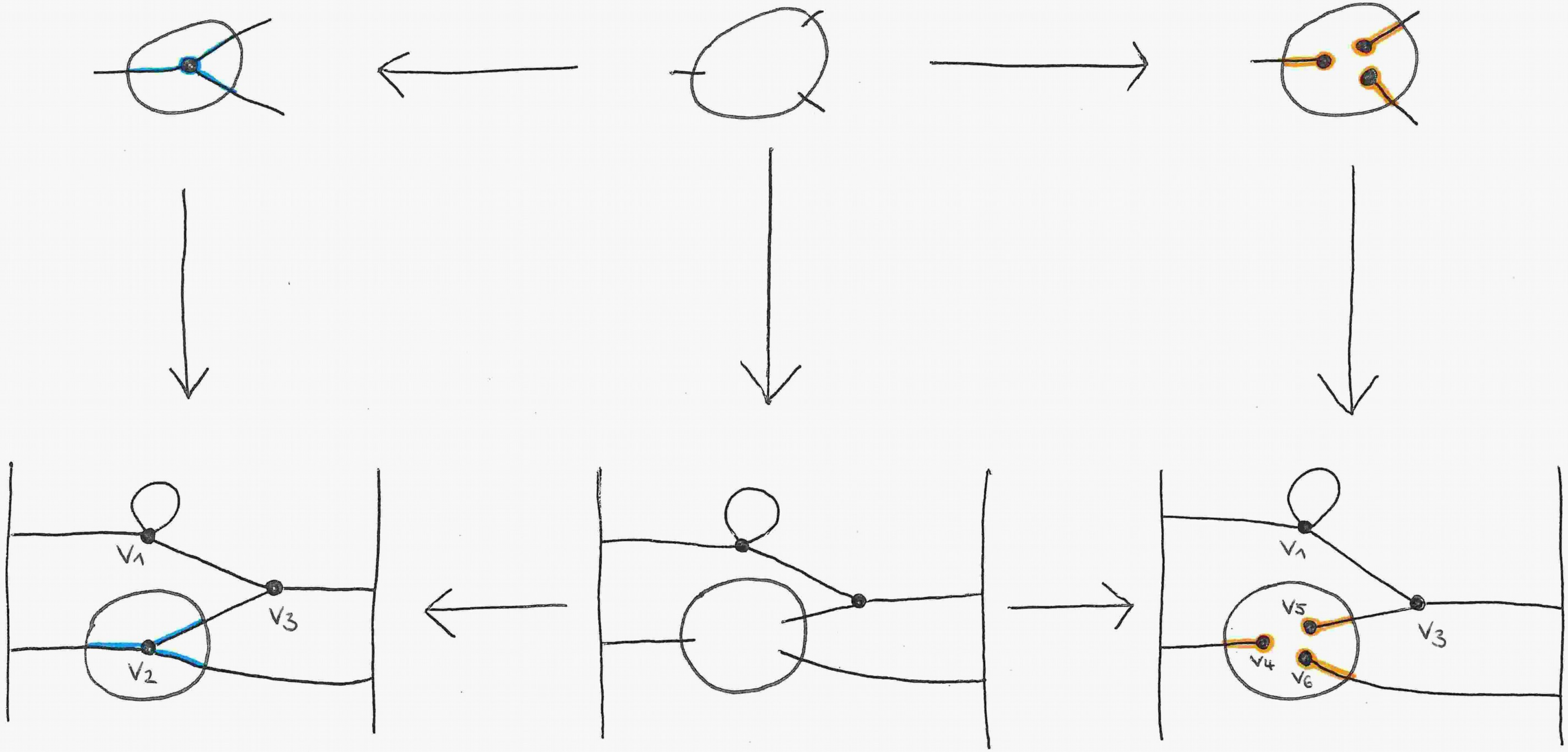
Graph Rewriting - Double Pushout Approach



Graph Rewriting - Double Pushout Approach



Graph Rewriting – Double Pushout Approach



v_1 : in, v_1, v_1, v_3

v_2 : in, v_3, out

v_3 : v_1, out, v_2

in : v_2, v_1

out : v_3, v_2

v_1 : in, v_1, v_1, v_3

v_4 : in ; v_5 : v_3 ; v_6 : out

v_3 : v_1, out, v_5

in : v_4, v_1

out : v_3, v_6

Graph Rewriting

Lemma:

Graph rewriting (as defined above) preserves planarity.

Proof:

- LHS of rewrite rule is a connected graph
  $\Rightarrow$ can be contracted to a single vertex
  (edge contraction preserves planarity)

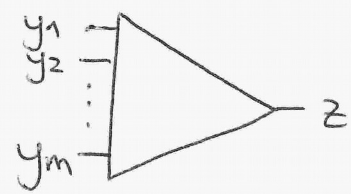
- substitution of a plane graph for a vertex preserves planarity

Operads

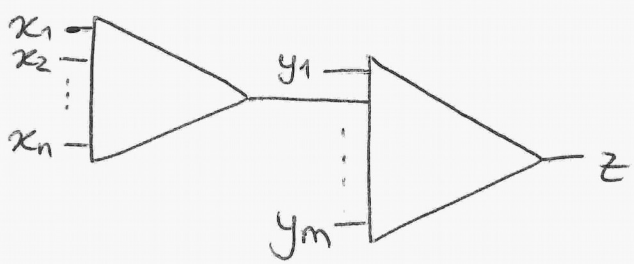
here: coloured operad (= multicategory)

An operad consists of:

- a collection of objects
- a collection of morphisms which take multiple inputs



- composition operation:

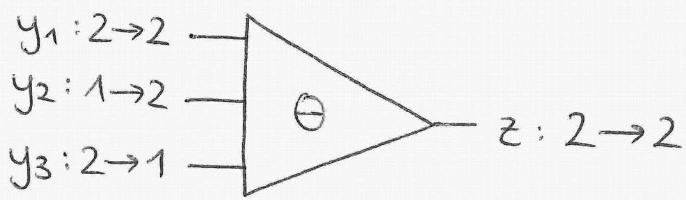


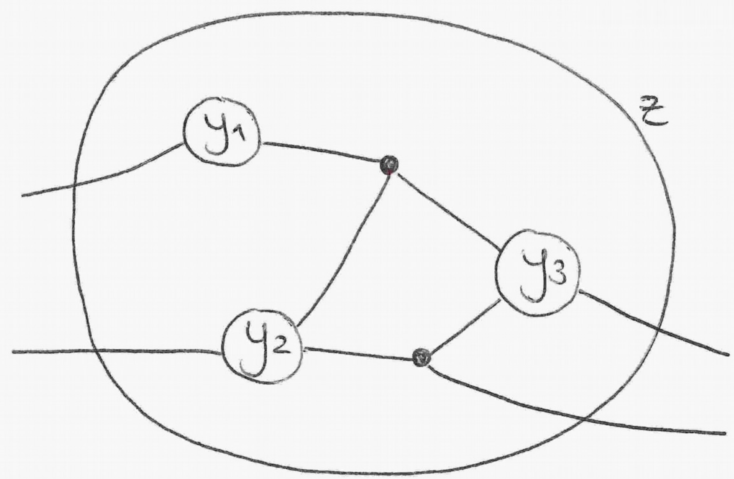
- identity x ——— x

... satisfying the usual identity and associativity laws.

The Operad of Plane Graphs (1)

- objects : connectivity of graph variables
- morphisms : graphs



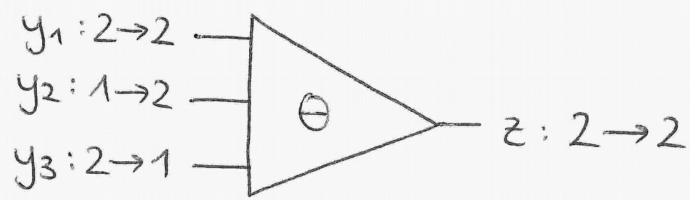
$\Theta =$ 

Similar idea to the operad of wiring diagrams (Spivak, 2013)

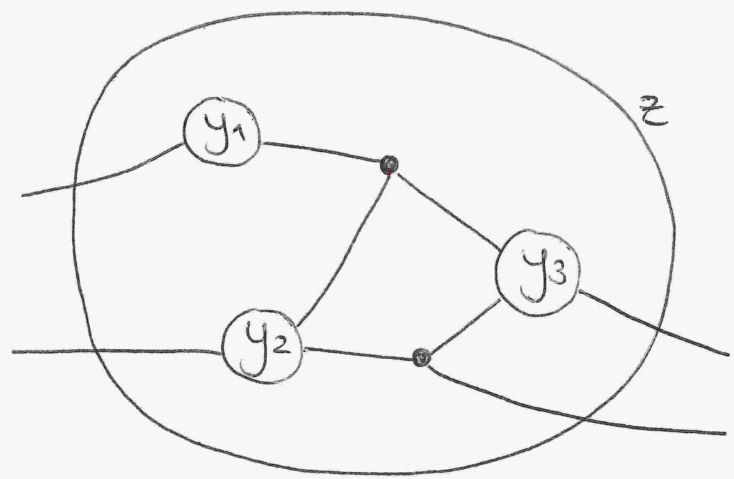
The Operad of Plane Graphs (1)

- objects : connectivity of graph variables
- morphisms : graphs

This is a symmetric operad



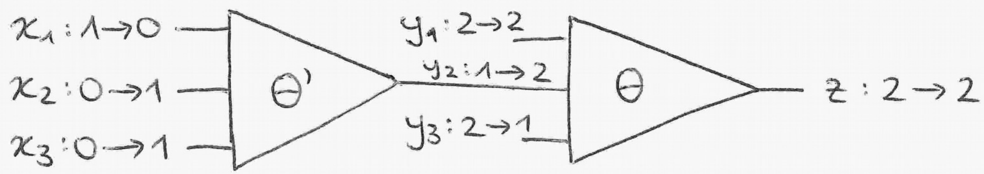
Θ =



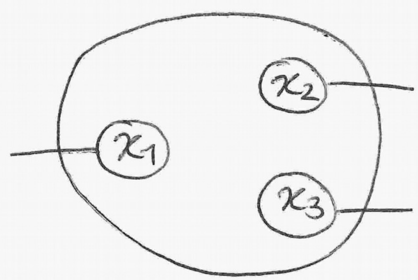
Similar idea to the operad of wiring diagrams (Spivak, 2013)

The Operad of Plane Graphs (2)

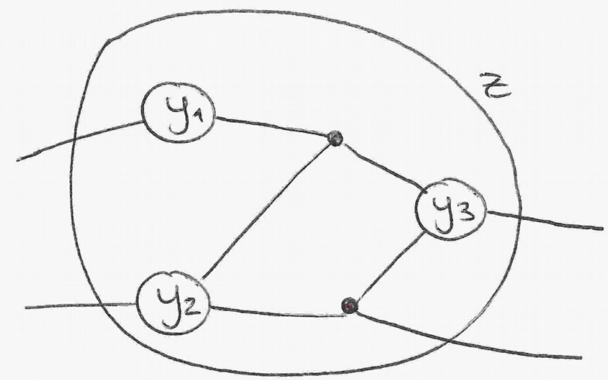
- composition is substitution for a graph variable



$\Theta' =$

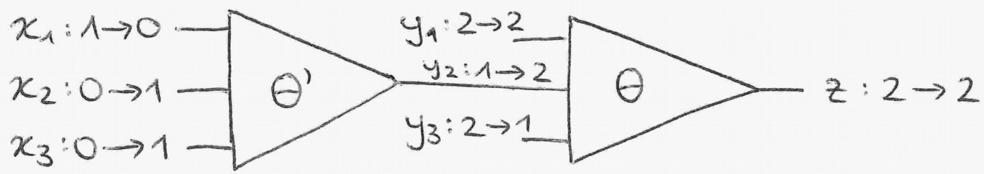


$\Theta =$

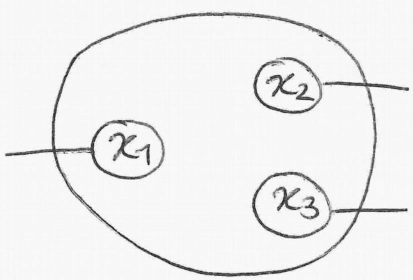


The Operad of Plane Graphs (2)

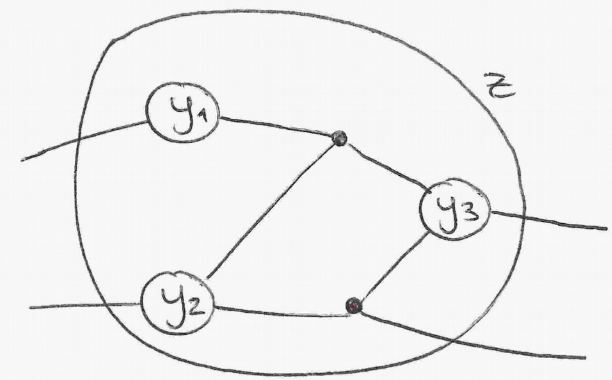
- composition is substitution for a graph variable



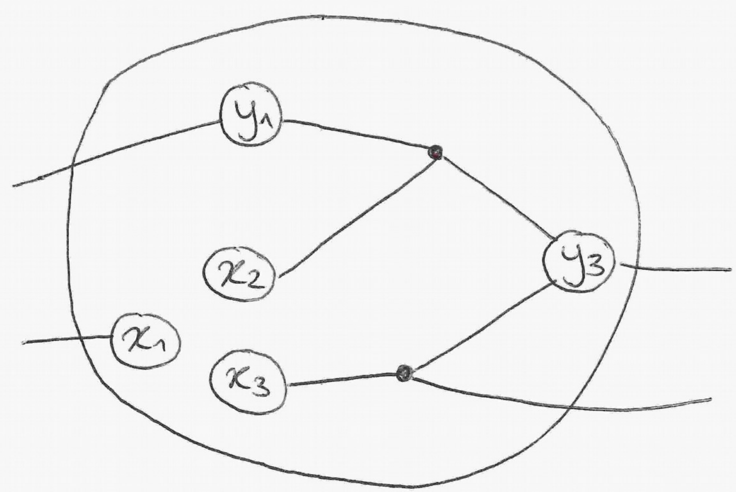
$\Theta' =$



$\Theta =$



$\Theta' ; \Theta =$



Summary

Plane graphs with a boundary vertex form an operad, where the composition operation is substitution.

- representing non-symmetric monoidal categories
- combinatorial presentation via rotation systems

Summary

Plane graphs with a boundary vertex form an operad, where the composition operation is substitution.

- representing non-symmetric monoidal categories
- combinatorial presentation via rotation systems

Future work:

- more complex types of wires
- adding geometry information
- cooperads: substitution becomes patternmatching

Summary

> Plane graphs with a boundary vertex form an operad, where the composition operation is substitution.

- representing non-symmetric monoidal categories
- combinatorial presentation via rotation systems

Future work:

- more complex types of wires
- adding geometry information
- cooperads: substitution becomes patternmatching

Thank you for your attention!

malin.altenmuller@strath.ac.uk github.com/maltenmuller/operads

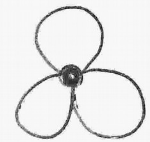
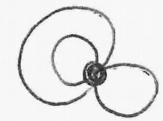
References

- Selinger, P. (2011). A survey of Graphical Languages for Monoidal Categories, pages 289–355. Springer Berlin Heidelberg.

- Spivak, D. (2013). The Operad of Wiring Diagrams: Formalizing a Graphical Language for Databases, Recursion and Plug-and-Play Circuits. CoRR, abs/1305.0297.

- Youngs, J.W.T. (1963). Minimal Imbeddings and the Genus of a Graph. Journal of Mathematics and Mechanics, 12(2): 303–315.

Extra: Self loops

- need to distinguish  and 

rotation systems [v,v,v,v,v,v] [v,v,v,v,v,v]

- introduce pointers to other end of edge [v,v,v,v,v,v] [v,v,v,v,v,v]

(validity check: well formed bracketing of pointers.

[v,v,v,v,v,v] is <u>not plane</u>!)

- works for both inner vertices and the boundary