A Datatype of Planar Graphs

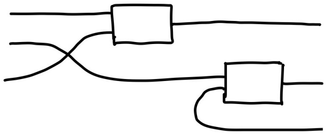
Malin Altenmüller and Conor McBride

University of Strathclyde

TYPES 2022

Why Planar Graphs?

- string diagrams as syntax for monoidal categories



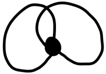
- graphs as the combinatorial objects representing string diagrams
- monoidal categories with specific *topological* properties: no wires cross!

What are Planar Graphs?

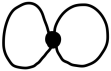
- embedding = drawings of a graphs on some surface
- graph can have multiple embeddings (same or different surface)

What are Planar Graphs?

- embedding = drawings of a graphs on some surface
- graph can have multiple embeddings (same or different surface)
- planar graph: has a plane embedding



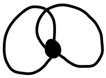
not plane

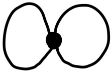


plane

What are Planar Graphs?

- embedding = drawings of a graphs on some surface
- graph can have multiple embeddings (same or different surface)
- planar graph: has a plane embedding



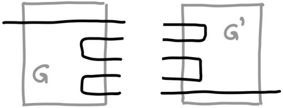


For embeddings, the order of edges around vertices matters!

Graphs as a Datatype?

Two problems:

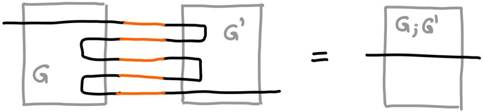
- graphs are cyclic
- composition is really nice on paper, but...

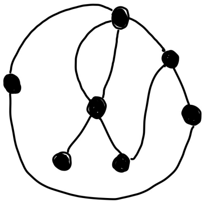


Graphs as a Datatype?

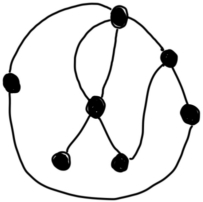
Two problems:

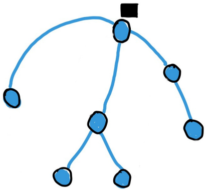
- graphs are cyclic
- composition is really nice on paper, but...



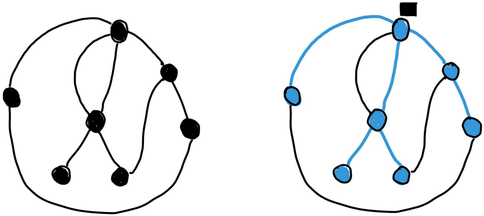


Spanning Trees to the Rescue





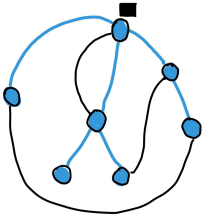
Spanning Trees to the Rescue



Graph = Spanning Tree + Non-Tree Edges

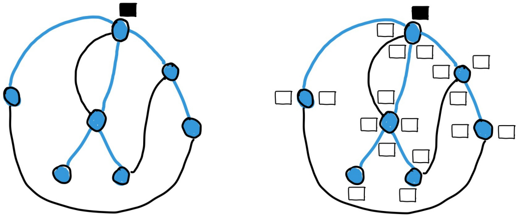
The Datatype(1)

Graph = Spanning Tree + Non-Tree Edges



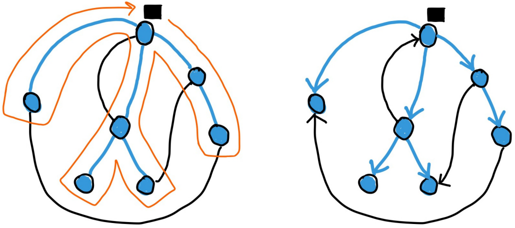
The Datatype(1)

Graph = Spanning Tree + Non-Tree Edges + Sectors



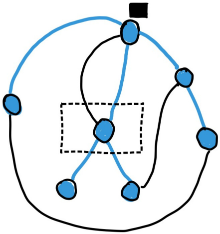
The Datatype(2)

Clockwise traversal of the whole structure:



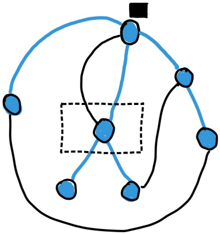
The Datatype(2)

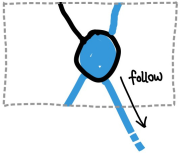
Clockwise traversal of the whole structure:



The Datatype(2)

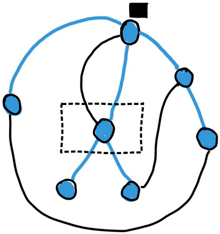
Clockwise traversal of the whole structure:

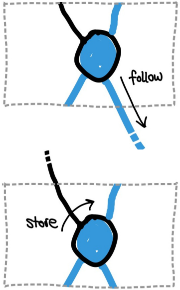




The Datatype(2)

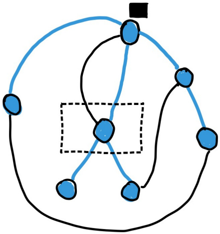
Clockwise traversal of the whole structure:

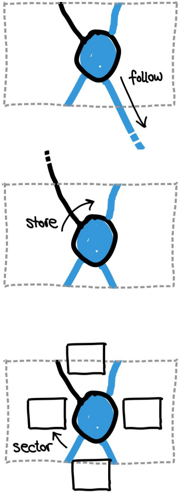




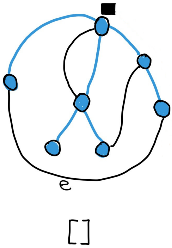
The Datatype(2)

Clockwise traversal of the whole structure:

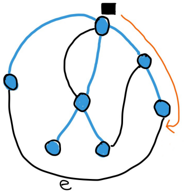




A Stack of Non-Tree Edges

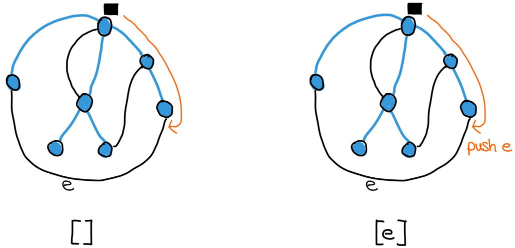


A Stack of Non-Tree Edges

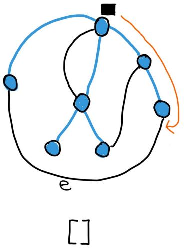


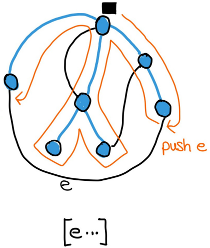
[]

A Stack of Non-Tree Edges

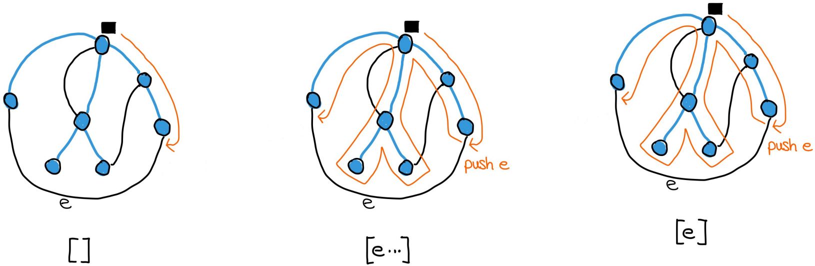


A Stack of Non-Tree Edges

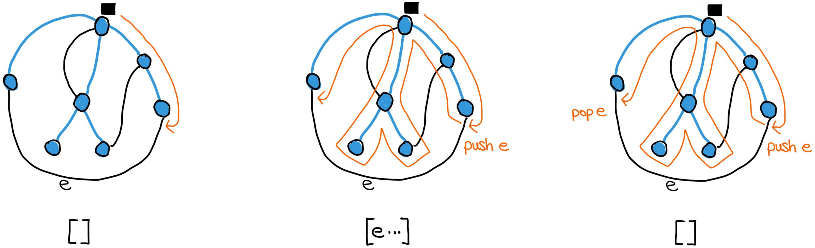




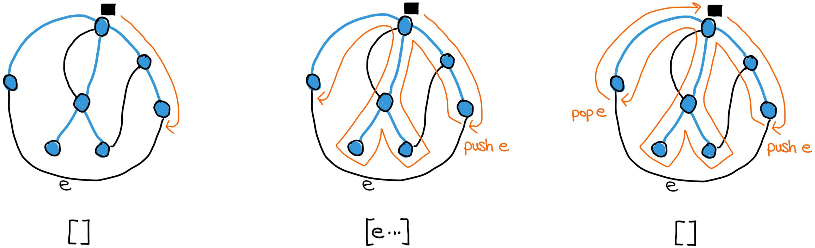
A Stack of Non-Tree Edges



A Stack of Non-Tree Edges



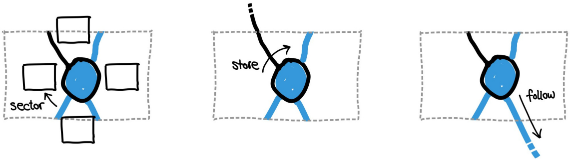
A Stack of Non-Tree Edges



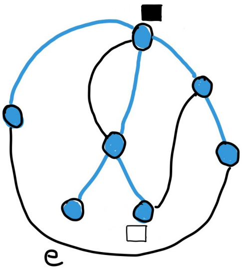
The Datatype(3)

Graph type *indexed* by stack of edges before and after traversal:

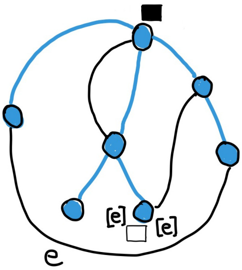
```
data Step : (List E × SE) → (List E × SE) → Set where
  sector : S → Step (es , sec)(es , edg)
  push   : (e : E) → Step (es , edg)(e,-es , sec)
  pop    : (e : E) → Step (e,-es , edg)(es , sec)
  span   : E → V → Star Step (as , sec)(bs , edg)
               →      Step (as , edg)(bs , sec)
```



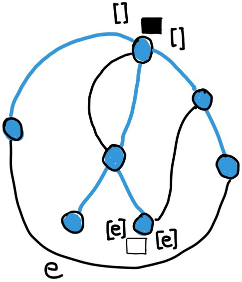
Indexing – Example



Indexing – Example



Indexing – Example



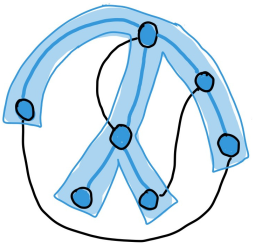
plane graph has index ([] , sec) ([] , sec)

Planarity

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

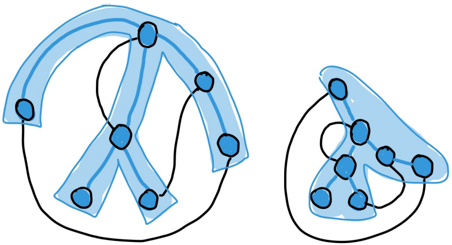
Planarity

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



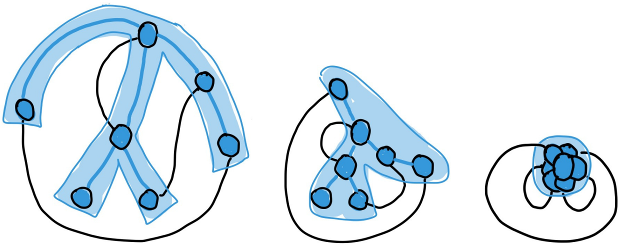
Planarity

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



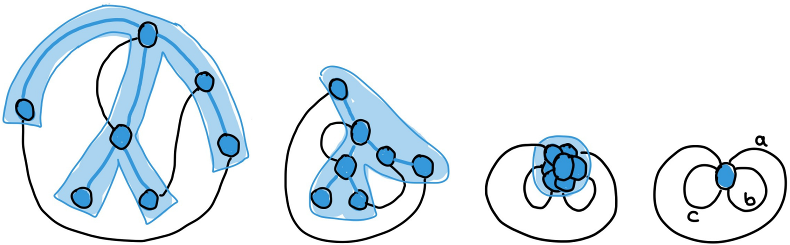
Planarity

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



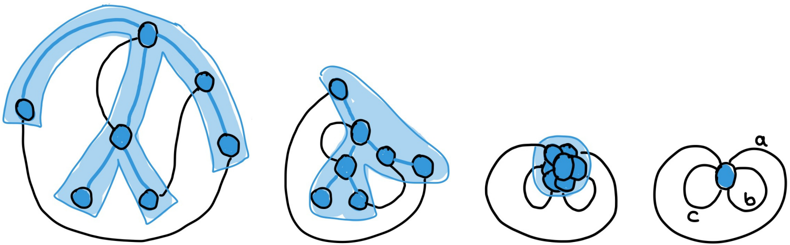
Planarity

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



Planarity

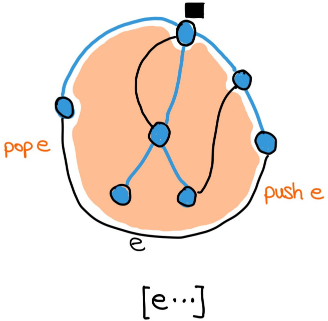
Theorem: A stack of non-tree edges ensures planarity of a graph.



Non-tree edges form a well bracketed word `abbcca`.

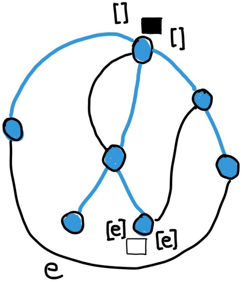
Locality

Non-tree edges delimit regions of the graph:



Zippers[1] for Graphs

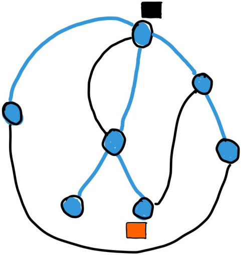
- focus to a sector in the graph



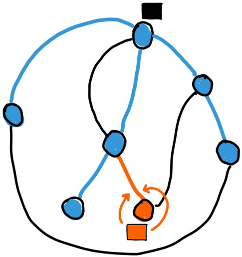
- store a path through the structure to that sector
- at each step: go along one tree edge, remember siblings

[1] Huet, "The Zipper".

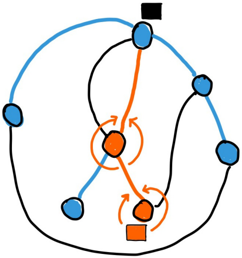
Zipper Example



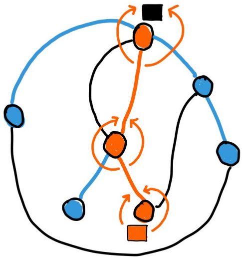
Zipper Example



Zipper Example

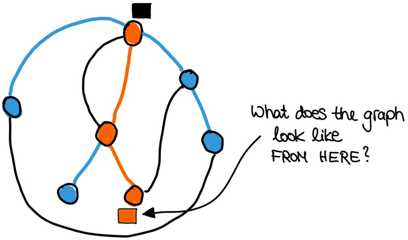


Zipper Example



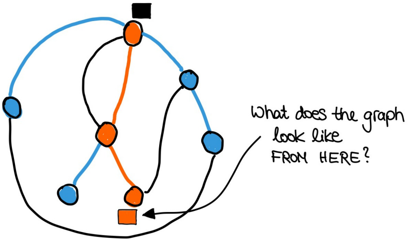
Re-rooting

- start from a zipper
- move the spanning tree's root to the sector in focus



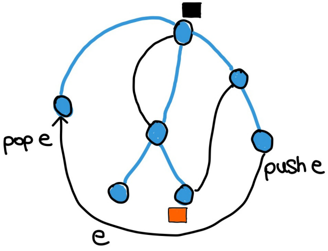
Re-rooting

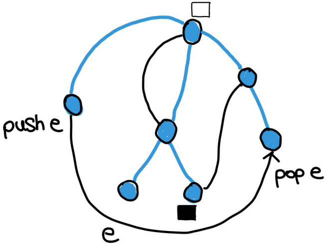
- start from a zipper
- move the spanning tree's root to the sector in focus



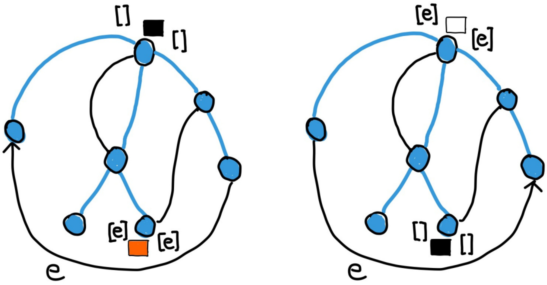
- change the order of traversal of the spanning tree
- swap the stack operation of the sector's stack edges

Turn Non-Tree Edges

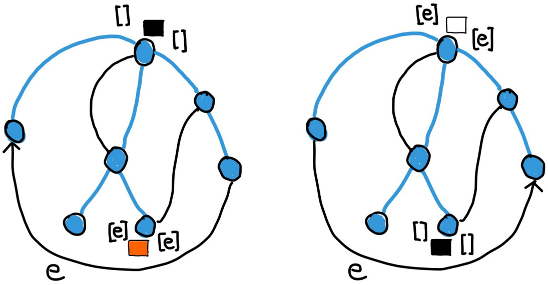




Turn Non-Tree Edges



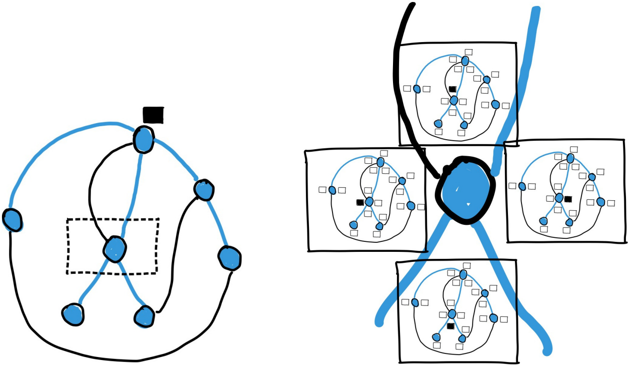
Turn Non-Tree Edges



Theorem: Re-rooting preserves planarity.

Ideas/Future Work(1)

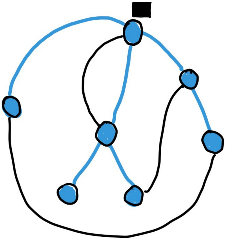
Store data at the sectors: graph re-rooted to here. Get a context comonad[2].



[2] Uustalu and Vene, "Comonadic Notions of Computation".

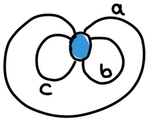
Ideas/Future Work(2)

Overconnected spanning trees represent *rational structures*.



Ideas/Future Work(3)

How about different surfaces from the plane?
Higher genus surfaces?
Non-orientable surfaces?
What to use instead of a stack?



Thank You!

A Datatype of Planar Graphs

Malin Altenmüller and Conor McBride

malin.altenmuller@strath.ac.uk

Huet, Gérard P. "The Zipper". In: *J. Funct. Program.* 7.5 (1997), pp. 549–554. URL: http://journals.cambridge.org/action/displayAbstract?aid=44121.

Uustalu, Tarmo and Varmo Vene. "Comonadic Notions of Computation". In: *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*. Ed. by Jirí Adámek and Clemens Kupke. Vol. 203. Electronic Notes in Theoretical Computer Science 5. Elsevier, 2008, pp. 263–284. DOI: 10.1016/j.entcs.2008.05.029. URL: https://doi.org/10.1016/j.entcs.2008.05.029.