Graph Traversals as Universal Constructions

Categorical Late Lunch

Robin Kaarsgaard* and Siddharth Bhaskar†

March 3, 2021

* School of Informatics, University of Edinburgh
† DIKU, Department of Computer Science, University of Copenhagen

Graph Traversals as Universal Constructions

~~Categorical~~ Graph Theoretical Late Lunch

Robin Kaarsgaard* and Siddharth Bhaskar†

March 3, 2021

* School of Informatics, University of Edinburgh
† DIKU, Department of Computer Science, University of Copenhagen

Graph Traversals as Universal Constructions

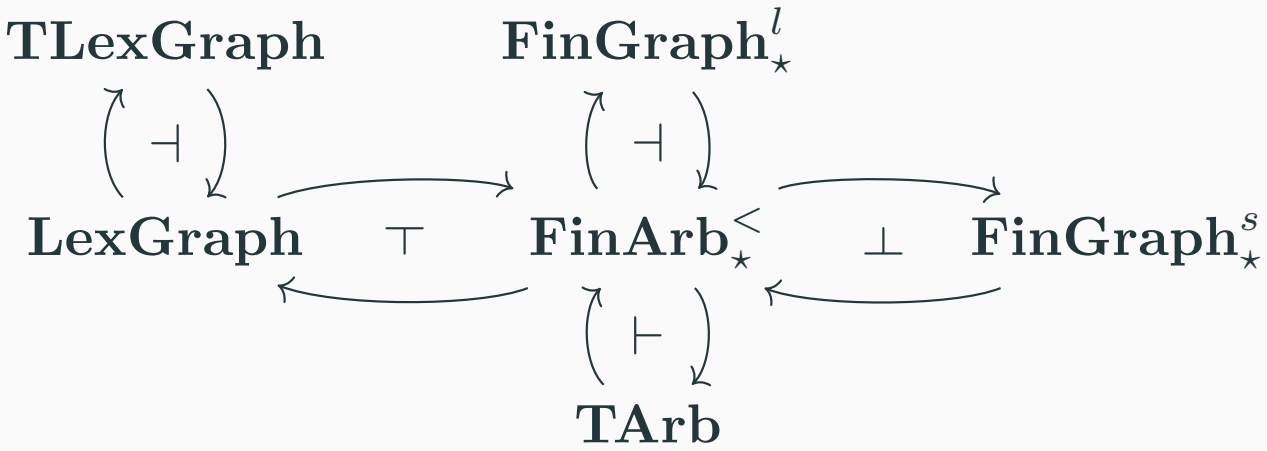
~~Categorical Graph Theoretical~~ Late Lunch

Robin Kaarsgaard* and Siddharth Bhaskar†

March 3, 2021

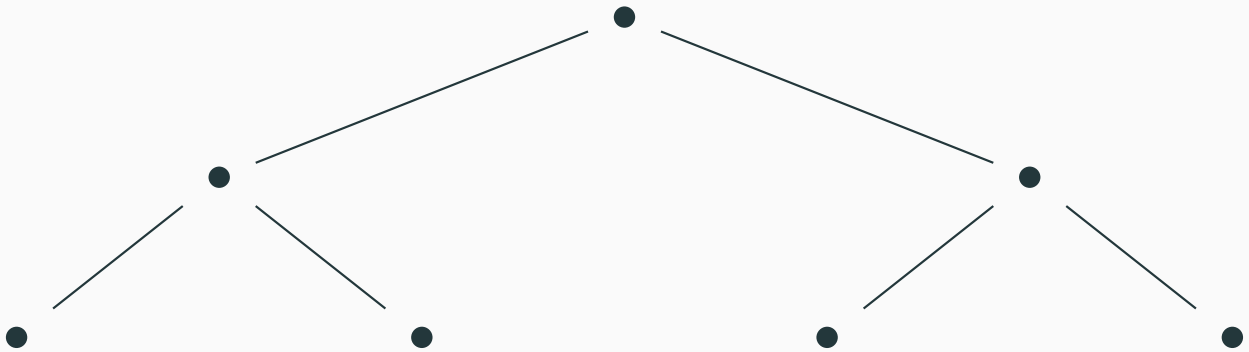
* School of Informatics, University of Edinburgh
† DIKU, Department of Computer Science, University of Copenhagen

An adventure in applied category theory



Graph traversals?

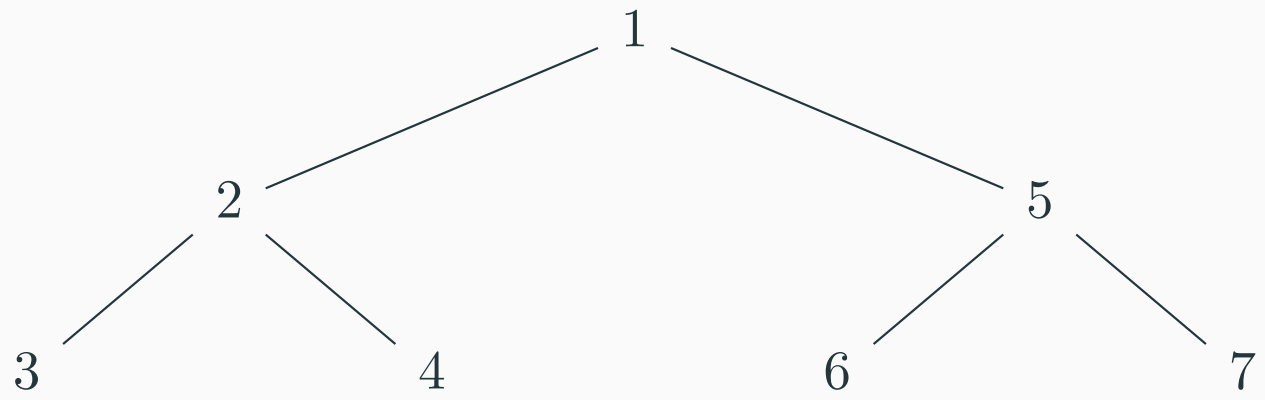
A graph traversal is a way of visiting the vertices in a graph in a way that respects graph structure.



Every graph search or pathfinding algorithm implements some form of traversal. It is also used in, e.g., proof search in logic and logic programming, where the choice of traversal is often crucial to the success of the algorithm.

Graph traversals?

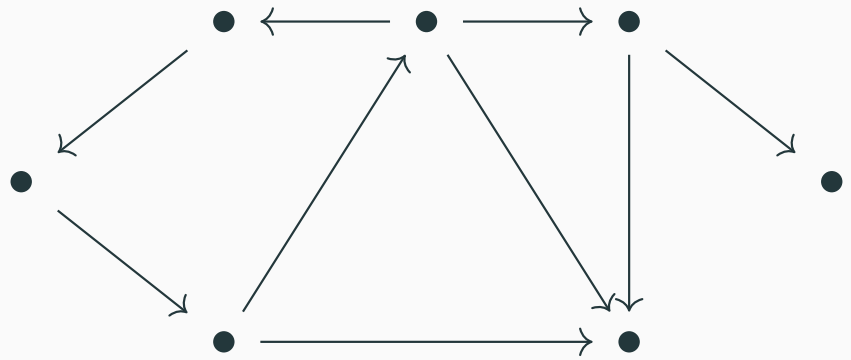
A graph traversal is a way of visiting the vertices in a graph in a way that respects graph structure.



Every graph search or pathfinding algorithm implements some form of traversal. It is also used in, e.g., proof search in logic and logic programming, where the choice of traversal is often crucial to the success of the algorithm.

Graphs

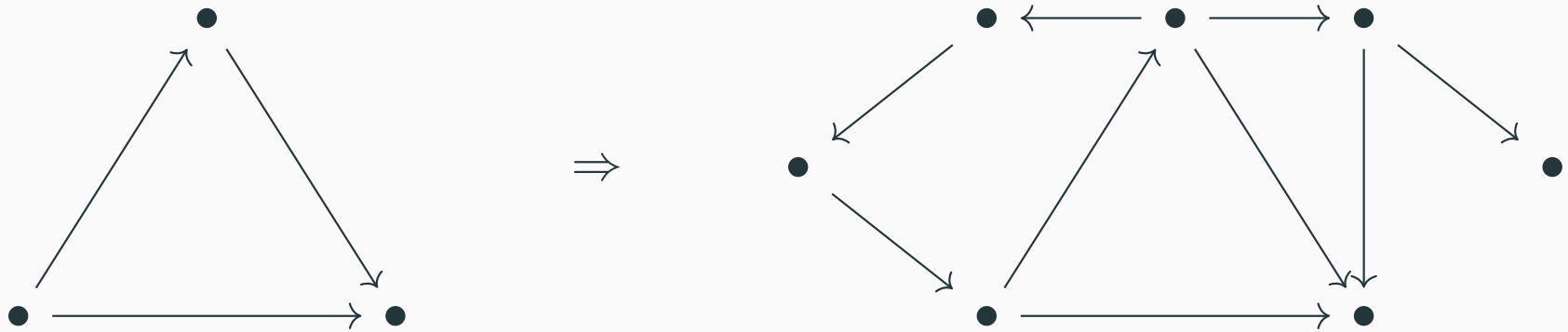
Finite directed graphs: $(G, \to)$.



A *path* in a graph is a finite sequence of vertices $v_1, v_2, \ldots, v_n$ such that $v_i \to v_{i+1}$ for all natural $1 \leq i < n$. Write $v_1 \rightsquigarrow v_n$ for (the existence of) a path from v_1 to v_n.

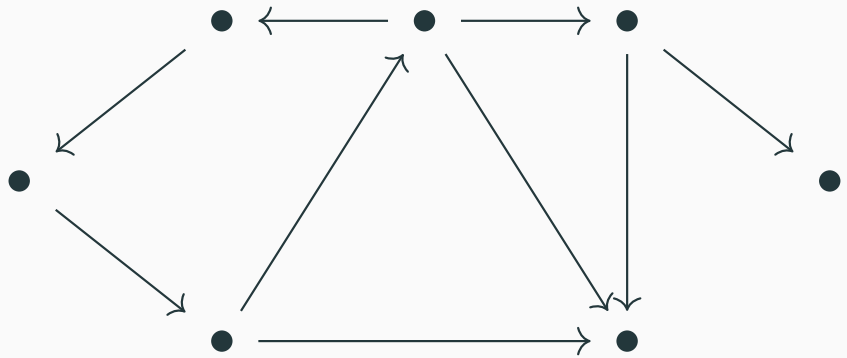
The *neighbourhood* of a vertex u in G is $N(u) = \{v \in G \mid u \to v\}$.

Graph homomorphisms



A *graph homomorphism* $G \xrightarrow{h} H$ is a function $G \to H$ satisfying $u \to v$ implies $h(u) \to h(v)$ for all $u, v \in G$.

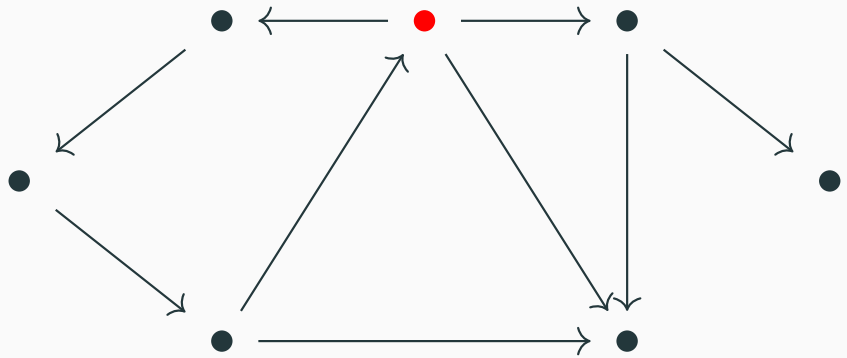
Traversals



A *traversal* is a linear order of vertices $<$ satisfying that for all $v \in G$, v is least or there exists $u \in G$ such that $u \to v$ and $u < v$.

In other words, a traversal is a vertex order that respects the graph structure.

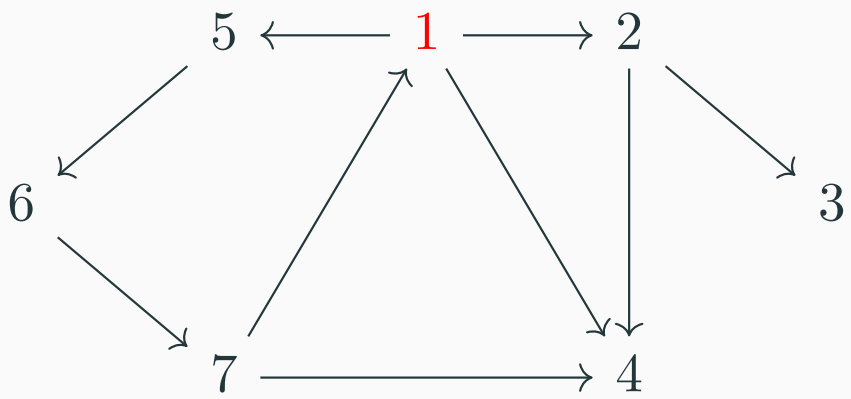
Traversals



A *traversal* is a linear order of vertices $<$ satisfying that for all $v \in G$, v is least or there exists $u \in G$ such that $u \to v$ and $u < v$.

In other words, a traversal is a vertex order that respects the graph structure.

Traversals



A *traversal* is a linear order of vertices $<$ satisfying that for all $v \in G$, v is least or there exists $u \in G$ such that $u \to v$ and $u < v$.

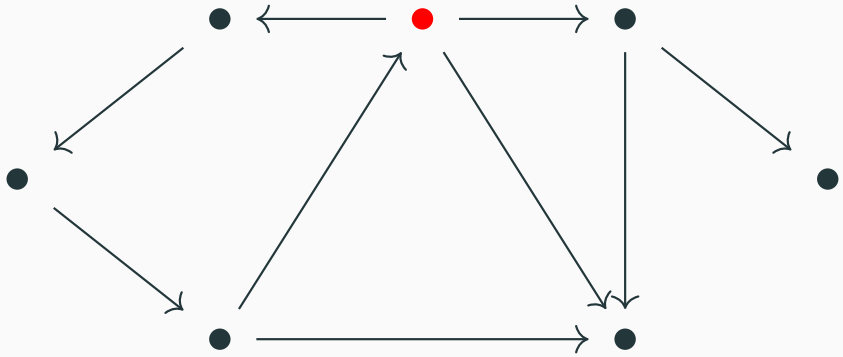
In other words, a traversal is a vertex order that respects the graph structure.

Depth-first traversal

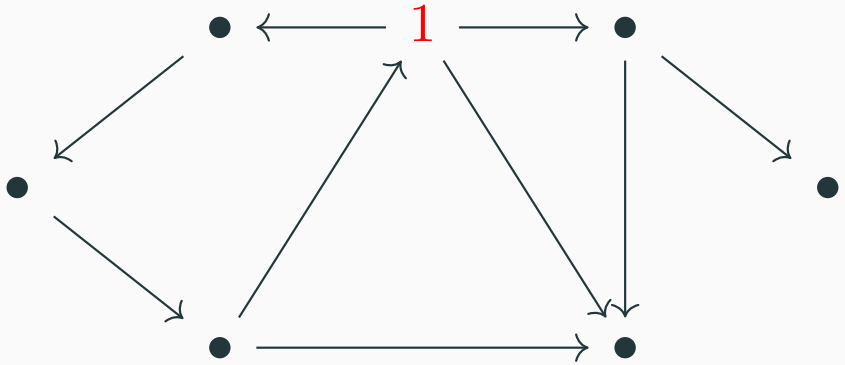
Given a linear order on the vertices of a graph, we can perform the *depth-first traversal* starting from a vertex v as follows:

1. Initialise an empty list A for visited vertices.
2. Add v to A.
3. Repeat:
   - 3.1 Compute $v_0 = \min(N(v) \setminus A)$.
     - 3.1.1 If no unvisited neighbour is found, backtrack and go back to 3.
     - 3.1.2 If no backtracking is possible, terminate and return A.
   - 3.2 Add v_0 to the end of A.
   - 3.3 Move to v_0.

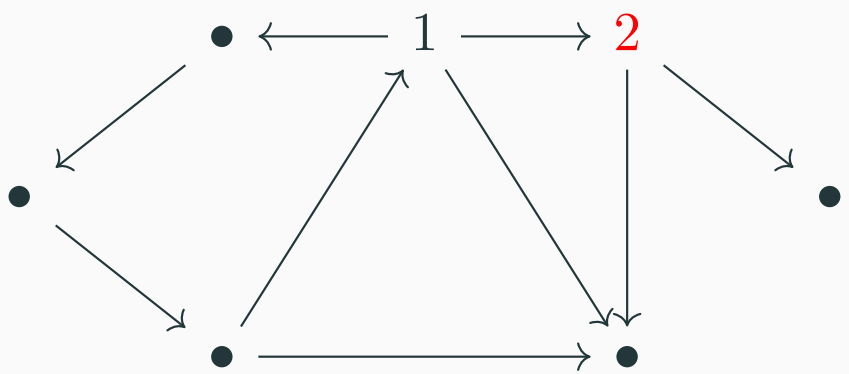
Depth-first traversal: Example



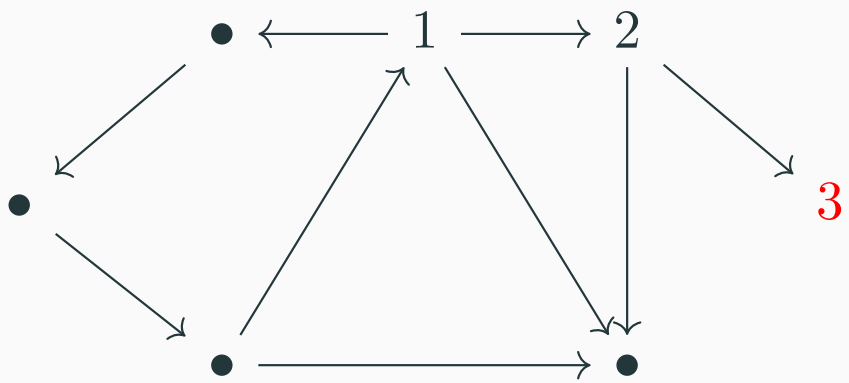
Depth-first traversal: Example



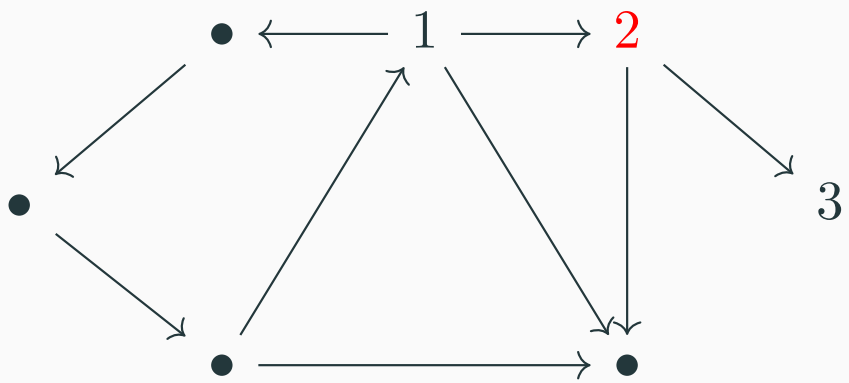
Depth-first traversal: Example



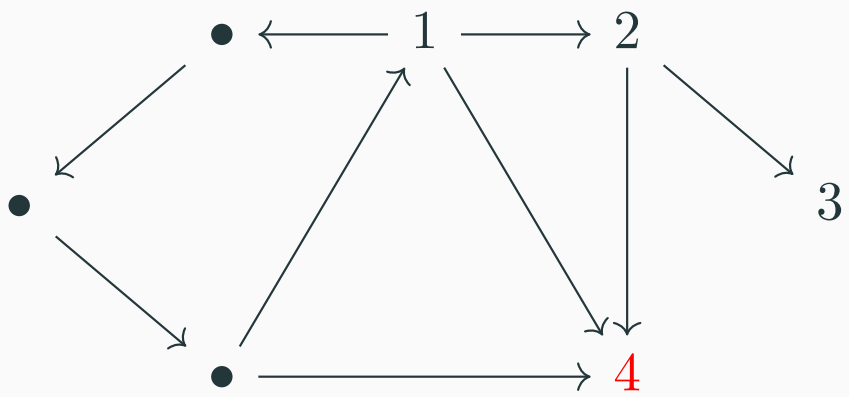
Depth-first traversal: Example



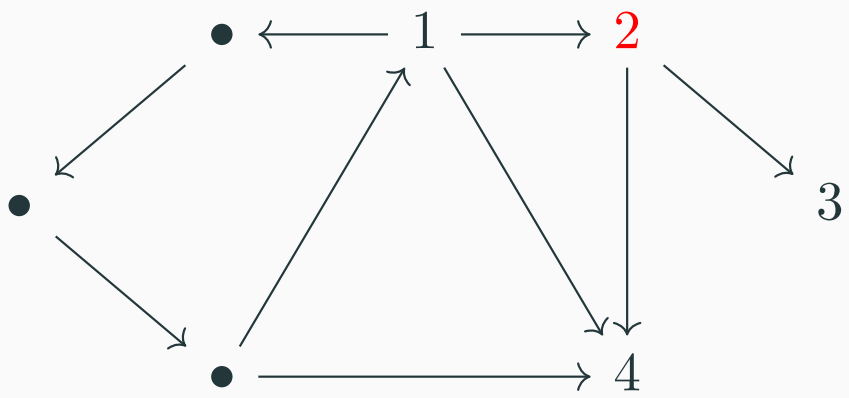
Depth-first traversal: Example



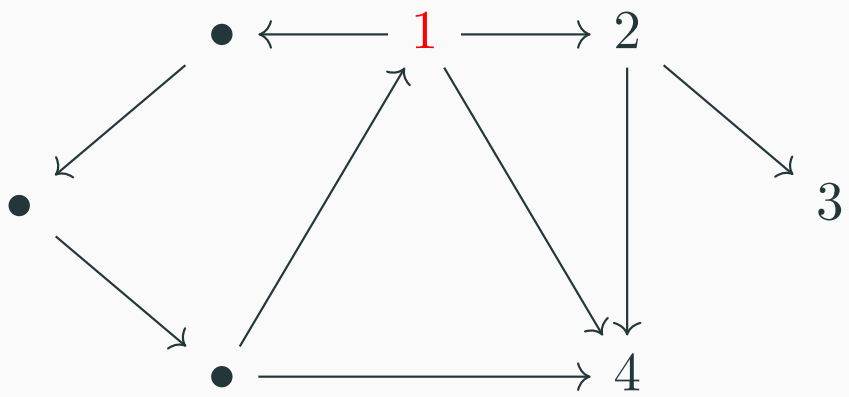
Depth-first traversal: Example



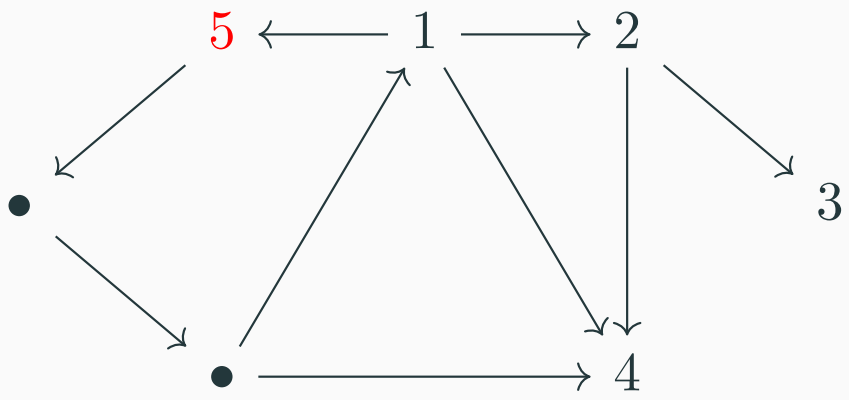
Depth-first traversal: Example



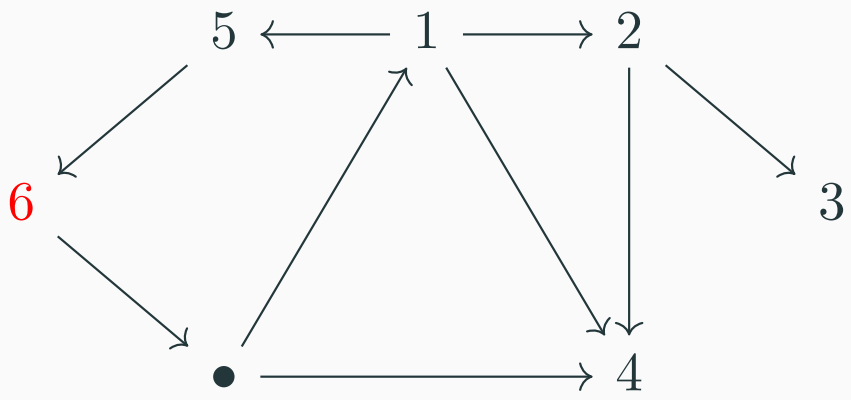
Depth-first traversal: Example



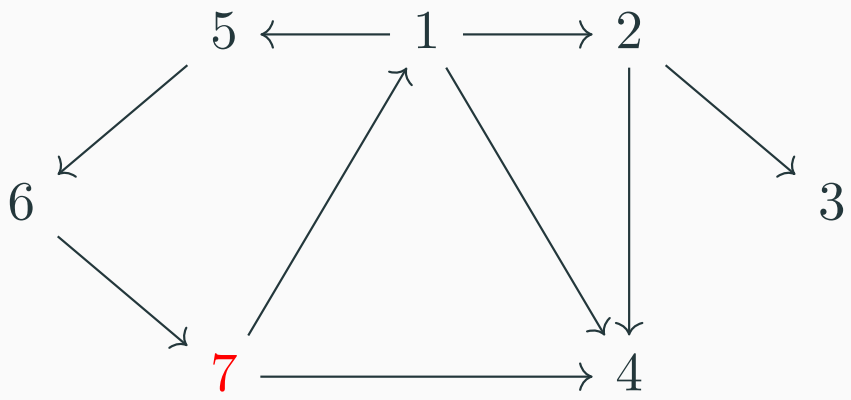
Depth-first traversal: Example



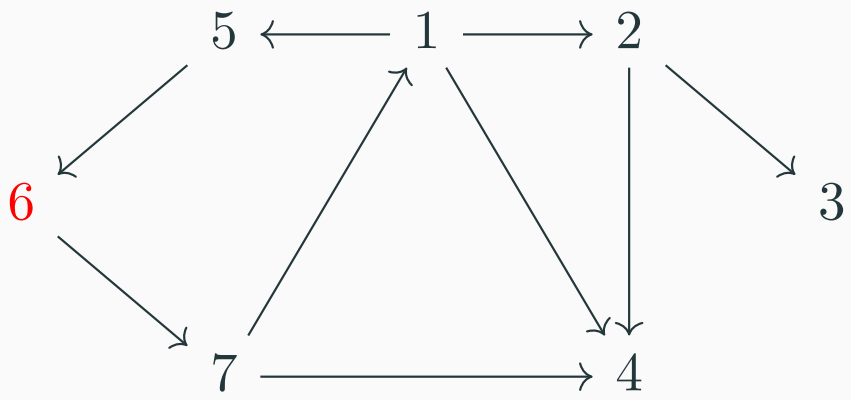
Depth-first traversal: Example



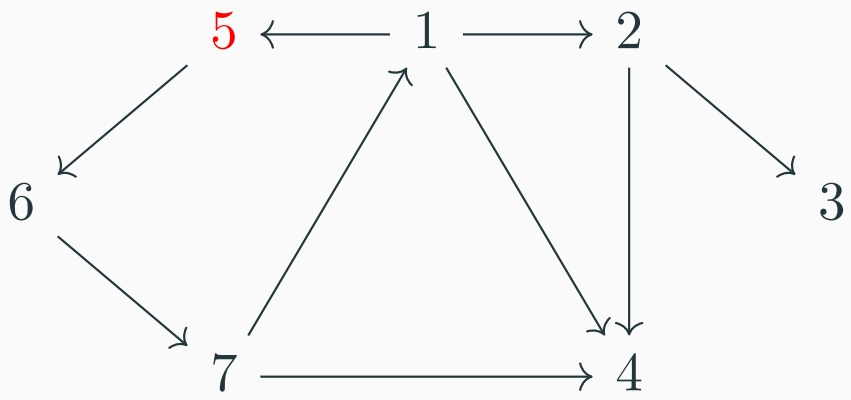
Depth-first traversal: Example



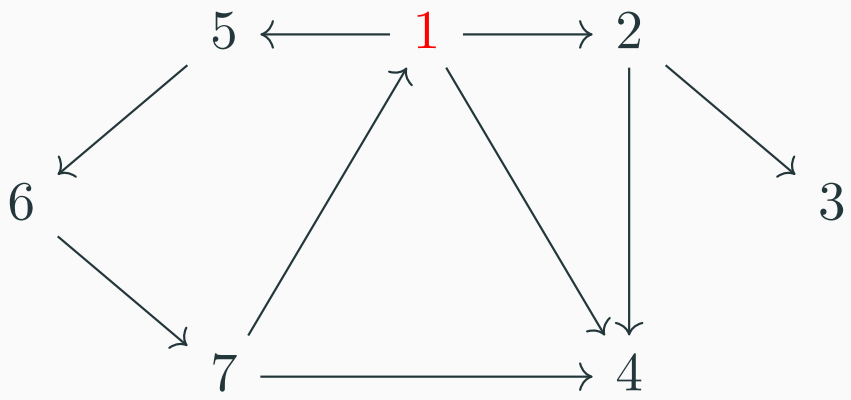
Depth-first traversal: Example



Depth-first traversal: Example



Depth-first traversal: Example



Categorifying traversals

A motivation for studying depth-first traversals categorically came from the following observations:

1. Any (ordered) graph (with a distinguished vertex) has a depth-first traversal.
2. Doing the depth-first traversal algorithm twice is the same as doing it once.

Categorifying traversals

A motivation for studying depth-first traversals categorically came from the following observations:

1. Any (ordered) graph (with a distinguished vertex) has a depth-first traversal.
2. Doing the depth-first traversal algorithm twice is the same as doing it once.

What does that have to do with category theory? Well...

Categorifying traversals

A motivation for studying depth-first traversals categorically came from the following observations:

1. Any (ordered) graph (with a distinguished vertex) has a depth-first traversal.
2. Doing the depth-first traversal algorithm twice is the same as doing it once.

What does that have to do with category theory? Well...

1. $G \to T(G)$.
2. $T(T(G)) \to T(G)$.

That looks like an idempotent monad!

There's just one tiny problem…

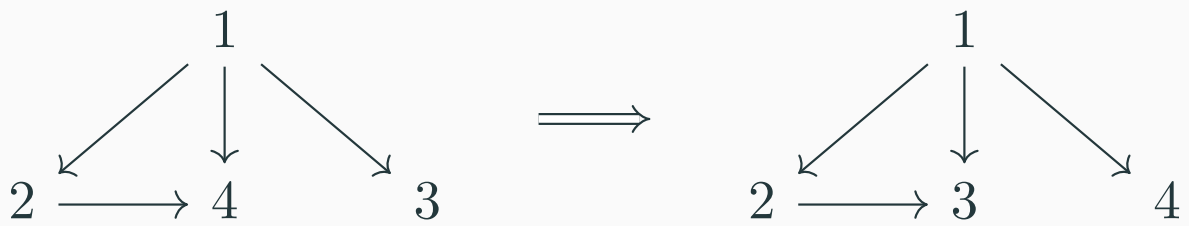
That looks like an idempotent monad!

There's just one tiny problem…It doesn't work.

That looks like an idempotent monad!

There's just one tiny problem…It doesn't work.

Any reasonable notion of graph homomorphism of vertex-ordered graphs must be monotone on vertices. But the homomorphism $G \to T(G)$ needn't be:

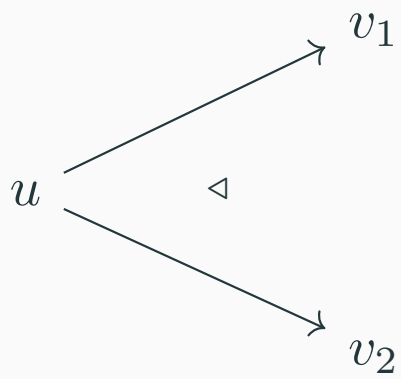


But even if it did, it doesn't answer the question of whether this arises from any interesting adjunction.

If at first you don't succeed...

Insight: Ordering *edges*, specifically edges with the same source, gives better results.

Instead of requiring that vertices be linearly ordered, require that all edges with the same source are linearly ordered:

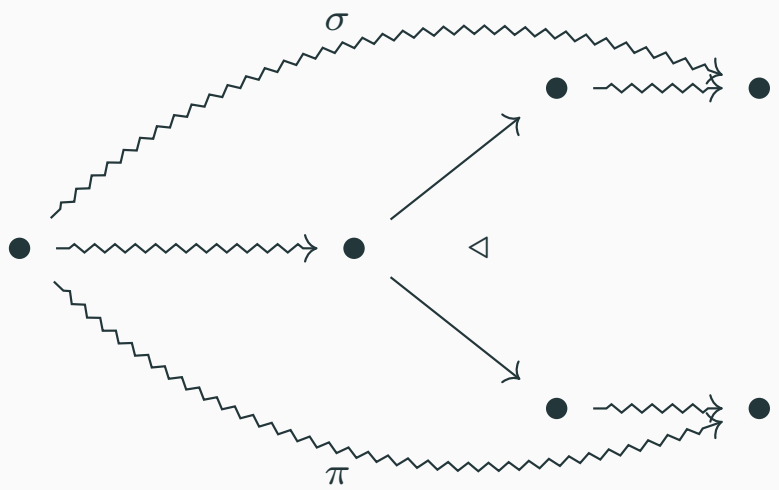


We call this an *edge-ordered graph*. The natural notion of graph homomorphism is one that preserves this local order: $u \to v_1 \triangleleft u \to v_2$ implies $h(u) \to h(v_1) \triangleleft h(u) \to h(v_2)$.

The lexicographic path order

In an edge-ordered graph, define an order $\prec$ on paths as follows:

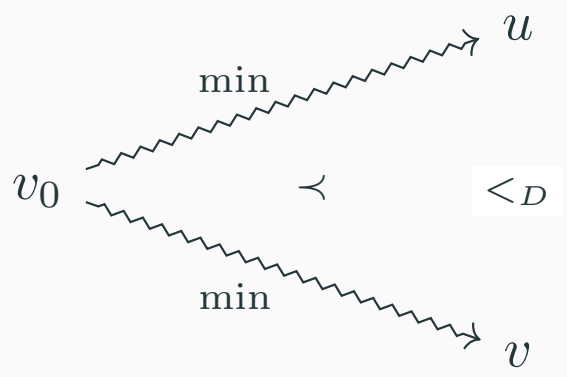
- if σ is a prefix of π then $\sigma \prec \pi$,
- otherwise, consider the first vertex at which they differ: if



then $\sigma \prec \pi$ (otherwise $\pi \prec \sigma$). It can be shown that this defines a linear order on paths* with the same source.

The lexicographic path order

In an edge-ordered graph with distinguished vertex v_0, this allows us to linearly order vertices by $u <_D v$ iff $\min(v_0 \rightsquigarrow u) \prec \min(v_0 \rightsquigarrow v)$.



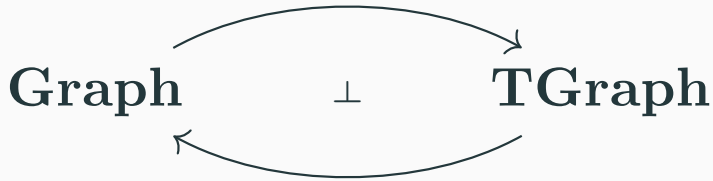
Theorem: $\cdot <_D \cdot$ is precisely the depth-first traversal of G.

To recap, we can obtain the depth-first traversal of G by lexicographically ordering paths out of v_0, and then comparing vertices u and v by comparing the lexicographically least paths $v_0 \rightsquigarrow u$ and $v_0 \rightsquigarrow v$.

If only there was some sort of way of constructing a graph where edges correspond to the existence of paths in another graph...

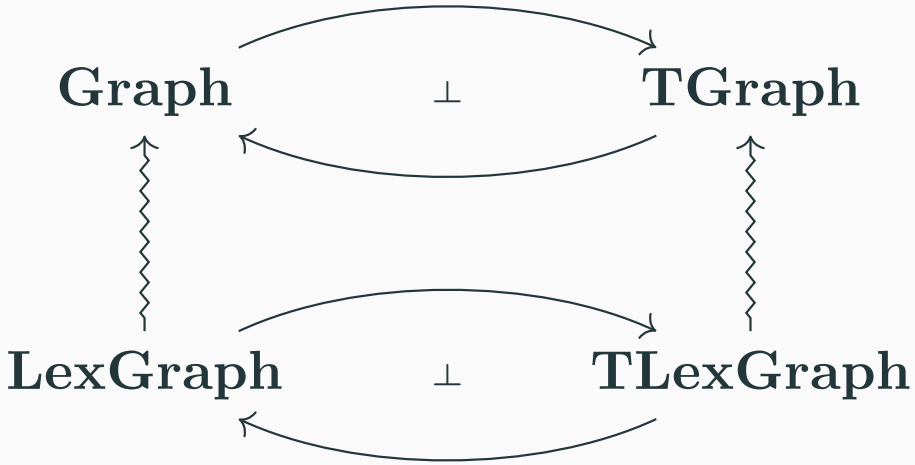
The transitive closure

There is: the transitive closure! It even has a universal construction via an adjunction.

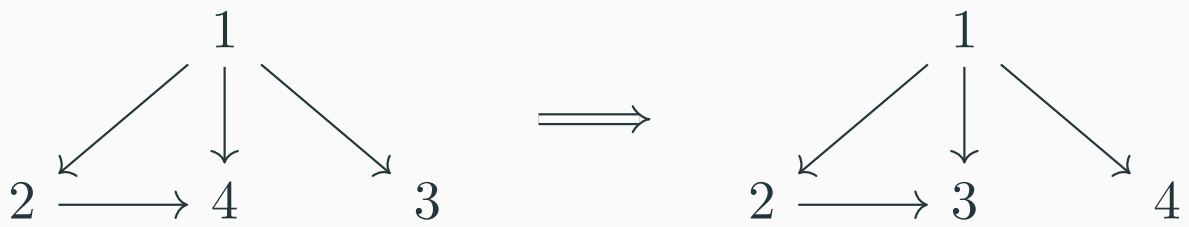


The transitive closure

There is: the transitive closure! It even has a universal construction via an adjunction. Let's extend that.

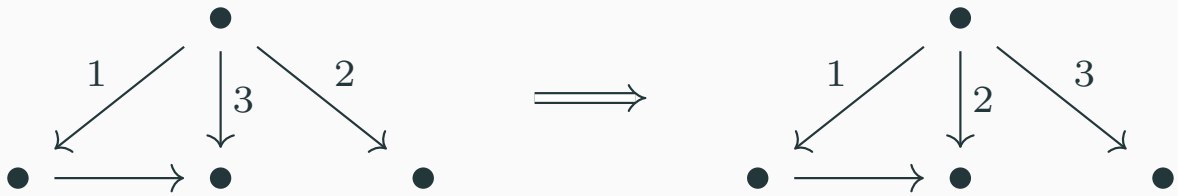


Remember this problem?



Lex-graphs

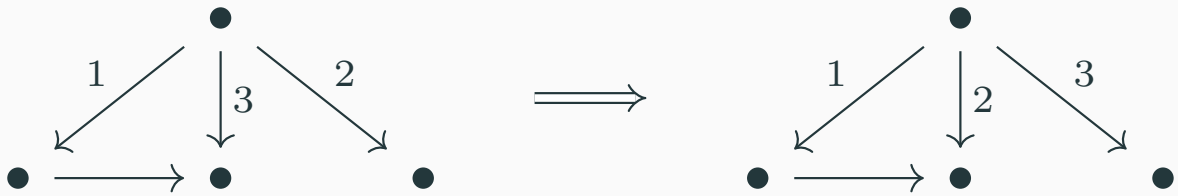
It's actually still a problem.



But now we can fix it!

Lex-graphs

It's actually still a problem.



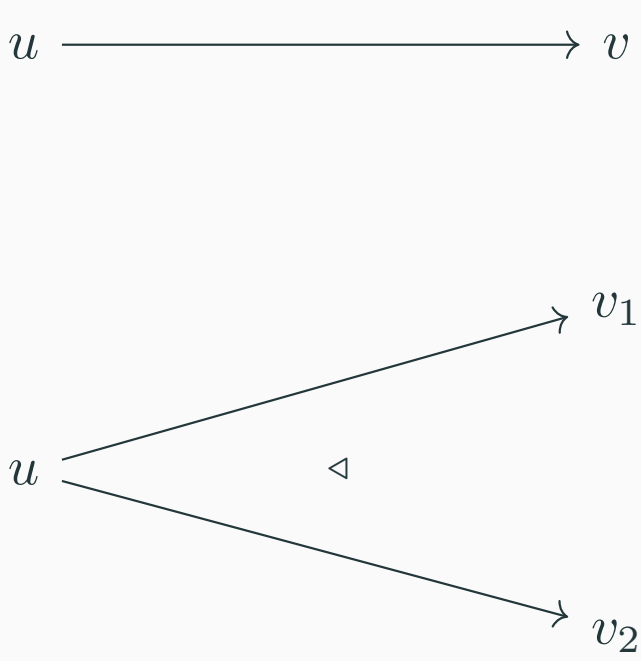
But now we can fix it!

A *lex-graph* is an edge-ordered graph with a distinguished vertex satisfying that the edge-order is compatible with the lexicographic order on lexicographically least paths: $u \to v_1 \lhd u \to v_2$ iff $\min(u \rightsquigarrow v_1) \prec \min(u \rightsquigarrow v_2)$.

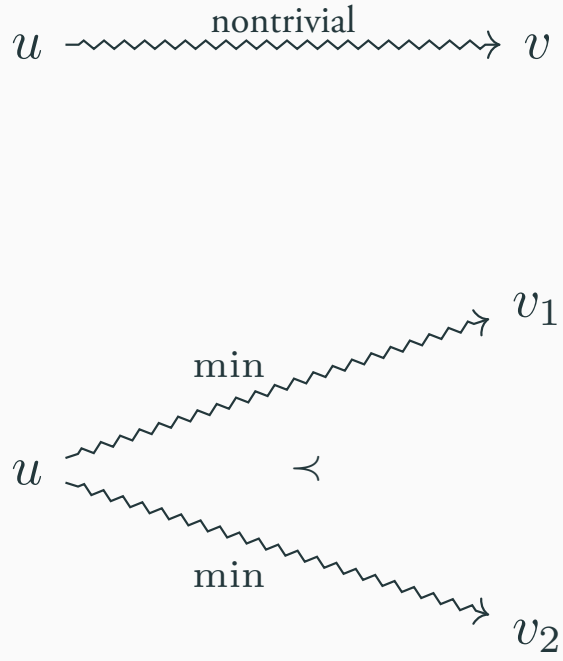
Note that this precludes the graph on the left!

The lexicographic-transitive closure

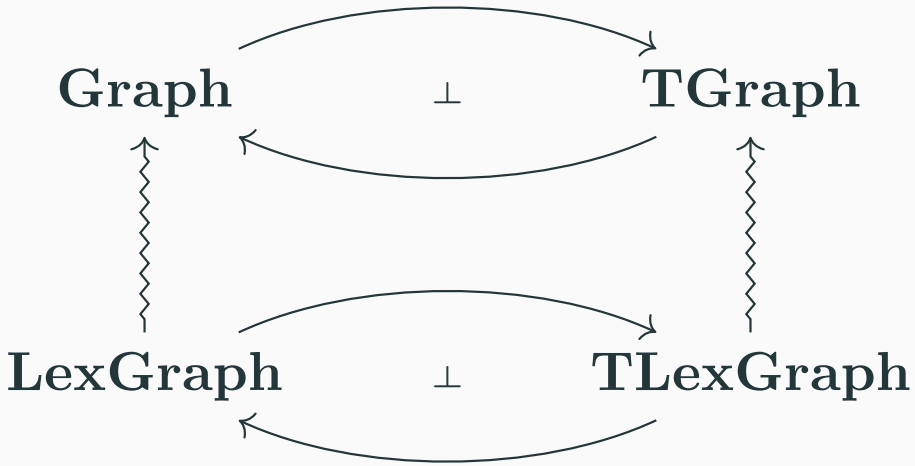
In $T(G)$:



In G:

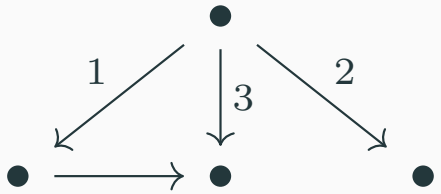


The lexicographic-transitive closure



Completing the picture

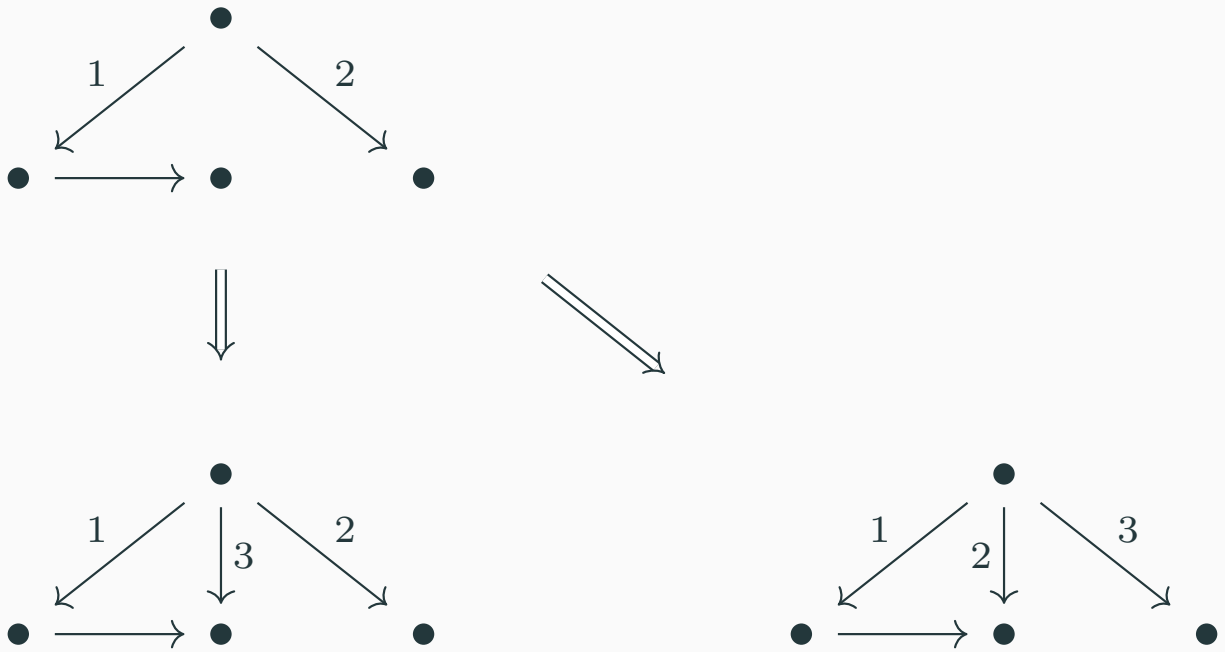
But what if we actually want the depth-first traversal of an arbitrary edge-ordered graph?



Observation: For the purposes of depth-first traversal, we can safely remove all edges that do not lie on a lexicographically least path.

This turns out to yield the *cofree (finite, edge-ordered, pointed) arborescence* of a (finite, edge-ordered, pointed) graph, *which are always lex-graphs*.

Completing the picture



Concluding remarks

- Other interesting features:
  - There is a functor **TLexGraph** $\to$ **Loset** recovering the depth-first traversal as the usual vertex order.
  - All of this works for the breadth-first traversal as well! You just* need to use a different path order (the *short-lex* order).
- Open questions:
  - Can useful algorithms actually be extracted from a characterisation such as this?
  - What other algorithms (graph or otherwise) can be formalised in this way?
- Preprint available! Email me: `robin.kaarsgaard@ed.ac.uk`.