A finite presentation of graphs of treewidth at most 3

Amina Doumane, Humeau Samuel, Damien Pous

-

LIP, ENS de Lyon

Outline

1. Graphs, terms, and treewidth

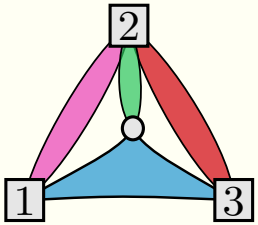
2. The main result

3. Recursively decomposing graphs of treewidth at most 3

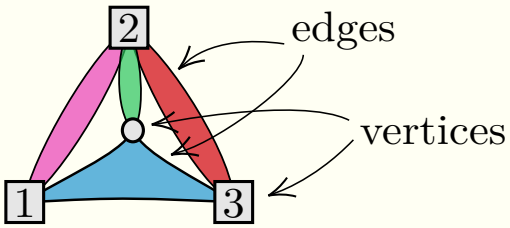
4. Conclusion

Graphs, terms, and treewidth

Graphs

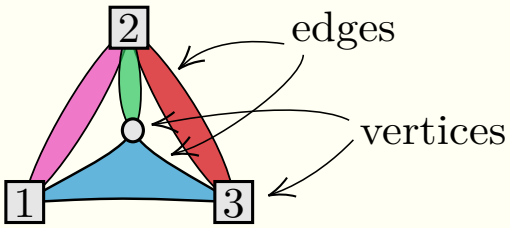


Graphs

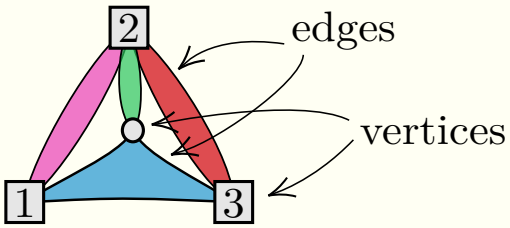


Graphs

Specificities:



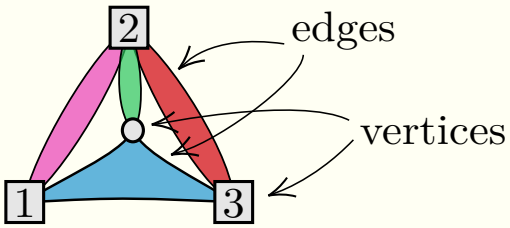
Graphs



Specificities:

- **Sources**: squared vertices, form the interface of the graph,

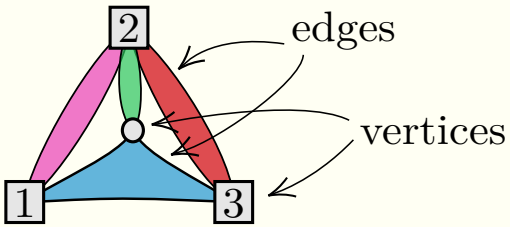
Graphs



Specificities:

- **Sources**: squared vertices, form the interface of the graph,
- **Order**: for graph and edge interfaces,

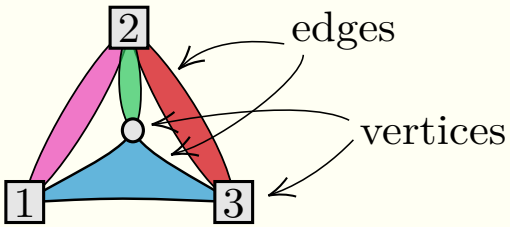
Graphs



Specificities:

- **Sources**: squared vertices, form the interface of the graph,
- **Order**: for graph and edge interfaces,
- **Arity**: number of sources,

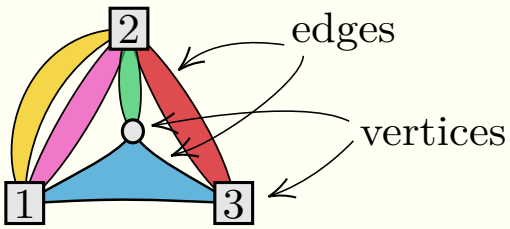
Graphs



Specificities:

- **Sources**: squared vertices, form the interface of the graph,
- **Order**: for graph and edge interfaces,
- **Arity**: number of sources,
- **Hyperedges**,

Graphs

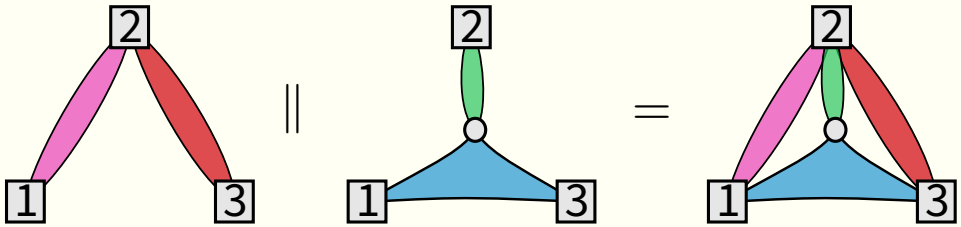


Specificities:

- **Sources**: squared vertices, form the interface of the graph,
- **Order**: for graph and edge interfaces,
- **Arity**: number of sources,
- **Hyperedges**,
- **Multiedges**.

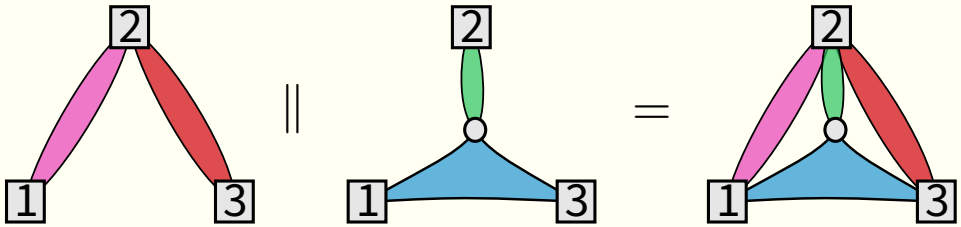
Graph operations and terms

Parallel operation, $- \parallel -$:

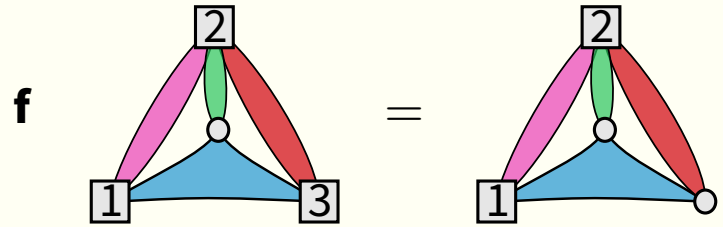


Graph operations and terms

Parallel operation, $- \parallel -$:

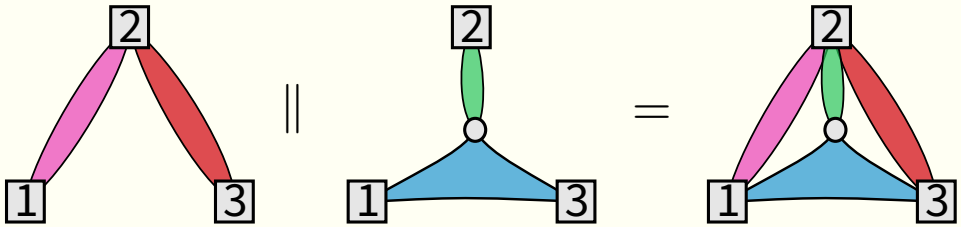


Forget operation, $\mathbf{f}(-)$:

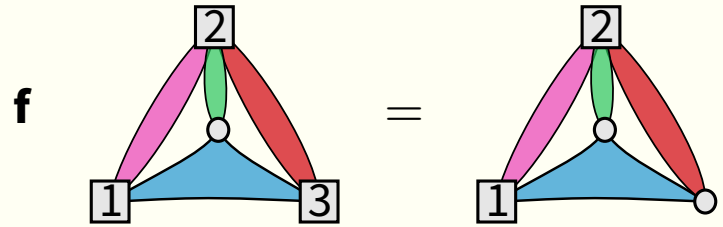


Graph operations and terms

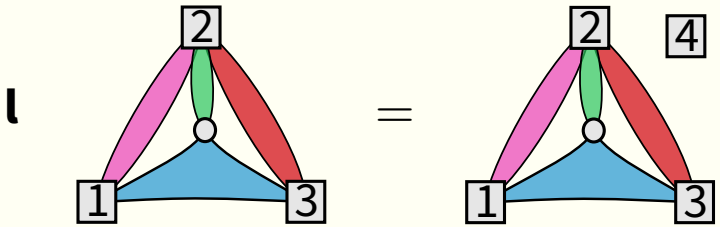
Parallel operation, $- \parallel -$:



Forget operation, $\mathbf{f}(-)$:

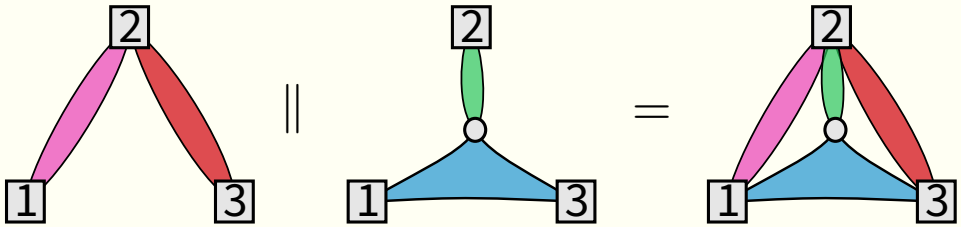


Lift operation, $\mathbf{l}(-)$:

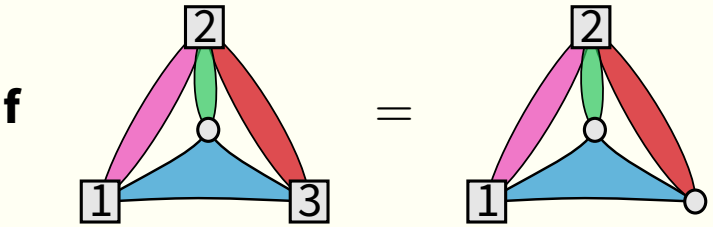


Graph operations and terms

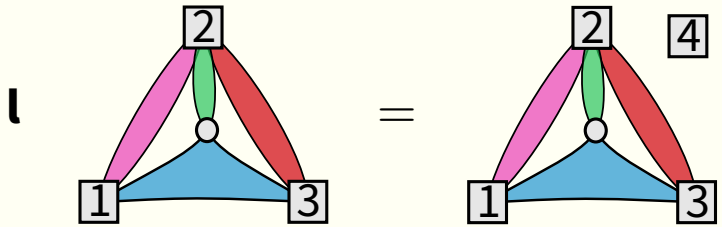
Parallel operation, $- \parallel -$:



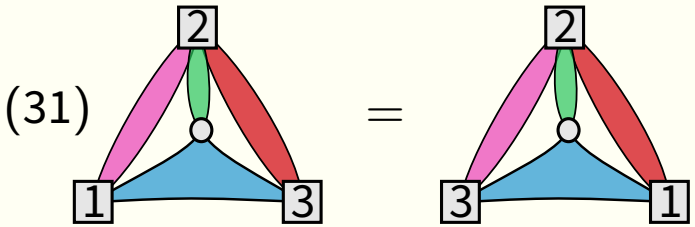
Forget operation, $\mathbf{f}(-)$:



Lift operation, $\mathbf{l}(-)$:

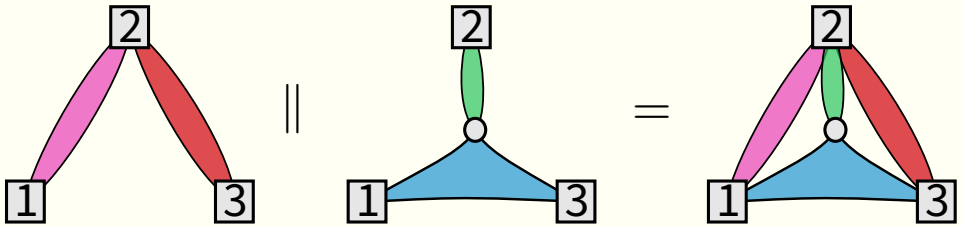


Permutation of sources:

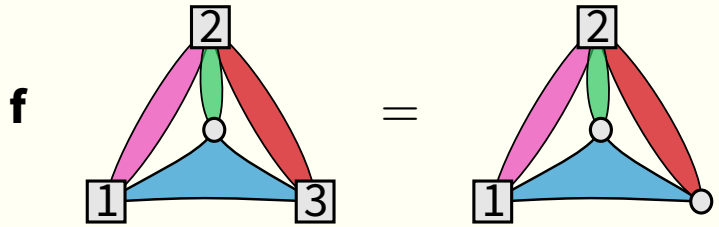


Graph operations and terms

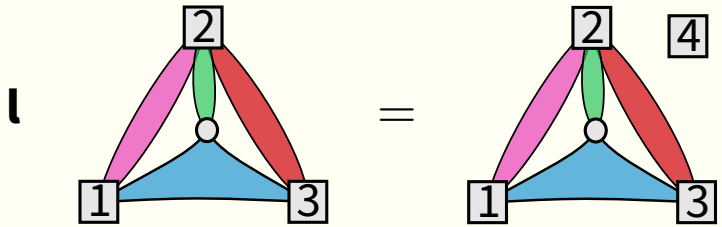
Parallel operation, $- \parallel -$:



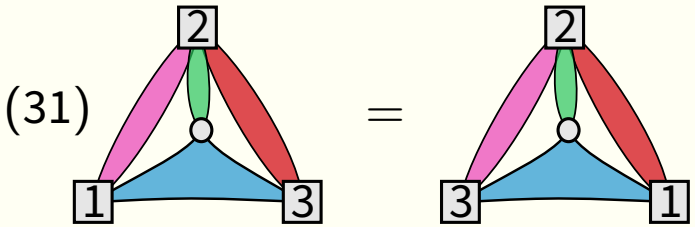
Forget operation, $\mathbf{f}(-)$:



Lift operation, $\mathbf{l}(-)$:



Permutation of sources:

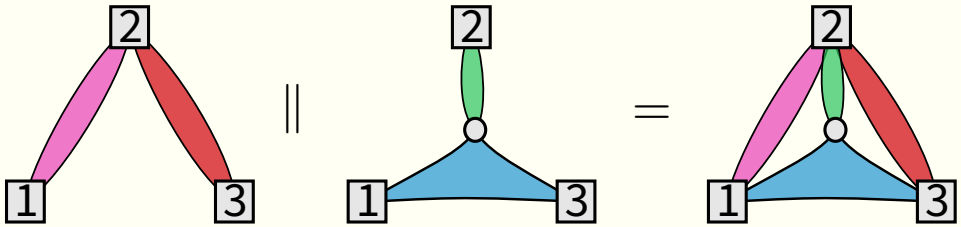


Corresponding grammar:

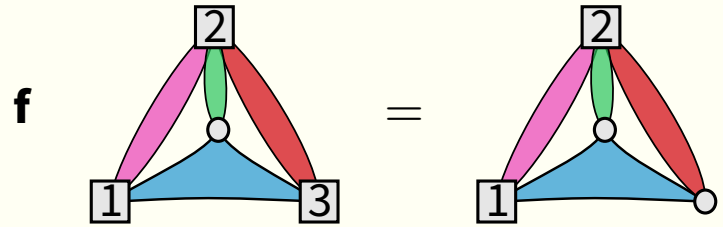
$$t, u ::= a \mid \emptyset \mid t \parallel u \mid \mathbf{l}\, t \mid \mathbf{f}\, t \mid \mathrm{p}\, t,$$

Graph operations and terms

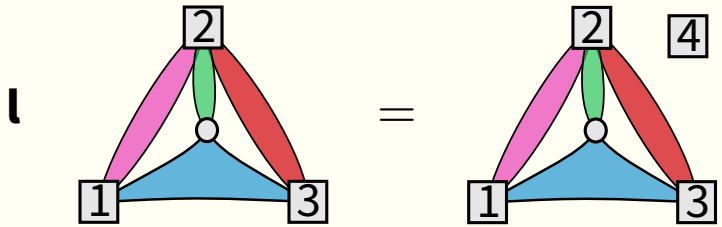
Parallel operation, $- \parallel -$:



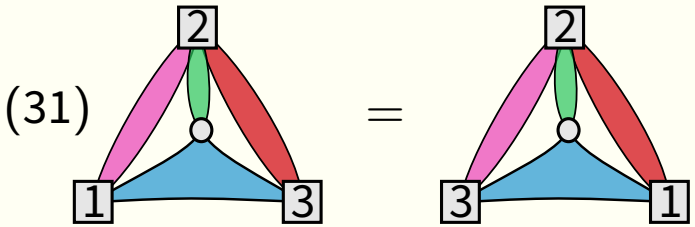
Forget operation, $\mathbf{f}(-)$:



Lift operation, $\mathbf{l}(-)$:



Permutation of sources:



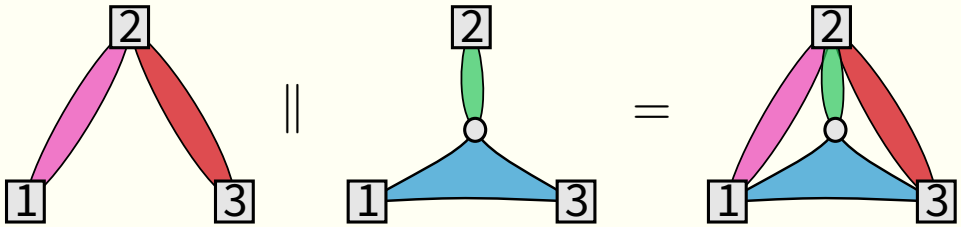
Corresponding grammar:

$$t, u ::= a \mid \emptyset \mid t \parallel u \mid \mathbf{l}\, t \mid \mathbf{f}\, t \mid \mathrm{p}\, t,$$

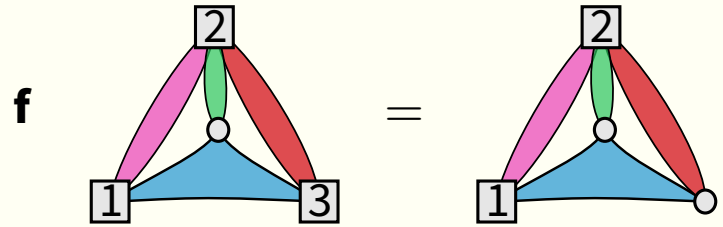
with a an edge, $\emptyset$ a graph reduced to its sources.

Graph operations and terms

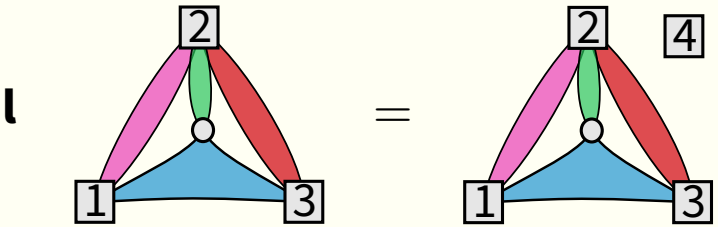
Parallel operation, $- \parallel -$:



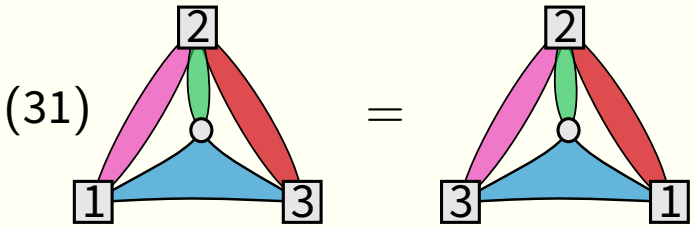
Forget operation, $\mathbf{f}(-)$:



Lift operation, $\mathbf{l}(-)$:



Permutation of sources:

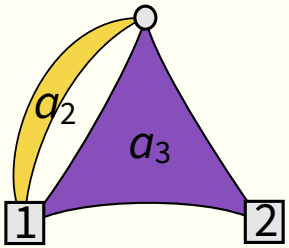


Corresponding grammar:

$$t, u ::= a \mid \emptyset \mid t \parallel u \mid \mathbf{l}\, t \mid \mathbf{f}\, t \mid \mathrm{p}\, t,$$

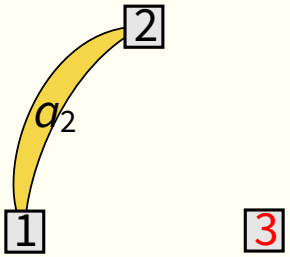
with a an edge, $\emptyset$ a graph reduced to its sources. Note terms have some arity too (a_k and $\emptyset_k$ for the corresponding graphs with k sources).

An example with two parsings



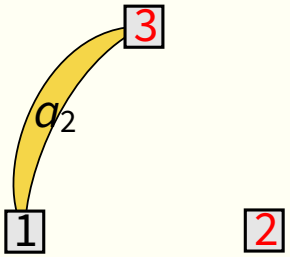
$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

An example with two parsings



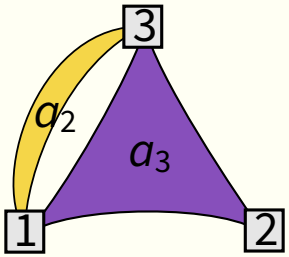
la_2

An example with two parsings



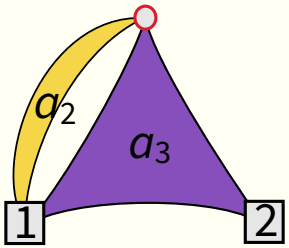
$(32)\ \mathbf{l}\, a_2$

An example with two parsings



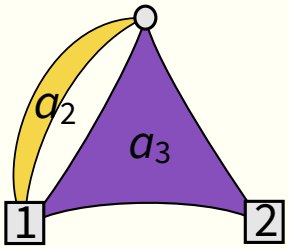
$$a_3 \parallel (32) \mathbf{l} a_2$$

An example with two parsings



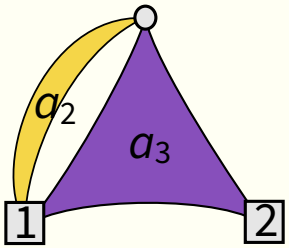
$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

An example with two parsings



$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

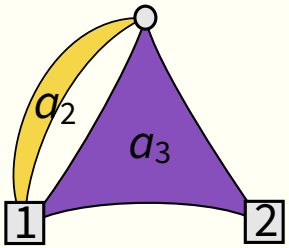
An example with two parsings



$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

- **Parsing** of a graph: a term that constructs the graph.

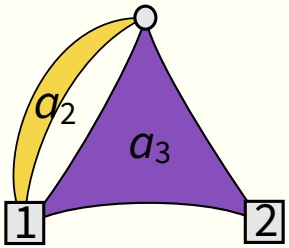
An example with two parsings



$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

- **Parsing** of a graph: a term that constructs the graph.
- **Graph of a term**: noted $\mathbf{g}\,t$.

An example with two parsings



$$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$$

- **Parsing** of a graph: a term that constructs the graph.
- **Graph of a term**: noted $\mathbf{g}\,t$.

Graphs can have multiple parsings: $\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$.

Graphs, terms, and treewidth

Goal: finding axioms that prove equal terms parsing the same graph,

Graphs, terms, and treewidth

Goal: finding axioms that prove equal terms parsing the same graph, focusing on terms of **width** at most 3, meaning (recursively) of arity at most 4.

Graphs, terms, and treewidth

Goal: finding axioms that prove equal terms parsing the same graph, focusing on terms of **width** at most 3, meaning (recursively) of arity at most 4.

Proposition

Given a non-negative integer k, graphs of terms of width at most k are exactly graphs of treewidth at most k.

Graphs, terms, and treewidth

Goal: finding axioms that prove equal terms parsing the same graph, focusing on terms of **width** at most 3, meaning (recursively) of arity at most 4.

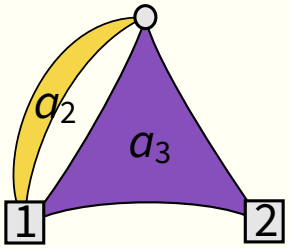
Proposition

Given a non-negative integer k, graphs of terms of width at most k are exactly graphs of treewidth at most k.

Here, this proposition serves as definition for treewidth.

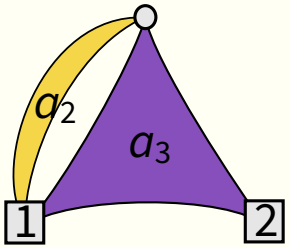
The main result

From one parsing to another ?



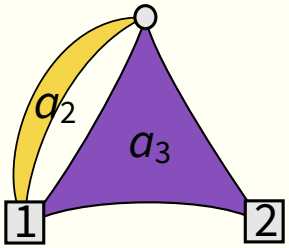
$\mathbf{f}(a_3 \parallel (32)\,\mathbf{l}\,a_2)$
$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$

From one parsing to another ?



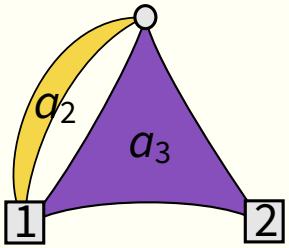
$\mathbf{f}((32)(32)a_3 \parallel (32)\,\mathbf{l}\,a_2)$
$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$

From one parsing to another ?



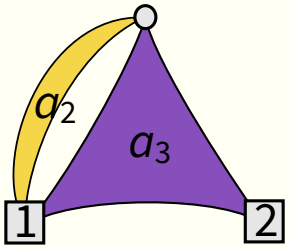
$\mathbf{f}((32)(32)a_3 \parallel (32)\,\mathbf{l}\,a_2)$
$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$

From one parsing to another ?



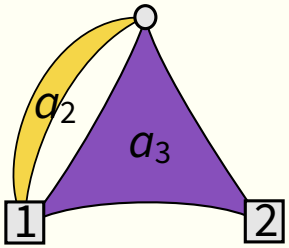
$\mathbf{f}\,(32)((32)a_3 \parallel \mathbf{l}\,a_2)$
$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$

From one parsing to another ?



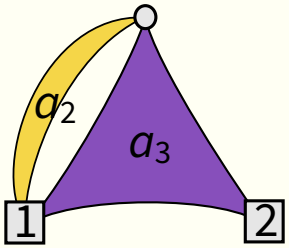
$\mathbf{f}(32)((32)a_3 \,||\, \mathbf{l}\, a_2)$
$\mathbf{f}(32)(\mathbf{l}\, a_2 \,||\, (32)a_3)$

From one parsing to another ?



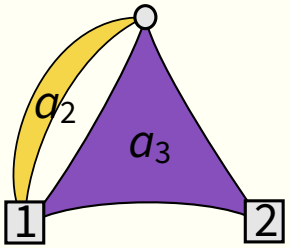
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$

From one parsing to another ?



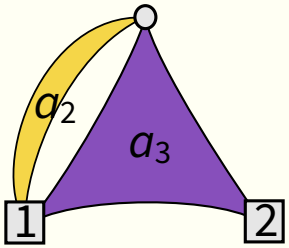
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$

From one parsing to another ?



$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$

From one parsing to another ?



$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$
$$\mathbf{f}(32)(\mathbf{l}\,a_2 \parallel (32)a_3)$$

Axiom: a pair of terms that can be used to prove two terms construct the same graph.

Problem

Is there a finite list of axiom Ax such that for all terms t, u of width at most 3

$$\mathrm{Ax} \vdash t = u \Leftrightarrow \mathbf{g}\,t \simeq \mathbf{g}\,u$$

holds ?

The axioms, the main result, and the method

Axioms Ax given by:

1. $a \parallel (b \parallel c) = (a \parallel b) \parallel c$,

The axioms, the main result, and the method

Axioms Ax given by:

1. $a \parallel (b \parallel c) = (a \parallel b) \parallel c, \quad a \parallel b = b \parallel a,$

The axioms, the main result, and the method

Axioms Ax given by:

1. $a \parallel (b \parallel c) = (a \parallel b) \parallel c$, $\quad a \parallel b = b \parallel a$, and $\quad a \parallel \emptyset = a$,
2. $pqa = (p \circ q)a$, and $\quad \mathrm{id}\, a = a$,
3. $p(a \parallel b) = pa \parallel pb$ and $\quad p\emptyset = \emptyset$,
4. $\mathbf{l}(a \parallel b) = \mathbf{l}\, a \parallel \mathbf{l}\, b$ and $\quad \mathbf{l}\, \emptyset = \emptyset$,
5. $p\, \mathbf{f}\, a = \mathbf{f}\, \dot{p} a$ and $\quad \mathbf{l}\, pa = \dot{p}\, \mathbf{l}\, a$,
6. $\mathbf{l}\, \mathbf{f}\, a = \mathbf{f}\, r\, \mathbf{l}\, a$ and $\quad \mathbf{l}\, \mathbf{l}\, a = r\, \mathbf{l}\, \mathbf{l}\, a$,
7. $\mathbf{f}\, a \parallel b = \mathbf{f}(a \parallel \mathbf{l}\, b)$,
8. + 4 other axioms not detailed here.

The axioms, the main result, and the method

Axioms Ax given by:

1. $a \parallel (b \parallel c) = (a \parallel b) \parallel c$, $\quad a \parallel b = b \parallel a$, and $\quad a \parallel \emptyset = a$,
2. $pqa = (p \circ q)a$, and $\quad \mathrm{id}\, a = a$,
3. $p(a \parallel b) = pa \parallel pb$ and $\quad p\emptyset = \emptyset$,
4. $\mathbf{l}(a \parallel b) = \mathbf{l}\, a \parallel \mathbf{l}\, b$ and $\quad \mathbf{l}\, \emptyset = \emptyset$,
5. $p\, \mathbf{f}\, a = \mathbf{f}\, \dot{p} a$ and $\quad \mathbf{l}\, pa = \dot{p}\, \mathbf{l}\, a$,
6. $\mathbf{l}\, \mathbf{f}\, a = \mathbf{f}\, r\, \mathbf{l}\, a$ and $\quad \mathbf{l}\, \mathbf{l}\, a = r\, \mathbf{l}\, \mathbf{l}\, a$,
7. $\mathbf{f}\, a \parallel b = \mathbf{f}(a \parallel \mathbf{l}\, b)$,
8. + 4 other axioms not detailed here.

Theorem

For all terms t, u of width at most 3 $\mathrm{Ax} \vdash t = u \Leftrightarrow \mathbf{g}\, t \simeq \mathbf{g}\, u$.

The axioms, the main result, and the method

Axioms Ax given by:

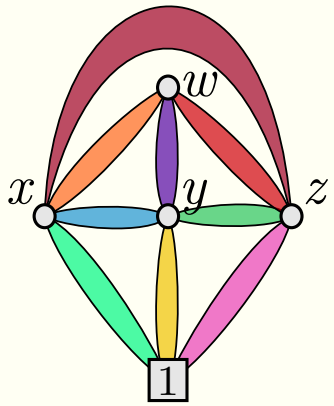
1. $a \parallel (b \parallel c) = (a \parallel b) \parallel c$, $\quad a \parallel b = b \parallel a$, and $\quad a \parallel \emptyset = a$,
2. $pqa = (p \circ q)a$, and $\quad \mathrm{id}\, a = a$,
3. $p(a \parallel b) = pa \parallel pb$ and $\quad p\emptyset = \emptyset$,
4. $\mathbf{l}(a \parallel b) = \mathbf{l}\, a \parallel \mathbf{l}\, b$ and $\quad \mathbf{l}\, \emptyset = \emptyset$,
5. $p\, \mathbf{f}\, a = \mathbf{f}\, \dot{p} a$ and $\quad \mathbf{l}\, pa = \dot{p}\, \mathbf{l}\, a$,
6. $\mathbf{l}\, \mathbf{f}\, a = \mathbf{f}\, r\, \mathbf{l}\, a$ and $\quad \mathbf{l}\, \mathbf{l}\, a = r\, \mathbf{l}\, \mathbf{l}\, a$,
7. $\mathbf{f}\, a \parallel b = \mathbf{f}(a \parallel \mathbf{l}\, b)$,
8. + 4 other axioms not detailed here.

Theorem

For all terms t, u of width at most 3 $\mathrm{Ax} \vdash t = u \Leftrightarrow \mathbf{g}\, t \simeq \mathbf{g}\, u$.

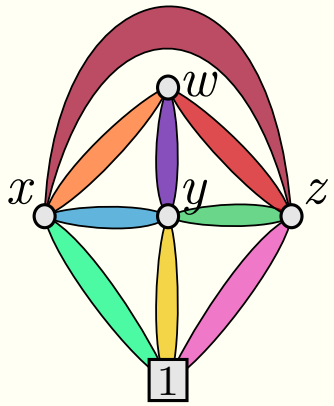
Proof idea: we use axioms to recursively decompose terms following connectivity structures adapted to our graphs.

The main difficulty: forget points non-determinism



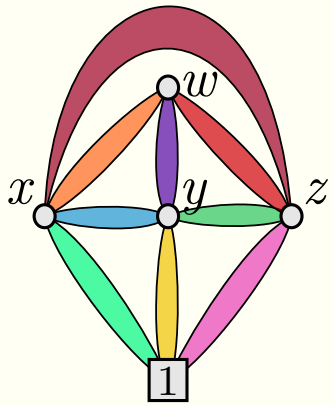
- making w a source gives a graph of treewidth 4 (because we would have K_5),

The main difficulty: forget points non-determinism



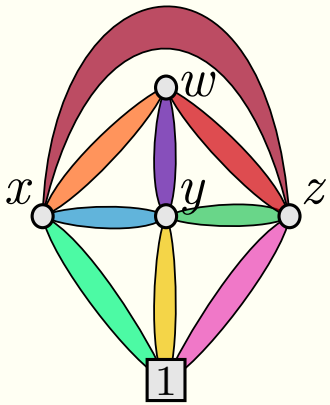
- making w a source gives a graph of treewidth 4 (because we would have K_5),
- making x a source gives a graph of treewidth 3,

The main difficulty: forget points non-determinism



- making w a source gives a graph of treewidth 4 (because we would have K_5),
- making x a source gives a graph of treewidth 3,
- and similarly for y and z.

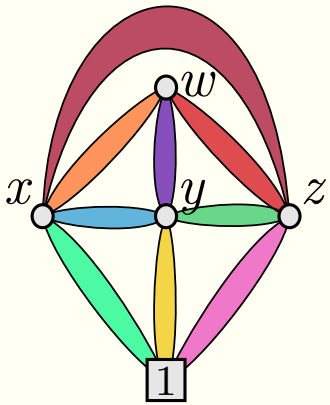
The main difficulty: forget points non-determinism



- making w a source gives a graph of treewidth 4 (because we would have K_5),
- making x a source gives a graph of treewidth 3,
- and similarly for y and z.

x, y, and z (but not w) are called **forget points**.

The main difficulty: forget points non-determinism

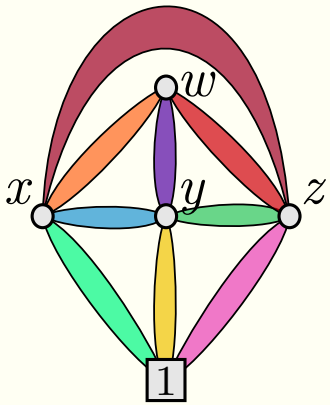


- making w a source gives a graph of treewidth 4 (because we would have K_5),
- making x a source gives a graph of treewidth 3,
- and similarly for y and z.

x, y, and z (but not w) are called **forget points**.

The forget operation is associated to forget points. The main difficulty in the proof of the main result is handling the non-determinism of forget points

The main difficulty: forget points non-determinism



- making w a source gives a graph of treewidth 4 (because we would have K_5),
- making x a source gives a graph of treewidth 3,
- and similarly for y and z.

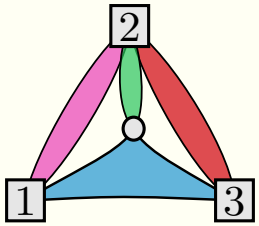
x, y, and z (but not w) are called **forget points**.

The forget operation is associated to forget points. The main difficulty in the proof of the main result is handling the non-determinism of forget points: a graph may have many of them as well as vertices that are not some.

Recursively decomposing graphs of treewidth at most 3

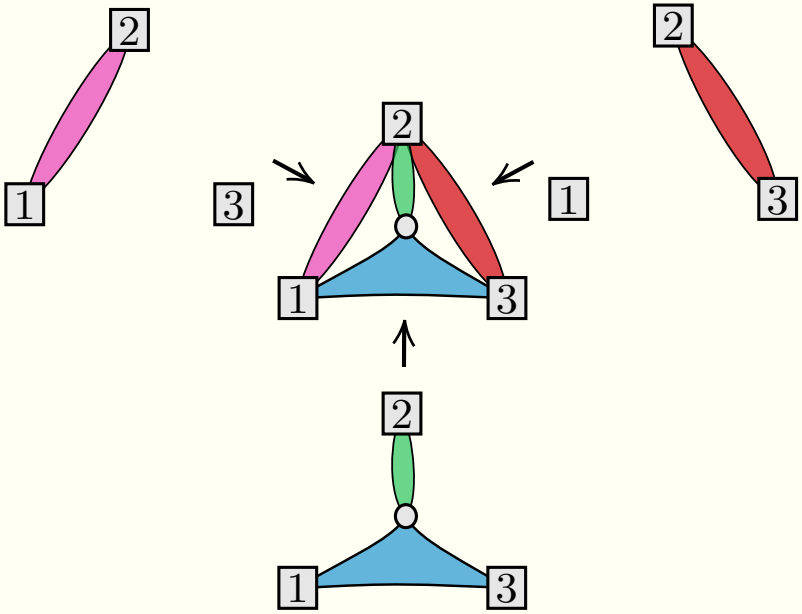
Decomposing graphs: prime decompositions

Consider the following graph:



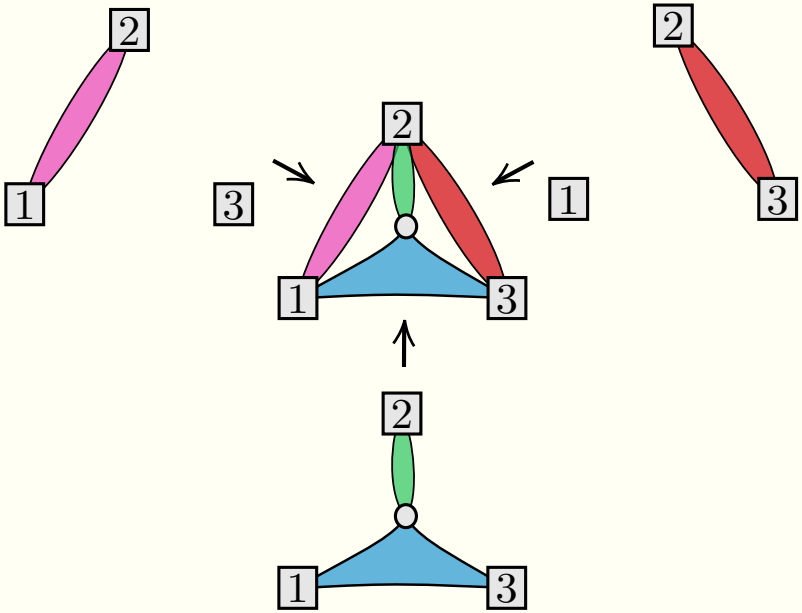
Decomposing graphs: prime decompositions

Consider the following graph:



Decomposing graphs: prime decompositions

Consider the following graph:

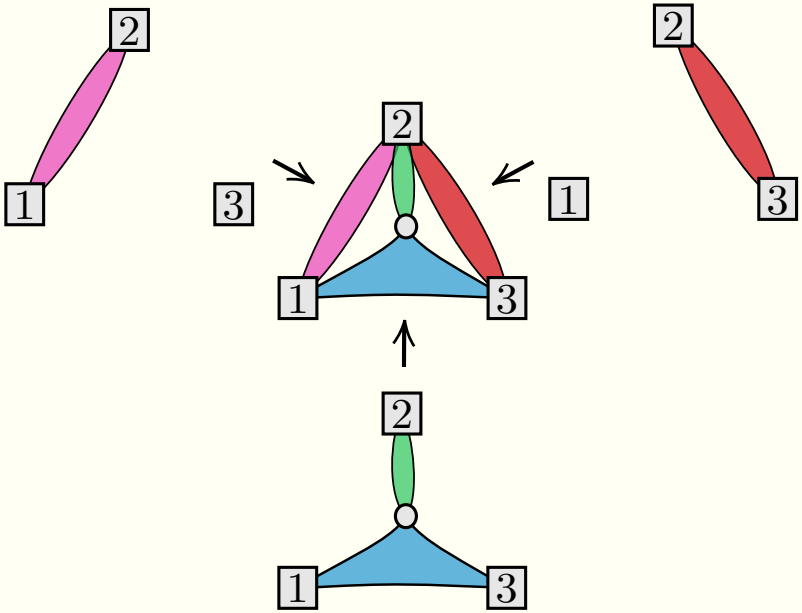


Basic simple graphs

- **Paths** between vertices

Decomposing graphs: prime decompositions

Consider the following graph:

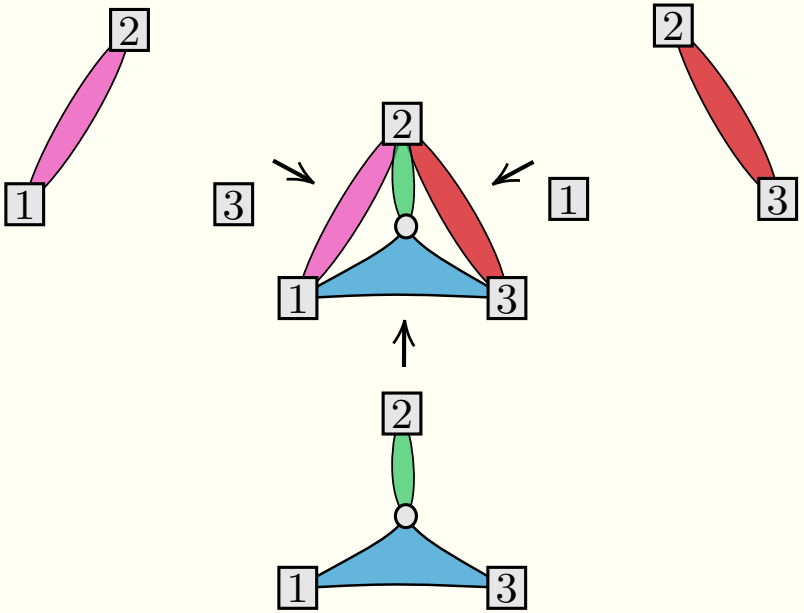


Basic simple graphs

- **Paths** between vertices,
- **Connected components**

Decomposing graphs: prime decompositions

Consider the following graph:

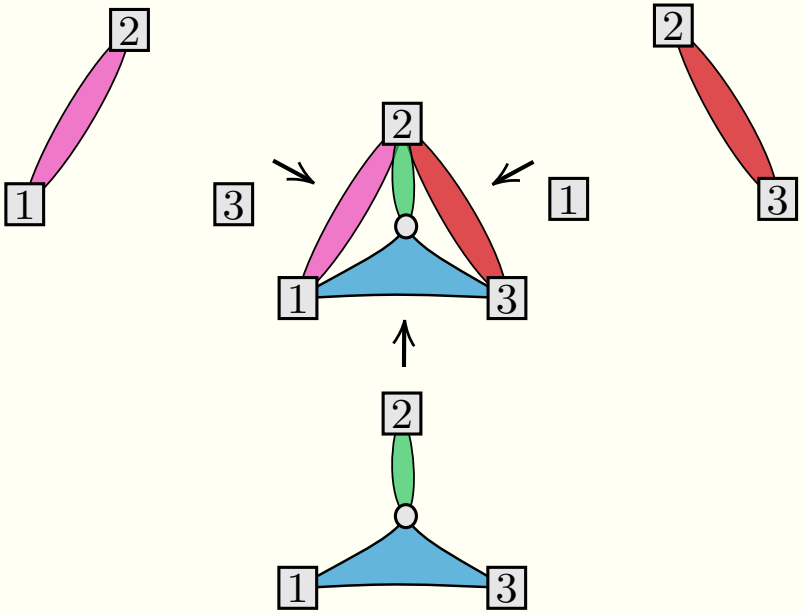


Basic simple graphs

- **Paths** between vertices,
- **Connected components**,
- **Proposition:** for a graph G, $G \simeq \uplus_i H_i$ with H_i its connected components.

Decomposing graphs: prime decompositions

Consider the following graph:



Basic simple graphs

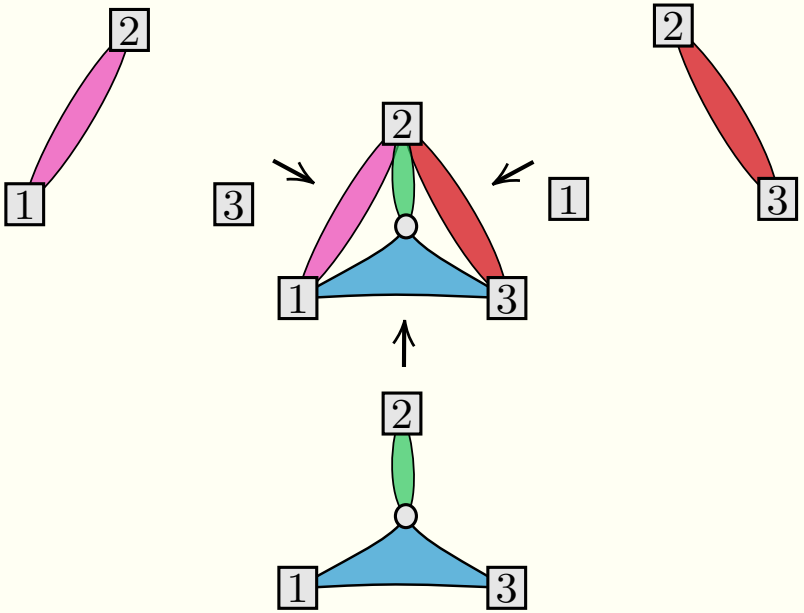
- **Paths** between vertices,
- **Connected components**,
- **Proposition:** for a graph G, $G \simeq \uplus_i H_i$ with H_i its connected components.

Sourced graphs

- **Paths without sources**

Decomposing graphs: prime decompositions

Consider the following graph:



Basic simple graphs

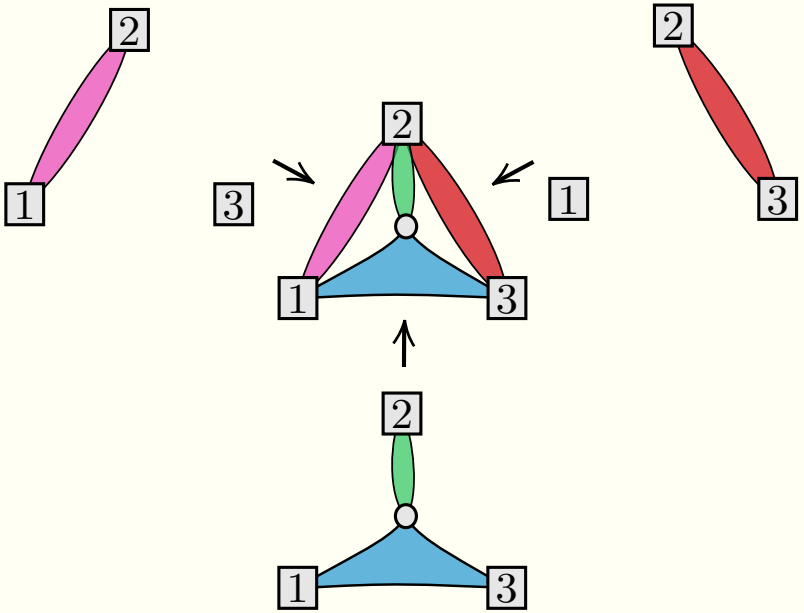
- **Paths** between vertices,
- **Connected components**,
- **Proposition:** for a graph G, $G \simeq \uplus_i H_i$ with H_i its connected components.

Sourced graphs

- **Paths without sources**,
- **Prime components**

Decomposing graphs: prime decompositions

Consider the following graph:



Basic simple graphs

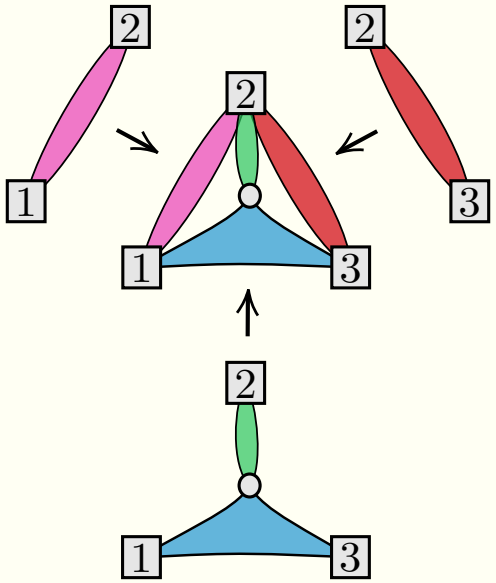
- **Paths** between vertices,
- **Connected components**,
- **Proposition:** for a graph G, $G \simeq \uplus_i H_i$ with H_i its connected components.

Sourced graphs

- **Paths without sources**,
- **Prime components**,
- **Proposition:** for a graph G, $G \simeq \|_i H_i$ with H_i its prime components

Decomposing graphs: full prime decompositions

Consider the following graph:



Basic simple graphs

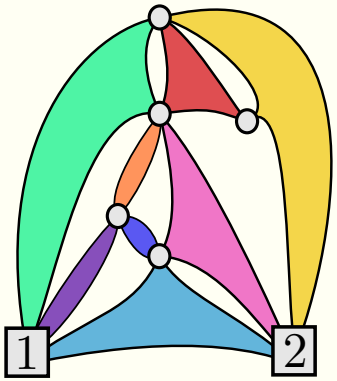
- **Paths** between vertices,
- **Connected components**
- **Proposition:** for a graph G, $G \simeq \uplus_i H_i$ with H_i its connected components.

Sourced graphs

- **Paths without sources**, except at endpoints
- **Full prime components**,
- **Proposition:** for a graph G, $G \simeq \|_i\, p_i \bullet^{j_i} H_i$ with H_i its full prime components

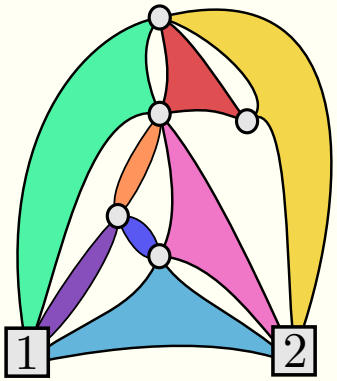
Decomposing graphs: anchors

Consider the following graph:



Decomposing graphs: anchors

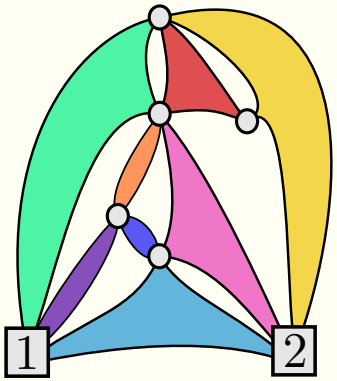
Consider the following graph:



To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Decomposing graphs: anchors

Consider the following graph:



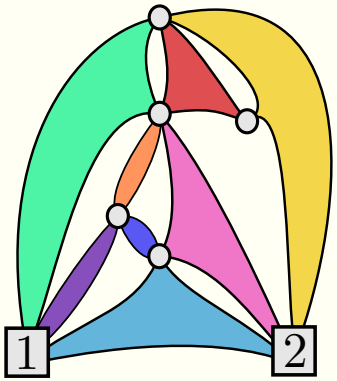
To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Basic simple graphs:

Cutvertices: vertices such that removing them disconnects the graph:

Decomposing graphs: anchors

Consider the following graph:

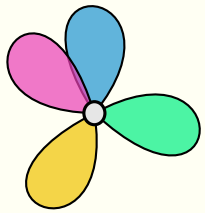


To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Basic simple graphs:

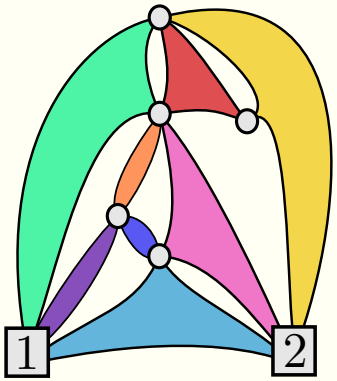
Cutvertices: vertices such that removing them disconnects the graph:

(petals are parts of the graphs)



Decomposing graphs: anchors

Consider the following graph:

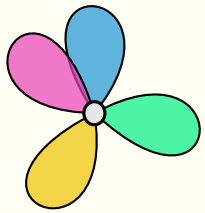


To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Basic simple graphs:

Cutvertices: vertices such that removing them disconnects the graph:

(petals are parts of the graphs)

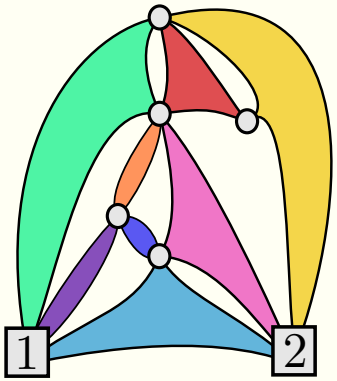


Sourced graphs:

Anchors:

Decomposing graphs: anchors

Consider the following graph:

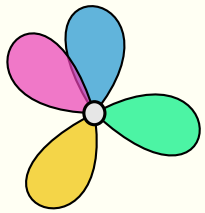


To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Basic simple graphs:

Cutvertices: vertices such that removing them disconnects the graph:

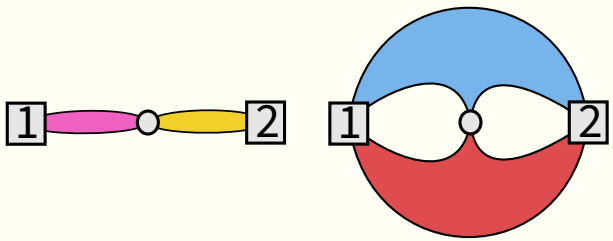
(petals are parts of the graphs)



Sourced graphs:

Anchors:

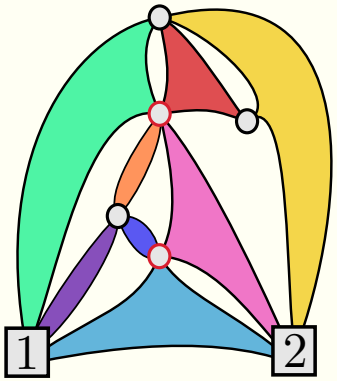
(here for arity 2)



$+1$ case

Decomposing graphs: anchors

Consider the following graph:

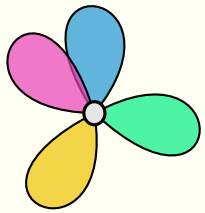


To decompose the graph further, we need to find some **forget points**, and do so as canonically as possible.

Basic simple graphs:

Cutvertices: vertices such that removing them disconnects the graph:

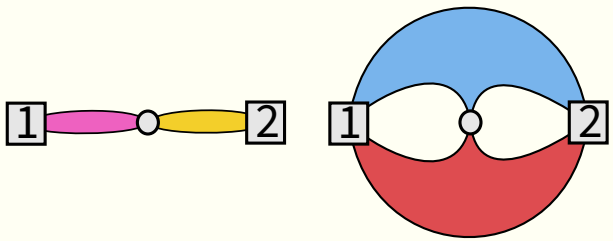
(petals are parts of the graphs)



Sourced graphs:

Anchors:

(here for arity 2)



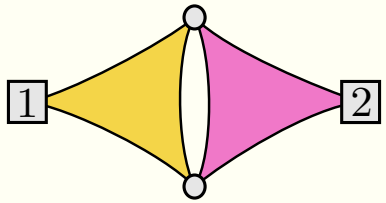
$+1$ case

Decomposing graphs: separation pairs

Hard graph: a full prime graph without anchors.

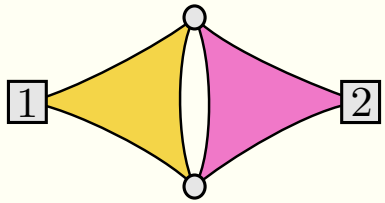
Decomposing graphs: separation pairs

Hard graph: a full prime graph without anchors.
Most basic hard graph:

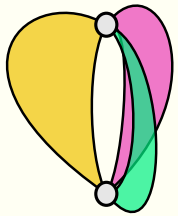


Decomposing graphs: separation pairs

Hard graph: a full prime graph without anchors.
Most basic hard graph:

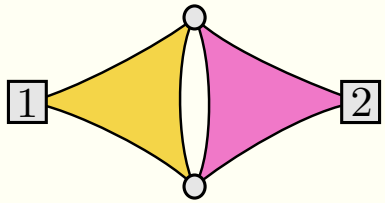


Basic simple graphs:
2-separators: pairs of vertices disconnecting the graph:

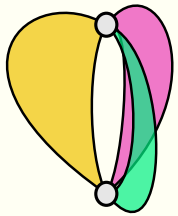


Decomposing graphs: separation pairs

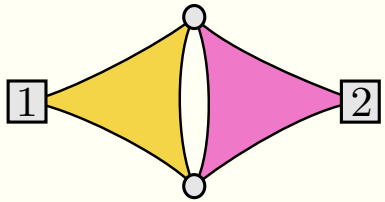
Hard graph: a full prime graph without anchors.
Most basic hard graph:



Basic simple graphs:
2-separators: pairs of vertices disconnecting the graph:

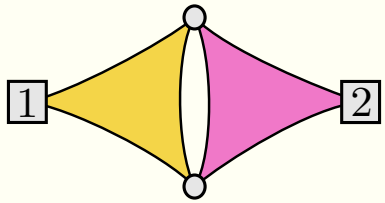


Sourced graphs:
Separation pair:

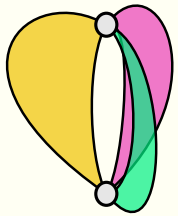


Decomposing graphs: separation pairs

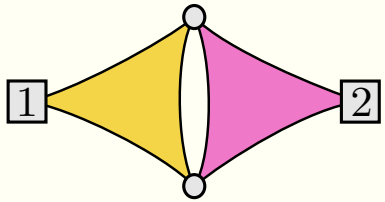
Hard graph: a full prime graph without anchors.
Most basic hard graph:



Basic simple graphs:
2-separators: pairs of vertices disconnecting the graph:



Sourced graphs:
Separation pair:



Proposition

Every hard graph of treewidth at most 3 has at least one separation pair consisting of forget points.

Classifying hard graphs of treewidth at most 3

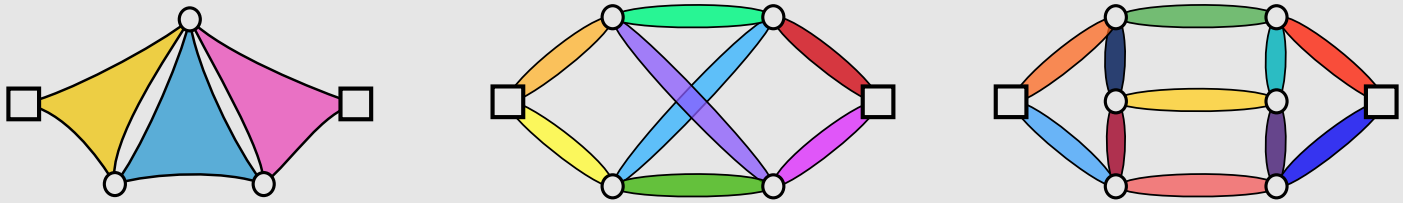
What happens if a hard graph has several separation pairs ?

Classifying hard graphs of treewidth at most 3

What happens if a hard graph has several separation pairs ?

Theorem

Whenever a hard graph has two distinct separation pairs whose vertices are forget points, then it must have one of the following shapes:



An important lemma

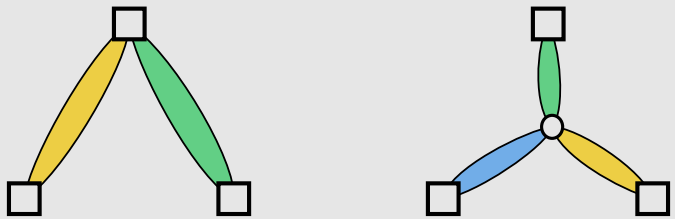
The proof of the last theorem requires a case analysis handling lots of possibilities.

An important lemma

The proof of the last theorem requires a case analysis handling lots of possibilities. The following lemma reduces their number:

Lemma

A graph with 3 sources either has the triangle K_3 as a minor on its sources, or it has one of the following shapes:

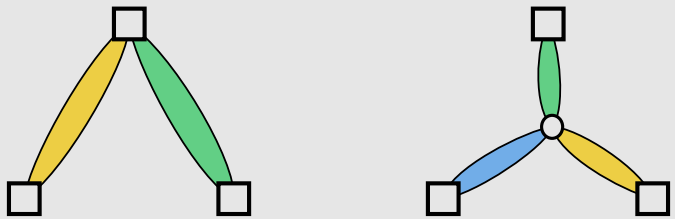


An important lemma

The proof of the last theorem requires a case analysis handling lots of possibilities. The following lemma reduces their number:

Lemma

A graph with 3 sources either has the triangle K_3 as a minor on its sources, or it has one of the following shapes:



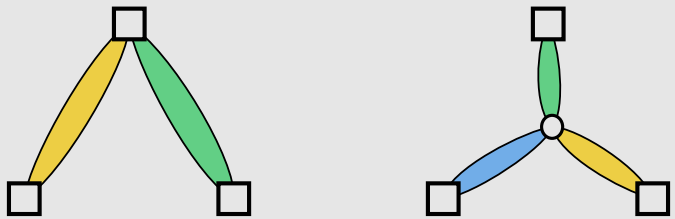
Graph minors: a way to see patterns in graphs, generalising the notion of subgraphs.

An important lemma

The proof of the last theorem requires a case analysis handling lots of possibilities. The following lemma reduces their number:

Lemma

A graph with 3 sources either has the triangle K_3 as a minor on its sources, or it has one of the following shapes:



Graph minors: a way to see patterns in graphs, generalising the notion of subgraphs.
This is reminiscent of the proposition stating that a graph without a cycle (i.e. without K_3 as a minor) is a tree. Our lemma is stronger.

Conclusion

Generalising ?

Generalising to treewidth at most 4 ?

Generalising ?

Generalising to treewidth at most 4 ?

- we can go up to 3-separators but no more,

Generalising ?

Generalising to treewidth at most 4 ?

- we can go up to 3-separators but no more,
- the number of analysed cases of the problem explodes,

Generalising ?

Generalising to treewidth at most 4 ?

- we can go up to 3-separators but no more,
- the number of analysed cases of the problem explodes,
- the main tool to reduce it here is the last lemma, which would need to be generalised for K_4 instead of K_3,

Generalising ?

Generalising to treewidth at most 4 ?

- we can go up to 3-separators but no more,
- the number of analysed cases of the problem explodes,
- the main tool to reduce it here is the last lemma, which would need to be generalised for K_4 instead of K_3,
- but such a generalisation is not possible without getting a weaker statement.

Algorithmic applications ?

Two possible **algorithmic applications**:

Algorithmic applications ?

Two possible **algorithmic applications**: testing treewidth at most 3,

Algorithmic applications ?

Two possible **algorithmic applications**: testing treewidth at most 3, solving the isomorphism problem for graphs of treewidth at most 3

Algorithmic applications ?

Two possible **algorithmic applications**: testing treewidth at most 3, solving the isomorphism problem for graphs of treewidth at most 3:

- no polynomial algorithms are known for these problems when treewidth is not fixed

Algorithmic applications ?

Two possible **algorithmic applications**: testing treewidth at most 3, solving the isomorphism problem for graphs of treewidth at most 3:

- no polynomial algorithms are known for these problems when treewidth is not fixed,
- linear algorithms are already known for both instances

Algorithmic applications ?

Two possible **algorithmic applications**: testing treewidth at most 3, solving the isomorphism problem for graphs of treewidth at most 3:

- no polynomial algorithms are known for these problems when treewidth is not fixed,
- linear algorithms are already known for both instances,
- but the structure our algorithms would follow is different from what exists already: we hope it can be used as heuristics for the problems on general graphs.

Summary

Reminders:

- graphs with sources, multi- and hyperedges,
- operations, terms, and treewidth,
- getting axioms for treewidth at most 3 ?
- Yes, proven by decomposing graphs following their connectivity structure (full prime decompositions, anchors, and separation pairs),
- two research lines: generalisation and algorithmic aspects (testing treewidth at most 3 and the isomorphism on graphs of treewidth at most 3).

Thank you for listening !