

TALLINN UNIVERSITY OF TECHNOLOGY  
DOCTORAL THESIS  
XX/2023

# Monoidal Width

ELENA DI LAVORE



TALLINN UNIVERSITY OF TECHNOLOGY  
School of IT  
Department of Software Science

**The dissertation was accepted for the defence of the degree of Doctor of Philosophy (Computer Science) on 9 November 2023**

**Supervisor:** Professor Paweł Sobociński,  
Department of Software Science, School of IT,  
Tallinn University of Technology  
Tallinn, Estonia

**Opponents:** Professor Samson Abramsky,  
University College London,  
London, United Kingdom

Professor Dan Marsden,  
University of Nottingham,  
Nottingham, United Kingdom

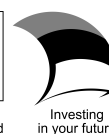
**Defence of the thesis:** 17 November 2023, Tallinn

**Declaration:**

*Hereby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology, has not been submitted for any academic degree elsewhere.*

Elena Di Lavore

\_\_\_\_\_ signature



TALLINNA TEHNIKAÜLIKOO  
DOKTORITÖÖ  
XX/2023

## Monoidiline Laius

ELENA DI LAVORE





## Abstract

Compositionality lies at the core of abstraction: local windows on a problem can be combined into a global understanding of it; models and code can be written so that parts can be reused or replaced without breaking the whole; problems can be solved by combining partial solutions. Compositionality may give algorithmic advantages as well. This is the case of divide-and-conquer algorithms, which use the compositional structure of problems to solve them efficiently. Courcelle's theorems are a remarkable example. They rely on a divide-and-conquer algorithm to show that checking monadic second order formulae is tractable on graphs of bounded tree or clique width.

The idea behind fixed-parameter tractability results of this kind is that divide-and-conquer algorithms are efficient on inputs that are structurally simple. In the case of graphs, tree and clique widths measure their structural complexity. When a graph has low width, combining partial solutions on it is tractable. This work aims to bring the techniques from parametrised complexity to monoidal categories.

This thesis introduces monoidal width to measure the structural complexity of morphisms in monoidal categories and investigates some of its properties. By choosing suitable categorical algebras, monoidal width captures tree width and clique width. Monoidal width relies on monoidal decompositions in the same way graph widths rely on graph decompositions and graph expressions. Monoidal decompositions are terms in the language of monoidal categories that specify the compositional structure needed by divide-and-conquer algorithms. A general strategy to obtain fixed-parameter tractability results for problems on monoidal categories highlights the conceptual importance of monoidal width: compositional algorithms make functorial problems tractable on morphisms of bounded monoidal width.



## Kokkuvõte

Kompositsioonilisus on abstraktsiooni juures tsentraalne: ülesande mõistmise osade kaupa saab kokku panna selle tervikuna mõistmiseks; mudelid ja koodi saab arendada nii, et nende osi on võimalik asendada või taaskasutada tervikut rikkumata; ülesande terviklahendus on leitav osalahendusi kombineerides. Kompositsioonilisus võib anda ka algoritmilisi eeliseid. Nii on näiteks jaga-ja-valitse algoritmidega, mille puhul ülesande efektiivseks lahendamiseks kasutatakse ära selle kompositsioonilist struktuuri. Üheks väljapaistvaks näiteks sellest on Courcelle'i teoreemid. Need põhinevad jaga-ja-valitse algoritmil ning näitavad, et monaadiliste teist järku valemite kontroll on praktiliselt arvutatav nendel graafidel, mille puu- või klikilaius on tõkestatud.

Taoliste fikseeritud parameetritega praktilise arvutatavuse tulemuste aluseks on asjaolu, et jaga-ja-valitse algoritmid on tõhusad struktuurselt lihtsate sisendite korral. Graafi puu- ja klikilaius mõõdavad selle struktuuri keerukust ning kui graafi laius on väike, on osalahenduste kombineerimine praktiliselt arvutatav. Siinne töö üritab tuua parametrizeeritud keerukuses kasutatavad võtted monoidilisse kategooriateooriasse.

Käesolev doktoritöö toob sisse monoidilise laiuse mõiste, et mõõta morfismide struktuuri keerukust monoidilistes kategooriates, ning uurib mõningaid selle omadusi. Valides sobiva kategooriad, on monoidiline laius puu- ja klikilaiuse vasteks. Monoidiline laius põhineb monoidilistel dekompositsioonidel samal viisil, nagu graafilaiused põhinevad graafi-dekompositsioonidel ning graafiavaldistel. Monoidilised dekompositsioonid on termid monoidiliste kategooriate keeles, mis kirjeldavad jaga-ja-valitse algoritmidele vajaliku kompositsioonilise struktuuri. Üldine strateegia monoidiliste kategooriate ülesannetel fikseeritud parameetritega praktilise arvutatavuse tulemuste saamiseks toob esile monoidilise laiuse kontseptuaalse olulisuse: kompositsioonilised algoritmid muudavad funktsionaalsed ülesanded praktiliselt arvutatavaks tõkestatud monoidilise laiusega morfismidel.

## Acknowledgements

The first thanks go to Paweł Sobociński, my supervisor, for the academic advice and freedom he gave me, for the trust he put on me, and for the amazing and lively group he has created in Tallinn, where I have grown so much as a researcher. I would like to thank Samson Abramsky, Dan Marsden and Niccolò Veltri for agreeing to examine this thesis and for their helpful feedback.

I owe a big thank you to Tomáš Jakl for introducing me to finite model theory, for his suggestions and for taking the time to read through my notes. I thank Jamie Vicary for inviting me to Cambridge and for his feedback on this work. I have enjoyed my academic visit in Pisa, and I thank Filippo Bonchi and Fabio Gadducci for their feedback and hospitality.

I am grateful to my coauthors, even if most of our joint work is not reflected in this thesis. I thank Giovanni de Felice and Mario Román for turning a stressful deadline into so much fun. I thank Valeria de Paiva for sharing her encyclopaedic knowledge on linear logic and her energy. I thank Nicoletta Sabadini for all the nice conversations we had in Como and for sharing her always grounded intuitions. Her and Bob Walters' work have particularly influenced my research.

I would like to thank all my PhD siblings, and Clémence and Diana in particular, for creating such an enjoyable and productive environment. I will remember these four years with affection.

Thanks from the bottom of my heart to my parents and my brother for their everlasting love and support. Grazie davvero. A special thank you goes to Mario for his ability to turn our ideas into fun papers, his continuous encouragement and support.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Fixed-parameter tractability . . . . .	1
1.2	Monoidal decompositions . . . . .	2
1.3	Related work . . . . .	3
1.4	Contributions and synopsis . . . . .	4
<b>2</b>	<b>Background</b>	<b>7</b>
2.1	Monoidal categories . . . . .	7
2.2	Graph complexity measures . . . . .	13
2.3	Divide-and-conquer algorithms . . . . .	19
<b>3</b>	<b>Monoidal Width</b>	<b>25</b>
3.1	Decompositions in monoidal categories . . . . .	25
3.2	Categories with copy . . . . .	28
3.3	Categories with biproducts . . . . .	31
<b>4</b>	<b>Interlude: Two Perspectives on Graphs</b>	<b>39</b>
4.1	Cospans of hypergraphs and relational structures . . . . .	39
4.2	Matrices . . . . .	44
4.3	Graphs with dangling edges . . . . .	45
<b>5</b>	<b>A Monoidal Algebra for Branch Width</b>	<b>69</b>
5.1	Inductive branch decompositions . . . . .	69
5.2	Bounding branch width . . . . .	73
<b>6</b>	<b>A Monoidal Algebra for Rank Width</b>	<b>77</b>
6.1	Inductive rank decompositions . . . . .	77
6.2	Bounding rank width . . . . .	81
<b>7</b>	<b>A Monoidal Courcelle-Makowsky Theorem</b>	<b>89</b>
7.1	Fixed-parameter tractability in monoidal categories . . . . .	89
7.2	Computing colimits compositionally . . . . .	92
<b>8</b>	<b>Conclusions</b>	<b>97</b>
	<b>Bibliography</b>	<b>99</b>



# Chapter 1

## Introduction

Famously, Caesar used to say “divide et impera”, divide and conquer, as a strategy to overcome enemies. This strategy is sometimes also useful to design algorithms. When the input has a simple structure, solving the problem on its components and then combining the solutions may be more efficient than solving it on the input as a whole. One of the most famous results in parametrised complexity, Courcelle’s theorem, relies on a divide-and-conquer algorithm to bound the time complexity of solving a class of problems on graphs.

### 1.1 Fixed-parameter tractability

Parametrised complexity studies computational complexity of problems depending on parameters. The problems that are tractable for given choices of the parameter are called *fixed-parameter tractable*. Courcelle’s theorem [Cou92a] is one of the most famous results in this field, and shows fixed-parameter tractability of checking monadic second order formulae on graphs. This is a hard problem in general, but becomes tractable when the input is restricted to belong to a class of bounded-width graphs.

There are similar results for different notions of width for graphs [Cou92a; CMR00; CO00]. We will be concerned with the general structure of these results rather than their details. They all rely on a decomposition algebra for graphs to determine the corresponding graph width. A decomposition algebra is a set of operations and a set of generators that allow graphs to be expressed as terms. Each operation has a cost and each term is priced according to the most expensive operation in it. Different terms may express the same graph and have different costs. The *width* of a graph is the cost of one of its cheapest terms.

The second ingredient for fixed-parameter tractability results like Courcelle’s is a preservation theorem. Given a decomposition algebra for graphs and a logic for them, a preservation theorem states that the operations preserve logical equivalence. As a consequence, given a term for a graph, the value of a formula on it can be determined compositionally. This computation is tractable when the input graphs are restricted to a bounded-width class because combining partial solutions takes constant time in the size of the input graph. A famous result of this kind is the Feferman-Vaught-Mostowski theorem [Fef57; FV59] that shows, via Ehrenfeucht-Fraïssé games [Fra55; Fra57; Ehr57; Ehr61], that the disjoint union of graphs preserves monadic second order logical equivalence.

Each fixed-parameter tractability result for checking monadic second order formulae on graphs relies on its own decomposition algebra and relative preservation theorem. The Courcelle-Makowsky theorem [CM02; Mak04] summarises the common technique to all these results. It assumes the existence of a decomposition algebra and a corresponding preservation theorem, which is the difficult part to show, and deduces fixed parameter tractability of checking formulae on graphs. This result is an almost straightforward consequence of

its assumptions but highlights the common proof structure to the mentioned graph fixed-parameter tractability results.

The insight that led to these results is the expression of graphs as terms. The graph widths defined by operations and generators had already been defined combinatorially [RS83; RS86; RS91; OS06] and led to fundamental results in graph theory and combinatorics, such as the famous Robertson and Seymour graph minor theorem [RS04]. However, the algebraic perspective on them gave the possibility to take advantage of graph decompositions to obtain algorithmic results. We bring these insights to the world of monoidal categories, where we define monoidal decompositions and the relative monoidal width. We show that compositional algorithms make functorial problems tractable on morphisms of bounded monoidal width.

Monoidal categories often serve as semantic universes for programs. Depending on the additional structure and properties of the chosen monoidal category, its morphisms may represent different kinds of computations, either classical [Lam86] or with effects [Gui80; Mog91]. With these models, program verification may be done compositionally and one may be able to obtain fixed-parameter tractability results.

For graph decompositions, different sets of operations may define the same width, while for monoidal decompositions, the choice of monoidal category determines the decomposition algebra: the operations are compositions and monoidal product. These are the canonical choice among all the possible operations that define equivalent width measures.

## 1.2 Monoidal decompositions

We define monoidal decompositions and monoidal width mimicking Courcelle's algebraic decompositions of graphs and their width. While for graphs the choice of operations determines the decomposition algebra, for monoidal decompositions it is the choice of monoidal category that determines, canonically, the operations: compositions and monoidal product. A monoidal decomposition of a morphism in a monoidal category is an expression of this morphism in terms of compositions and monoidal products of "smaller" morphisms.

There may be different monoidal decompositions of the same morphism, some more efficient than others, and monoidal width measures the cost of a most efficient decomposition. The cost of a decomposition depends on the operations that appear in it and their cost. The composition of two morphisms may represent running two processes one after the other with some information passed along a channel from the first process to the second, or it may represent running two processes that have access to the same resource and need to synchronise along a common boundary to access the resource. Resource sharing, synchronisation and information sharing are costly operations and their cost increases with the size of the common boundary. We assign to composition operations a cost that increases with the size of the shared boundary. On the other hand, monoidal products usually represent running processes in parallel, without communication. Monoidal products are, thus, usually, cheap operations with constant cost. With these choices, monoidal width incentivises parallelism: highly parallelised monoidal decompositions will be cheaper than highly sequential ones. The monoidal decompositions in Figure 1.1 exemplify this phenomenon. The monoidal decomposition on the left cuts the morphism along 4 wires, while the biggest cut in the one on the right is along 2 wires.



Figure 1.1: An inefficient (left) and an efficient (right) monoidal decompositions.

We study monoidal width in two categorical algebras of graphs. We give syntactic presentations of them in terms of generators and equations. The algebra of discrete cospans of graphs is equivalent to the prop generated by a Frobenius monoid with an added “edge” generator. The algebra of graphs with dangling edges is equivalent to the prop generated by a bialgebra with an added “vertex” generator. We show that Courcelle’s operations for tree width derive from compositions and monoidal product in the monoidal category of Frobenius graphs, while those for rank width and clique width derive from compositions and monoidal product in the monoidal category of bialgebra graphs. In fact, we show that monoidal width in the first of these categories is equivalent to tree width, while, in the second, it is equivalent to clique width.

Inspired by the Courcelle-Makowsky analogous result for graphs [CM02; Mak04], we conclude by giving a general strategy for showing fixed-parameter tractability of problems on monoidal categories. The choice of decomposition algebra is given by fixing a monoidal category of inputs. The structural part of the preservation theorems corresponds to functoriality of the mapping from inputs to solutions, and the computational part of these results bounds the cost of combining partial solutions. Composing two partial solutions needs to be linear in the size of the components, but it can be arbitrarily complex in the size of the common boundary. These conditions make computing solutions efficient on inputs of bounded monoidal width.

### 1.3 Related work

Since the first definitions of graph decompositions and relative widths, there have been two main approaches to them. A more combinatorial one, where decompositions are combinatorial objects, paths or trees with additional data, and a more algebraic one, where decompositions are terms that express graphs as the result of operations applied to generators. Some of the first combinatorial approaches to graph widths define tree decompositions [BB73; Hal76], which proved fundamental for Robertson and Seymour’s graph minors series [RS83] that culminated with the proof of the graph minors theorem [RS04]. This result shows a combinatorial property of graphs: they are well-quasi-ordered under the graph minor partial order. On the other hand, the algebraic and syntactic approaches to graph decompositions led to results in complexity theory. One of the earliest syntactic definitions of graph decompositions define them in terms of operations and generators [PRS88]. This idea was rediscovered by Bauderon and Courcelle [BC87] and developed into Courcelle’s monadic second order logic of graphs series [Cou90]. This line of research led to fixed-parameter tractability results for graphs [Cou92a; CO00; CK09].

Mowshowitz and Dehmer’s review [MD12] give a thorough taxonomy of graph complexity measures, while Bodlaender’s classical review [Bod93b] and a more recent one by Hliněný et al. [Hli+08] summarise algorithmic applications of tree width and related widths.

As mentioned above, the algebraic approach to graph decompositions led to results in parametrised complexity, but this is one of few examples where algebraic, or “structural”, methods have been adopted in complexity theory. Considered the success of this perspective, other recent lines of research aim to bridge the gap between algebraic methods and complexity results, to relate *structure* and *power* [AS21].

**Graph grammars.** Our work follows the syntactic approach to graph decompositions by Bauderon and Courcelle [BC87]. This started the monadic second order logic of graphs series [Cou90] where syntactic decompositions of graphs give the possibility to show fixed-parameter tractability of checking monadic second order formulae on graphs. Different decomposition algebras define different classes of bounded-width graphs. The first decomposition algebra defines tree width [BC87; Cou90] and leads to the relative fixed-parameter tractability result [Cou92a]. Similar results hold for decomposition algebras defining clique width and rank width [CER93; CO00; CK07; CK09]. These results share the proof structure, which is summarised by Courcelle and Makowsky [CM02; Mak04]. We will recall definitions and results about these graph decompositions in Section 2.3 in detail.

Although we will not refer to it later on, it is worth mentioning the twin width series [Bon+21] that recently started an active line of research by defining a graph complexity measure that is stronger than the known ones but still admits fixed-parameter tractable first-order model checking, twin width [Bon+21].

**Game comonads.** Another prolific approach to connect structure and power targets logic games. Logic games are a common, if not the most common, technique to show preservation theorems. The proof of the Feferman-Vaught-Mostowski preservation theorem [Fef57; FV59] relies on Ehrenfeucht-Fraïssé games [Fra55; Fra57; Ehr57; Ehr61] to show logical equivalence of structures. A logic game consists of two players, Spoiler and Duplicator, that in turns choose vertices of two relational structures. Spoiler tries to show that the two structures are not logically equivalent, while Duplicator’s goal is to show that they are. The details of the moves of each player and the details of the rules of the game determine the logic fragment that defines logical equivalence.

Game comonads are families of comonads on the category of relational structures and their homomorphisms that are indexed by a resource. For a game comonad  $\mathbf{C}$ , the comonad  $\mathbf{C}_k$  associates to a relational structure the relational structure of plays on it that use at most  $k$  resources. The type of resource determines the type of logical equivalence of the corresponding game. Intuitively, the resource bounds the size of the windows through which the relational structure can be looked at.

Game comonads unify logic games and their corresponding logical equivalence with graph widths, and systematise these correspondences. For a game comonad  $\mathbf{C}$ , the existence of a winning strategy for Duplicator on structures  $G$  and  $H$  is witnessed by the existence of a coKleisli morphism or isomorphism  $\mathbf{C}_k(G) \rightarrow H$  and characterises logical equivalence for a specific logic fragment. Different comonads define logical equivalence for different logical fragments [ADW17; AM21; AS21; ÓD21; MS22]. Widths are, instead, characterised by the coalgebra number. The coalgebra number of a structure  $G$  with respect to a game comonad  $\mathbf{C}$  is the minimum  $k$  for which  $\mathbf{C}_k$  admits a  $G$ -coalgebra  $G \rightarrow \mathbf{C}_k(G)$ . The pebbling comonad defines tree width [ADW17], the Ehrenfeucht-Fraïssé comonad defines tree depth [AS21] and the pebble-relation comonad defines path width [MS22].

The game comonad approach recovers classical results from finite model theory [Pai20; DJR21; AJP22] and gives general strategies to obtain new ones [AR23]. In particular, Jakl, Marsden and Shah [JMS23] focus on abstracting the Feferman-Vaught-Mostowski preservation theorems, an issue we do not touch upon.

**Cospan decompositions.** Blume et al. [Blu+11] noticed that the categorical algebra behind tree decompositions is that of cospans of graphs. Their work characterises path and tree decompositions in terms of path- and tree-shaped colimits in the category of graphs and their homomorphisms. Following a similar intuition, Bumpus, Kocsis and Master [BK21; Bum21; BKM23] generalised tree decompositions beyond graphs. The starting point of this line of work is a characterisation of tree width in terms of Halin’s  $S$ -functions [Hal76]. These approaches define decompositions “globally”: they are functors whose domain determines the shape of the decomposition.

## 1.4 Contributions and synopsis

This thesis defines monoidal width and investigates some of its properties. It is based on published work by the author [DHS21; DS22; DS23].

- Monoidal width and monoidal decompositions are defined in Section 3.1.
- By choosing a suitable categorical algebra of graphs with vertex interfaces, Theorem 5.16 shows equivalence of monoidal width with branch width and tree width.
- Similarly, Theorem 6.19 relies on a categorical algebra of graphs with edge interfaces to show equivalence of monoidal width with rank width and clique width.

- Theorem 4.44 provides a syntactic presentation of graphs with edge interfaces.
- Theorem 7.6 shows that functorial problems on morphisms in monoidal categories that admit a compositional algorithm (Definitions 7.1 and 7.4) are fixed-parameter tractable with parameter monoidal width. This result mimicks the Courcelle-Makowsky result about fixed-parameter tractability of checking formulae on relational structures.

**Synopsis** Chapter 2 gives some background on both category theory and graph decompositions. Section 2.1 recalls monoidal categories and props, while Sections 2.2 and 2.3 recall graph widths and their application to fixed-parameter tractability results. In particular, we recall the definitions of tree width, branch width, clique width and rank width, both the original combinatorial ones and the ones in terms of operations on graphs and generators.

Chapter 3 introduces monoidal width and two simple examples. The definition of monoidal decompositions and monoidal width are in Section 3.1, and Sections 3.2 and 3.3 study monoidal width of coherent copy morphisms and in categories with biproducts.

The main study case for monoidal decompositions are graphs. Chapter 4 recalls two categories where morphisms are graphs with interfaces, one where the interfaces are vertices, in Section 4.1, and one where the interfaces are edges, in Section 4.3. Graphs with vertex interfaces are discrete cospans of graphs and can be syntactically presented by a Frobenius monoid with an added “edge” generator. Graphs with edge interfaces are matrices quotiented by an equivalence relation and can be syntactically presented by a bialgebra with an added “vertex” generator. We show how compositions and monoidal products in the monoidal category of Frobenius graphs express the operations for tree decompositions, while those in the monoidal category of bialgebra graphs express the operations for rank and clique decompositions. Chapter 5 is dedicated to showing that monoidal width in the monoidal category of Frobenius graphs is equivalent to branch and tree widths, while Chapter 6 shows that monoidal width in the monoidal category of bialgebra graphs is equivalent to rank and clique widths. These equivalences rely on constructing a monoidal decomposition from a branch or rank decomposition and vice versa. As intermediate step between monoidal and graph decompositions we construct inductive branch and rank decompositions.

Chapter 7 concludes with a version of the Courcelle-Makowsky theorem for fixed-parameter tractability for problems on monoidal categories. Section 7.2 applies this result to the case of computing colimits in presheaf categories.





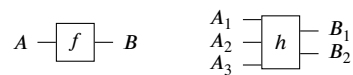
## Chapter 2

# Background

This chapter introduces some background about monoidal categories and fixed-parameter tractability in the attempt to make this work accessible from both the “structure” community studying category theory and the “power” community studying computational complexity. Section 2.1 recalls the definitions of monoidal category and prop, and their string diagrammatic syntax and interpretation as theories of processes. Section 2.3 recalls relational structures, some preservation theorems and their consequences as fixed-parameter tractability results.

### 2.1 Monoidal categories

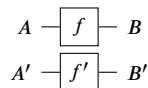
Monoidal categories [Mac63] often serve as process theories. Depending on the additional properties and structure on the chosen monoidal category, its morphisms may represent classical computations [Lam86; JH90], computations with effects [Gui80; Mog91; AM99; CFS16; Rom23] or different kinds of computational models, from automata [KSW97b; KSW97a; Di +23; Di +21a], to signal flow graphs [BSZ14; BSZ15] and dataflow computations [Oli84; Şte86b; Şte86a; KSW99; KSW02; UV08; MHH16; SK19; CVP21; DFR22; Gar23]. Similarly, they may represent processes of different kinds, like stochastic processes [Pan99; Fri20; Sta17; Ste21; DR23], linear processes [BSZ17; Bon+19b; Bon+19a], partial processes [Car87; RR88; CO89; CLO7; Di +21b], or quantum processes [ACO9; CS12; HV19]. Morphisms are depicted as boxes with input and output wires. These wires are the objects, which specify the resources that can be transformed by processes.



The categorical structure allows processes to be composed sequentially: for two morphisms  $f : A \rightarrow B$  and  $g : B \rightarrow C$ , there is a composite morphism  $f \circ g : A \rightarrow C$  that, usually, represent the process of executing  $f$  first and then  $g$ .



The monoidal structure also allows morphisms to be composed in parallel: for two morphisms  $f : A \rightarrow B$  and  $f' : A' \rightarrow B'$ , there is a composite morphism  $f \otimes f' : A \otimes A' \rightarrow B \otimes B'$  that, usually, represent the process of executing  $f$  and  $f'$  at the same time.



Both these composition operations have units. The identity morphisms  $\mathbb{1}_A$  are the units for sequential composition: they represent the process that “does nothing” to a resource, so composing sequentially with the identity morphism should not change the process.

$$A \text{ --- } \boxed{f} \text{ --- } B = A \text{ --- } \boxed{f} \text{ --- } B = A \text{ --- } \boxed{f} \text{ --- } B$$

The monoidal unit  $I$  is the unit for parallel composition: it represents “absence of resources”, so a process that produces as outputs, or requires as inputs, a resource  $A$  and the monoidal unit  $I$ , it is essentially the same as the same process only producing, or requiring,  $A$ . This reflects in the algebra of monoidal categories with natural isomorphisms  $A \otimes I \cong A \cong I \otimes A$ . Some processes may not take any inputs,  $s : I \rightarrow B$ , or do not produce any outputs,  $t : A \rightarrow I$ .

$$\boxed{s} \text{ --- } B \quad A \text{ --- } \boxed{t}$$

The string diagrammatic syntax is convenient because it hides the bureaucracy isomorphisms that ensure associativity and unitality of the monoidal structure, and equations like functoriality of the monoidal product,  $(f \otimes f') \circ (g \otimes g') = (f \circ g) \otimes (f' \circ g')$  also become trivial in string diagrams.

$$\begin{array}{c} A \text{ --- } \boxed{f} \text{ --- } \boxed{g} \text{ --- } C \\ A' \text{ --- } \boxed{f'} \text{ --- } \boxed{g'} \text{ --- } C' \end{array}$$

A monoidal category is a category with extra structure, the monoidal product  $\otimes$  and monoidal unit  $I$ , subject to coherence conditions given by natural transformations that witness associativity,  $\alpha$ , and unitality,  $\lambda$  and  $\rho$ , of the monoidal structure.

**Definition 2.1** ([Mac63]). A monoidal category  $(C, \otimes, I)$  is given by a category  $C$ , a functor  $(-\otimes -) : C \times C \rightarrow C$  and an object  $I$  of  $C$  with coherence natural isomorphisms  $\alpha : (-\otimes(-\otimes-)) \rightarrow ((-\otimes-)\otimes-)$ , the associator,  $\lambda : (I \otimes -) \rightarrow \mathbb{1}$ , the left unitor, and  $\rho : (-\otimes I) \rightarrow \mathbb{1}$ , the right unitor, satisfying the pentagon and triangle equations below.

$$\begin{array}{ccc} & A \otimes ((B \otimes C) \otimes D) & \\ \uparrow \mathbb{1} \otimes \alpha_{B,C,D} & & \searrow \alpha_{A,(B \otimes C),D} \\ A \otimes (B \otimes (C \otimes D)) & & (A \otimes (B \otimes C)) \otimes D \\ \downarrow \alpha_{A,B,(C \otimes D)} & & \downarrow \alpha_{A,B,C} \otimes \mathbb{1} \\ (A \otimes B) \otimes (C \otimes D) & \xrightarrow{\alpha_{(A \otimes B),C,D}} & ((A \otimes B) \otimes C) \otimes D \end{array} \quad \begin{array}{ccc} (A \otimes I) \otimes B & \xrightarrow{\alpha_{A,I,B}} & A \otimes (I \otimes B) \\ \downarrow \rho_A \otimes \mathbb{1} & & \downarrow \mathbb{1} \otimes \lambda_B \\ A \otimes B & & A \otimes B \end{array}$$

A monoidal category is *strict* if the coherence isomorphisms are identities.

Morphisms of monoidal categories are monoidal functors, which are functors that preserve the monoidal structure.

**Definition 2.2.** A (strong) monoidal functor  $F : (C, \otimes, I) \rightarrow (D, \boxtimes, J)$  between two monoidal categories is a functor  $F : C \rightarrow D$  between the underlying categories that respects the monoidal structure. This means that there are natural isomorphisms  $\varepsilon : J \rightarrow F(I)$  and  $\mu : F(-) \boxtimes F(-) \rightarrow F(-\otimes -)$  that are associative and unital.

$$\begin{array}{ccc} (F(A) \boxtimes F(B)) \boxtimes F(C) & \xrightarrow{\alpha_F} & F(A) \boxtimes (F(B) \boxtimes F(C)) \\ \downarrow \mu \boxtimes \mathbb{1} & & \downarrow \mathbb{1} \boxtimes \mu \\ (F(A \otimes B) \boxtimes F(C)) & & F(A) \boxtimes F(B \otimes C) \\ \downarrow \mu & & \downarrow \mu \\ (F((A \otimes B) \otimes C)) & \xrightarrow{F(\alpha)} & F(A \otimes (B \otimes C)) \end{array}$$

$$\begin{array}{ccc}
J \boxtimes \mathbf{F}(A) & \xrightarrow{\varepsilon \boxtimes \mathbb{1}} & \mathbf{F}(J) \boxtimes \mathbf{F}(A) \\
\downarrow \lambda_{\mathbf{F}} & & \downarrow \mu \\
\mathbf{F}(A) & \xleftarrow{\mathbf{F}(\lambda)} & \mathbf{F}(J \otimes A)
\end{array}
\qquad
\begin{array}{ccc}
\mathbf{F}(A) \boxtimes J & \xrightarrow{\varepsilon \boxtimes \mathbb{1}} & \mathbf{F}(A) \boxtimes \mathbf{F}(J) \\
\downarrow \rho_{\mathbf{F}} & & \downarrow \mu \\
\mathbf{F}(A) & \xleftarrow{\mathbf{F}(\rho)} & \mathbf{F}(A \otimes J)
\end{array}$$

Locally small monoidal categories and monoidal functors form a monoidal category  $\text{MonCat}$  with the Cartesian product and the one-object category.

*Example 2.3* (The monoidal category of hypergraphs). A *hypergraph*  $G = (V, E, \text{ends})$  is a set of vertices  $V$ , a set of edges  $E$  and a function  $\text{ends} : E \rightarrow \wp(V)^1$ . A morphism  $h : G \rightarrow H$  of graphs is a pair of functions  $h_V : V_G \rightarrow V_H$  and  $h_E : E_G \rightarrow E_H$  that preserve the adjacency relation:  $h_E \circ \text{ends}_G = \text{ends}_H \circ \wp(h_V)$ . Hypergraphs and their homomorphisms form a monoidal category  $\text{UHGraph}$  with the coproduct monoidal structure where the monoidal product is component-wise disjoint union and the monoidal unit is the empty graph.

The Coherence Theorem for monoidal categories [Mac78, Section VII.2] states that all well-typed equations between morphisms constructed only from  $\alpha, \lambda, \rho$  and the categorical and monoidal structure hold. A consequence of this result is the Strictification Theorem [Mac78, Section XI.3].

**Theorem 2.4** (Strictification [Mac78]). *Every monoidal category is monoidally equivalent to a strict one.*

The Coherence and Strictification Theorems allow us to forget about associators and unitors when showing equalities between morphisms.

*Remark 2.5.* Let  $\mathbf{C}$  be a monoidal category,  $\mathbf{S}$  be its strictification and let  $\mathbf{H} : \mathbf{C} \rightarrow \mathbf{S}$  and  $\mathbf{A} : \mathbf{S} \rightarrow \mathbf{C}$  be the strong monoidal functors giving the equivalence between them. The Coherence Theorem gives a unique natural isomorphism  $\phi_A : A \cong \mathbf{A}(\mathbf{H}(A))$ . For each morphism  $f : A \rightarrow B$  in  $\mathbf{C}$ , its image  $\mathbf{A}(\mathbf{H}(f))$  does not necessarily coincide with  $f$ , but  $f = \phi_A \circ \mathbf{A}(\mathbf{H}(f)) \circ \phi_B^{-1}$ . This means that, every time we show an equality  $u = v$  between morphisms  $u, v : X \rightarrow Y$  in the strictification  $\mathbf{S}$ , we can deduce that  $f = g$ , for all objects  $A$  and  $B$  and morphisms  $f, g : A \rightarrow B$  in  $\mathbf{C}$  such that  $\mathbf{H}(f) = u$  and  $\mathbf{H}(g) = v$ , because  $f = \phi_A \circ \mathbf{A}(u) \circ \phi_B^{-1} = \phi_A \circ \mathbf{A}(v) \circ \phi_B^{-1} = g$ . In particular, a syntax for strict monoidal categories gives a syntax for monoidal categories.

*Example 2.6* (The monoidal category of monoidal signatures). A *monoidal signature*  $\Sigma = E \rightrightarrows V^*$  is a set of types  $V$ , a set of generators  $E$ , and source and target functions  $s, t : E \rightarrow V^*$  that associate to each generator the types of its inputs and outputs. A monoidal signature is *one-sorted* if  $V$  contains only one element. A morphism  $h : \Sigma \rightarrow \Sigma'$  of monoidal signatures is a pair of functions  $h_V : V \rightarrow V'$  and  $h_E : E \rightarrow E'$  that preserve the inputs and outputs:  $h_E \circ s' = s \circ h_V^*$  and  $h_E \circ t' = t \circ h_V^*$ . Monoidal signatures and their morphisms form a monoidal category  $\text{MonSig}$  where monoidal product is disjoint union. This is the comma category  $(\mathbb{1} \downarrow \mathbf{L})$  for the identity functor and the functor  $\mathbf{L} : V \mapsto V^* \times V^*$ . One-sorted monoidal signatures form a full subcategory  $1\text{MonSig}$  of  $\text{MonSig}$ .

Given a monoidal signature  $\Sigma$ , a string diagram over  $\Sigma$  is obtained by composing sequentially or in parallel some of the generators in  $\Sigma$ . String diagrams are a convenient and formal syntax for monoidal categories. More precisely, there is an adjunction between the category  $\text{MonSig}$  of monoidal signatures and the category  $\text{MonCat}$  of monoidal categories, where the free monoidal category on a monoidal signature  $\Sigma$  is given by string diagrams on  $\Sigma$  [JS91, Theorem 1.2]. See Selinger's survey [Sel11] for an overview of string diagrammatic calculi.

**Theorem 2.7** ([JS91]). *String diagrams on a monoidal signature  $\Sigma$  form a strict monoidal category and, in fact, the free strict monoidal category on  $\Sigma$ .*

<sup>1</sup>We indicate with  $\wp$  the covariant powerset functor.

Symmetric monoidal categories are monoidal categories equipped with processes  $\bowtie$ , the symmetries, that permute the order of resources. This family of processes is compatible with the monoidal structure,

$$\begin{array}{c}
 A \otimes B \quad C \\
 \diagdown \quad \diagup \\
 \quad \quad A \otimes B \\
 \diagup \quad \diagdown \\
 C \quad A \otimes B
 \end{array}
 =
 \begin{array}{c}
 A \quad C \\
 \diagdown \quad \diagup \\
 \quad \quad A \\
 \diagup \quad \diagdown \\
 B \quad B
 \end{array}
 \text{ and }
 \begin{array}{c}
 A \quad B \otimes C \\
 \diagdown \quad \diagup \\
 \quad \quad A \\
 \diagup \quad \diagdown \\
 B \otimes C \quad A
 \end{array}
 =
 \begin{array}{c}
 A \quad C \\
 \diagdown \quad \diagup \\
 \quad \quad A \\
 \diagup \quad \diagdown \\
 B \quad B \\
 \quad \quad C
 \end{array}
 ,$$

it defines a natural transformation,

$$\begin{array}{c}
 A \quad D \\
 \diagdown \quad \diagup \\
 \quad \quad D \\
 \diagup \quad \diagdown \\
 C \quad B
 \end{array}
 =
 \begin{array}{c}
 A \quad D \\
 \diagdown \quad \diagup \\
 \quad \quad D \\
 \diagup \quad \diagdown \\
 C \quad B
 \end{array}
 ,$$

and it is an isomorphism,

$$\begin{array}{c}
 A \quad A \\
 \diagdown \quad \diagup \\
 \quad \quad A \\
 \diagup \quad \diagdown \\
 B \quad B
 \end{array}
 =
 \begin{array}{c}
 A \quad A \\
 \diagdown \quad \diagup \\
 \quad \quad A \\
 \diagup \quad \diagdown \\
 B \quad B
 \end{array}
 .$$

**Definition 2.8.** A *braided monoidal category* is a monoidal category  $(C, \otimes, I)$  with a natural isomorphism  $\sigma : (- \otimes -) \rightarrow (= \otimes -)$  that is compatible with the monoidal product.

$$\begin{array}{ccccc}
 A \otimes (B \otimes C) & \xrightarrow{\alpha} & (A \otimes B) \otimes C & & (A \otimes B) \otimes C & \xrightarrow{\alpha^{-1}} & A \otimes (B \otimes C) \\
 \downarrow \mathbb{1} \otimes \sigma_{B,C} & & \downarrow \sigma_{A \otimes B, C} & & \downarrow \sigma_{A,B} \otimes \mathbb{1} & & \downarrow \sigma_{A, B \otimes C} \\
 A \otimes (C \otimes B) & & C \otimes (A \otimes B) & & (B \otimes A) \otimes C & & (B \otimes C) \otimes A \\
 \downarrow \alpha & & \downarrow \alpha & & \downarrow \alpha^{-1} & & \downarrow \alpha^{-1} \\
 (A \otimes C) \otimes B & \xrightarrow{\sigma_{A,C} \otimes \mathbb{1}} & (C \otimes A) \otimes B & & B \otimes (A \otimes C) & \xrightarrow{\mathbb{1} \otimes \sigma_{A,C}} & B \otimes (C \otimes A)
 \end{array}$$

A braided monoidal category is *symmetric* if the inverse of  $\sigma_{A,B}$  is  $\sigma_{B,A}$ .

*Example 2.9.* The monoidal category of graphs is symmetric with the obvious isomorphism lifted from the category Set of sets and functions.

Symmetric monoidal functors are monoidal functors that preserve the symmetries.

**Definition 2.10.** A *braided monoidal functor*  $F : C \rightarrow D$  between braided monoidal categories  $(C, \otimes, I)$  and  $(D, \boxtimes, J)$  is a monoidal functor that respects the braiding.

$$\begin{array}{ccc}
 F(A) \boxtimes F(B) & \xrightarrow{\sigma_F} & F(B) \boxtimes F(A) \\
 \downarrow \mu & & \downarrow \mu \\
 F(A \otimes B) & \xrightarrow{F(\sigma)} & F(B \otimes A)
 \end{array}$$

A *symmetric monoidal functor* is a braided monoidal functor between symmetric monoidal categories. Locally small symmetric monoidal categories and symmetric monoidal functors form a symmetric monoidal category  $\text{SymMonCat}$ .

Coherence for symmetric monoidal categories [Mac78, Section XI.1] ensures that all well-typed equations between morphisms that have the same underlying permutation and are constructed only from  $\alpha, \lambda, \rho, \sigma$

and the categorical and monoidal structure hold. The strictification  $S$  of a symmetric monoidal category  $C$  is also symmetric and the symmetry on  $S$  is defined as

$$A \otimes B \cong \mathbf{H}(A(A \otimes B)) \xrightarrow{\mathbf{H}\mu^{-1}} \mathbf{H}(A(A) \otimes A(B)) \xrightarrow{\mathbf{H}\sigma_A} \mathbf{H}(A(B) \otimes A(A)) \xrightarrow{\mathbf{H}\mu} \mathbf{H}(A(B \otimes A)) \cong B \otimes A .$$

Given a monoidal signature  $\Sigma$ , a string diagram with symmetries over that signature is a string diagram over  $\Sigma$  where wires are allowed to be permuted. String diagrams with symmetries are a convenient and formal syntax for symmetric monoidal categories. More precisely, there is an adjunction between the category  $\text{MonSig}$  of monoidal signatures and the category  $\text{SymMonCat}$  of symmetric monoidal categories, where the free symmetric monoidal category on a monoidal signature  $\Sigma$  is given by string diagrams with symmetries over  $\Sigma$  [JS91, Theorem 2.3].

**Theorem 2.11** ([JS91]). *String diagrams with symmetries on a monoidal signature  $\Sigma$  form a symmetric strict monoidal category and, in fact, the free symmetric strict monoidal category on  $\Sigma$ .*

**Props and finitely presented props**

When a process theory only has one resource or there is no interest in recording the distinction between the resources, the only relevant information about the inputs and outputs of processes is their number. Props [Mac65] provide an algebra for these “untyped” process theories. They are symmetric strict monoidal categories where the objects are natural numbers and morphisms  $n \rightarrow m$  represent processes with  $n$  inputs and  $m$  outputs.

**Definition 2.12.** A prop is a symmetric strict monoidal category whose objects are natural numbers, the monoidal product on them is addition and monoidal unit is 0.

*Example 2.13.* The skeleton of the category  $\text{FinSet}$  of finite sets and functions is a prop.

**Definition 2.14.** A homomorphism of props is an identity-on-objects symmetric strict monoidal functor. Props and their homomorphisms form a category  $\text{Prop}$  that is a subcategory of  $\text{SymMonCat}$ .

Some props can be presented by a finite set of generators because the adjunction between monoidal signatures and symmetric monoidal categories restricts to an adjunction between the category of one-sorted monoidal signatures  $1\text{MonSig}$  and the category  $\text{Prop}$  of props. As a consequence, the morphisms of free props are one-sorted string diagrams with symmetries.

Some theories impose equations on their processes. For example, multiplying by the neutral element needs to return the input as it is, so the theory of commutative monoids, in Figure 2.1, is presented by two generators, the multiplication  $\text{D}^- : 2 \rightarrow 1$  and the unit  $\text{O}^- : 0 \rightarrow 1$ , subject to equations that ensure unitality, associativity and commutativity. Formally, this prop is a coequaliser: if we indicate with  $M_0$  the free prop on the generators  $\{\text{D}^-, \text{O}^-\}$ , with  $E$  the free prop on three generators  $\{u : 1 \rightarrow 1, a : 3 \rightarrow 1, c : 2 \rightarrow 1\}$ , and with  $\mathbf{l}, \mathbf{r} : E \rightarrow M_0$  the prop morphisms that point to the left- and right-hand sides of the three equations in Figure 2.1,



the prop that gives the theory of commutative monoids  $M$  is the coequaliser of  $\mathbf{l}$  and  $\mathbf{r}$  in Prop:

$$E \xrightarrow{\mathbf{l}, \mathbf{r}} M_0 \xrightarrow{\mathbf{q}} M.$$

*Example 2.15* ([Lac04]). The skeleton of the category  $\text{FinSet}$  of finite sets and functions is presented by a commutative monoid (Figure 2.1).

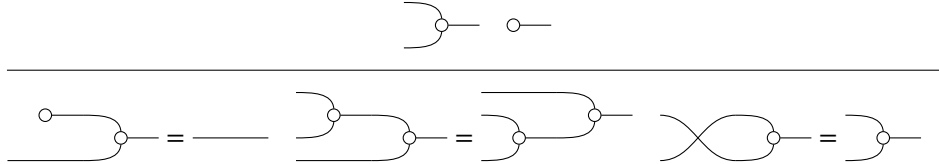


Figure 2.1: Generators and equations for a monoid.

Any prop  $S$  contains the initial prop  $P_0$  of permutations as subprop. This determines two prop morphisms  $\mathbf{l} : P_0 \otimes S \rightarrow S$  and  $\mathbf{r} : S \otimes P_0 \rightarrow S$  by pre- and post-composition because  $P_0 \cong P_0^{\text{op}}$ . The composition  $S \otimes_{P_0} T$  of two props  $S$  and  $T$  is the coequaliser of  $\mathbb{1}_S \otimes \mathbf{l}$  and  $\mathbf{r} \otimes \mathbb{1}_T$ .

$$S \otimes_{P_0} T \xrightarrow{\mathbb{1}_S \otimes \mathbf{l}, \mathbf{r} \otimes \mathbb{1}_T} S \otimes T \rightarrow S \otimes_{P_0} T$$

Composite props are characterised by factorisations of their morphisms. This result will be useful to show the syntactic presentations of the props in Section 4.3.

**Theorem 2.16** ([Lac04, Theorem 4.6]). *Let  $R, S$  and  $T$  be props with prop morphisms  $\iota_S : S \rightarrow R$  and  $\iota_T : T \rightarrow R$ . Suppose that any morphism  $r : m \rightarrow n$  in  $R$  can be written as a composition  $r = \iota_S(s) \circ \iota_T(t)$  for some  $s : m \rightarrow p$  in  $S$  and some  $t : p \rightarrow n$  in  $T$ , uniquely up to permutations  $\sigma : p \rightarrow p$ . Then,  $R$  is the composite of  $S$  and  $T$  via a distributive law  $\lambda : T \otimes_{P_0} S \rightarrow S \otimes_{P_0} T$  that associates to a pair  $(t \mid s)$  the pair  $(\hat{s} \mid \hat{t})$ , where  $\iota_S(\hat{s}) \circ \iota_T(\hat{t})$  is the unique factorisation of  $\iota_T(t) \circ \iota_S(s)$ .*

As explained in detail in Zanasi's PhD thesis [Zan15, Proposition 2.27], when composing finitely presented props, the distributive law  $\lambda$  gives the additional equations that determine the composite theory: for each pair  $(t \mid s)$  in  $T \otimes_{P_0} S$ , we add the equation  $\iota_T(t) \circ \iota_S(s) = \iota_S(\hat{s}) \circ \iota_T(\hat{t})$ . In other words, the composed prop  $S \otimes_{P_0} T$  is the coequaliser

$$T \otimes_{P_0} S \xrightarrow{\mathbf{l}, \mathbf{r}} S + T \rightarrow S \otimes_{P_0} T$$

of the prop morphisms  $\mathbf{l}$  and  $\mathbf{r}$  defined by

$$\mathbf{l}(t \mid s) := \iota_T(t) \circ \iota_S(s) \quad \text{and} \quad \mathbf{r}(t \mid s) := \iota_S(\hat{s}) \circ \iota_T(\hat{t}).$$

Coproducts of props are particular cases of prop compositions where the distributive law does not add any extra equation: the set of equations of the coproduct of two props is the disjoint union of the sets of equations of the components.

**Proposition 2.17** ([Zan15, Proposition 2.11]). *Let  $P_1$  and  $P_2$  be two props presented by generators and equations  $(\Sigma_1, E_1)$  and  $(\Sigma_2, E_2)$ . Then, their coproduct  $P_1 + P_2$  is presented by the disjoint union of the generators and equations of  $P_1$  and  $P_2$ ,  $(\Sigma_1 \sqcup \Sigma_2, E_1 \sqcup E_2)$ .*

## 2.2 Graph complexity measures

Several important applications of model checking reduce to deciding whether a formula  $\phi$  is true in a graph or, more generally, in a relational structure  $G$ .

$$G \models \phi$$

This problem is hard in general, even when the formula  $\phi$  is fixed. For example, for monadic second order logic and every level  $\Sigma_i^P$  of the polynomial hierarchy, there are formulae and classes of structures that make the model checking problem complete for  $\Sigma_i^P$  [MP96]. However, when the input graph is structurally simple, monadic second order formulae can be checked efficiently.

The structural complexity of graphs may be measured in different ways and different measures may define different classes of “simple” graphs. Tree width and clique width are some of the most famous measures of this kind: classes of graphs with bounded clique width might not have bounded tree width. This section recalls these graph complexity measures and two equivalent ones, while the next shows how they serve the design of efficient model checking algorithms.

All these graph widths rely on a corresponding notion of decomposition that indicates how graphs can be split in smaller subgraphs according to some specific rules. There may be different decompositions of the same graph and some of them may be more efficient than others. The width of a graph is the complexity of the most efficient decompositions.

We recall the definitions of graphs, hypergraphs and relational structures.

**Definition 2.18.** An *undirected (multi-)hypergraph*  $G = (V, E, \text{ends})$  is determined by a function  $\text{ends} : E \rightarrow \wp(V)$  that assigns to each edge  $e \in E$  a set of vertices  $\text{ends}(e) \subseteq V$ , the endpoints of  $e$ . An *undirected (multi-)graph* is a hypergraph where all the edges have at most two endpoints.

Note that, with this definition, edges in a hypergraph can have multiple endpoints or none, and there can be parallel edges between the same vertices.

Relational structures can be described as generalised hypergraphs where the vertices can be connected by different “types” of edges. A *relational signature* fixes a set of types for the edges.

**Definition 2.19.** A *relational signature* is a set  $\tau$  of relational symbols with a specified arity  $\alpha : \tau \rightarrow \mathbb{N}$ .

We will write finite relational signatures as sets of pairs  $\tau = \{(R_1, \alpha_1), \dots, (R_n, \alpha_n)\}$ , where  $\alpha_i := \alpha(R_i)$ .

*Example 2.20.* The relational signature for graphs contains a single relation of arity 2,  $\tau_{gr} = \{(E, 2)\}$ , that specifies which vertices are connected by an edge, while that for hypergraphs contains a relation for each arity  $n$ ,  $\tau_{hyp} = \{(E_n, n) : n \in \mathbb{N}\}$ , that specify which sets of vertices are connected by a hyperedge.

**Definition 2.21.** For a relational signature  $\tau$ , a *relational  $\tau$ -structure*  $G$  is a set  $V$  of vertices with an  $\alpha_R$ -ary relation  $R^G \subseteq V^{\alpha_R}$  for each relational symbol  $R$  of arity  $\alpha_R$  in the signature  $\tau$ .

*Example 2.22.* Graphs and hypergraphs can be encoded as relational structures for the signatures  $\tau_{gr}$  and  $\tau_{hyp}$  defined in Example 2.20. In principle, relational symbols are ordered, but we can restrict to unordered relational structures.

While we will work with relational structures, we focus on hypergraphs for defining graph decompositions. This distinction does not matter because tree and branch decompositions do not depend on the labels of the relational structures or on the order in which the vertices are related by a relational symbol. In fact, the tree and branch widths of a relational structure coincide with the tree and branch widths of its underlying hypergraph. We fix some graph theoretic nomenclature. Trees and, in particular, subcubic trees are part of the data of tree and branch decompositions.

**Definition 2.23.** Two distinct vertices  $v, w \in V$  are *neighbours* in a hypergraph  $G$  if they are both endpoints of the same edge  $e \in E$ ,  $v, w \in \text{ends}(e)$ . A *path* in  $G$  is a sequence of vertices  $(v_1, \dots, v_n)$  with a sequence of distinct hyperedges  $(e_1, \dots, e_{n-1})$  such that  $v_i, v_{i+1} \in \text{ends}(e_i)$  are both endpoints of the hyperedge  $e_i$ , for every  $i = 1, \dots, n-1$ . A *cycle* in  $G$  is a path where the first vertex  $v_1$  coincides with the last one  $v_n$ .

**Definition 2.24.** A hypergraph is *connected* if there is a path between any two vertices. A *tree* is a connected acyclic graph. A *subcubic tree* is a tree where every vertex has at most three neighbours. Vertices with one neighbour are the *leaves*.

### Tree width and branch width

Tree width and branch width are equivalent graph complexity measures which, intuitively, measure how tree-like a graph is. This section recalls tree width and branch width for undirected multi-hypergraphs, which we will simply call hypergraphs.

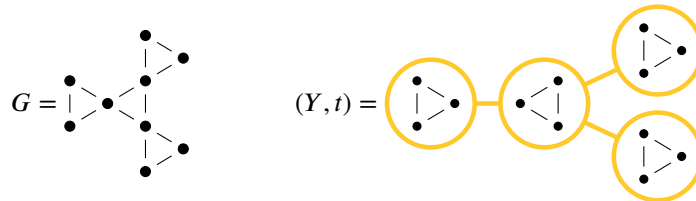
**Tree width.** Tree width, introduced by Robertson and Seymour [RS86, Section 1], measures the structural complexity of relational structures by comparing their structure to trees. In fact, forests have tree width 2, while the family of cliques has unbounded tree width. Tree width is based on tree decompositions, which specify a way of aggregating the vertices of a graph in a tree shape. This information is recorded in a tree whose nodes are labelled by sets of vertices in the graph, called *bags*. The conditions on the bags ensure that they respect the shape of the tree.

**Definition 2.25.** A *tree decomposition* of a hypergraph  $G = (V, E)$  is a pair  $(Y, t)$  of a tree  $Y$  and a function  $t : \text{vertices}(Y) \rightarrow \wp(V)$  such that:

1. Every vertex  $v$  is in at least one of the bags  $t(i)$ ,  $\bigcup_{i \in \text{vertices}(Y)} t(i) = V$ .
2. For every edge  $e \in E$  there is a node  $i \in \text{vertices}(Y)$  whose bag  $t(i)$  contains all the endpoints  $\text{ends}(e)$  of  $e$ .
3. The subgraphs induced by the bags are glued in a tree shape, i.e. the intersection of any two bags  $t(i)$  and  $t(k)$  is contained in all the bags  $t(j)$  corresponding to nodes  $j \in \text{vertices}(Y)$  that are on the path between  $i$  and  $k$  on the tree  $Y$ .

A tree decomposition of a relational  $\tau$ -structure is a tree decomposition of its underlying undirected hypergraph.

*Example 2.26.* A tree decomposition of a hypergraph  $G = (V, E)$  is a tree  $Y$  with a labelling  $t$  of its nodes. Every node  $i \in \text{vertices}(Y)$  induces the subgraph  $G[t(i)]$  of  $G$  on the bag  $t(i)$ . We draw the decomposition  $(Y, t)$  as a tree where the nodes are bubbles containing the subgraphs  $G[t(i)]$  of  $G$  induced by the bags  $t(i)$ .



The width of a tree decomposition  $(Y, t)$  of a graph  $G$  is the number of vertices in the biggest bag. Intuitively, it is the maximum number of vertices that need to be “hidden” in a bag to obtain a tree shape from the graph. The cost of the decomposition in Example 2.26 is 3 as all the bags contain three vertices. Different decompositions can have different widths, but the tree width of a graph is the width of a minimal one.



**Definition 2.27.** Given a tree decomposition  $(Y, t)$  of a graph  $G$ , its *width* is the maximum cardinality of its bags,  $\text{wd}(Y, t) := \max_{i \in \text{vertices}(Y)} |t(i)|$ . The *tree width* of  $G$  is given by the min-max formula:

$$\text{tw}(G) := \min_{(Y,t)} \text{wd}(Y, t).$$

Note that Robertson and Seymour subtract 1 from  $\text{tw}(G)$  so that trees have tree width 1. To minimise bureaucratic overhead, we ignore this and, according to this convention, trees and forests have tree width 2, while the clique on  $n$  vertices has tree width  $n$ .

*Remark 2.28.* The *Gaifman graph* of a hypergraph is the graph obtained by replacing every hyperedge with  $n$  endpoints by an  $n$ -clique. The tree width of a hypergraph is the same as the tree width of its Gaifman graph because the tree width of an  $n$ -clique and the tree width of a hypergraph on  $n$  vertices that are all connected by a single edge are both  $n$ .

**Branch width.** Branch width was introduced by Robertson and Seymour as alternative to tree width [RS91, Section 4]. While a tree decomposition splits a graph into subgraphs, a branch decomposition imposes that these subgraphs contain only one edge. Intuitively, this should not matter. In fact, the corresponding complexity measure, branch width, is equivalent to tree width.

**Definition 2.29.** The *hyperedge size* of a relational  $\tau$ -structure  $G$  is the maximum arity of the relations with non-empty interpretation:  $\gamma(G) := \max_{R \in \tau} \alpha_R$ . The *hyperedge size* of a relational signature  $\tau$  is the maximum arity of its symbols:  $\gamma(\tau) := \max_{R \in \tau} \alpha_R$ .

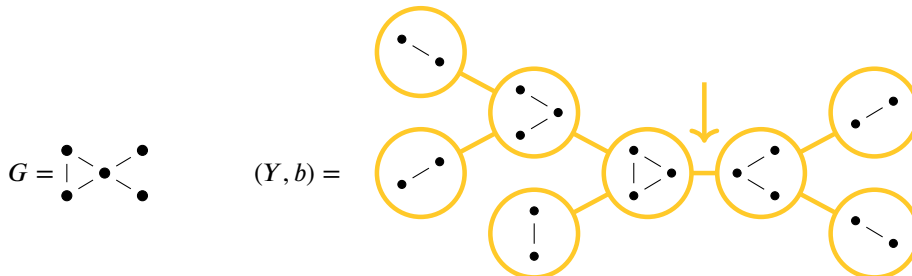
**Theorem 2.30** ([RS91, Theorem 5.1]). *Branch width is equivalent to tree width. More precisely, for a hypergraph  $G$ ,*

$$\max\{\text{bwd}(G), \gamma(G)\} \leq \text{tw}(G) \leq \max\left\{\frac{3}{2}\text{bwd}(G), \gamma(G), 1\right\}.$$

A branch decomposition is a tree where the leaves are in bijection with the edges of the graph. If this tree had a root, a branch decomposition would be a recipe for successively splitting the graph in two parts along its vertices until both parts contain only one edge.

**Definition 2.31.** A *branch decomposition* of a hypergraph  $G = (V, E)$  is a pair  $(Y, b)$  of a subcubic tree  $Y$  and a bijection  $b : \text{leaves}(Y) \cong E$  between the leaves of  $Y$  and the edges of  $G$ . A branch decomposition of a relational  $\tau$ -structure is a branch decomposition of its underlying hypergraph.

*Example 2.32.* If we choose an edge of  $Y$  to be the starting point of the decomposition, we can extend the labelling to the internal vertices of the tree by labelling them with the gluing of the labels of their children. In this way, a branch decomposition is a way of splitting a graph by cutting along its vertices.



Each splitting of the graph cuts along some vertices, as shown in Example 2.32 and each edge  $e$  in the tree  $Y$  determines a splitting of the graph. More precisely, it determines a 2-partition of the leaves of  $Y$ , which, through  $b$ , determines a 2-partition  $\{A_e, B_e\}$  of the edges of  $G$ . This corresponds to a splitting of the graph  $G$  into two subgraphs  $G_1$  and  $G_2$ . Intuitively, the order of an edge  $e$  is the number of vertices that  $G_1$  and  $G_2$  have in common as subgraphs of  $G$ . Given the partition  $\{A_e, B_e\}$  of the edges of  $G$ , we say that a vertex  $v$  of  $G$  *separates*  $A_e$  and  $B_e$  whenever there are an edge in  $x \in A_e$  and an edge in  $y \in B_e$  that are both adjacent to  $v$ :  $v \in \text{ends}(A_e) \cap \text{ends}(B_e)$ .

**Definition 2.33.** The *order* of an edge  $e$  in a branch decomposition  $(Y, b)$  of a hypergraph  $G$  is the number of vertices that separate  $A_e$  and  $B_e$ :  $\text{ord}(e) := |\text{ends}(A_e) \cap \text{ends}(B_e)|$ .

In Example 2.32, there is only one vertex separating the first two subgraphs of the decomposition. This means that the corresponding edge in the decomposition tree has order 1. The width of a decomposition is the maximum number of vertices in all cuts. The branch width of a graph is the width of a most efficient decomposition.

**Definition 2.34.** The *width* of a branch decomposition  $(Y, b)$  of a hypergraph  $G = (V, E)$  is the maximum order of its edges,  $\text{wd}(Y, b) := \max_{e \in \text{Edges}(Y)} \text{ord}(e)$ . The *branch width* of a hypergraph  $G$  is given by the min-max formula:

$$\text{bwd}(G) := \min_{(Y, b)} \text{wd}(Y, b).$$

### Clique width and rank width

Clique width and rank width are equivalent graph complexity measures that are “stronger” than tree width and branch width: every graph of bounded tree width has bounded clique width but vice-versa is not true. This section recalls clique width and rank width for undirected multi-graphs.

**Clique width.** In the same way that trees are simple according to tree width, cliques, and cographs more generally, are simple according to clique width. Clique decompositions, introduced by Courcelle, Engelfriet and Rozenberg [CER93; CO00], have a more algebraic flavour compared to the combinatorial definitions of tree and branch decompositions. They are terms formed by some operations and constants that specify a graph where the vertices have labels. The operations can rename the labels, create edges and take the disjoint union of graphs. The constants create a single 1-labelled vertex or the empty graph.

**Definition 2.35.** An  $n$ -labelled graph  $(G, l)$  is a graph  $G = (E, V)$  with a labelling function  $l : V \rightarrow \{1, \dots, n\}$ .

- The generating graphs are the 1-labelled empty graph,  $\emptyset_1$ , and the graph  $v_1$  with a single 1-labelled vertex.
- The *renaming* of labels  $\text{Rename}_{i \rightarrow j}^n$  of an  $n$ -labelled graph  $(G, l)$  is the graph  $(G, l')$ , where the vertices with label  $i$  now have label  $j$ :  $l'(v) = l(v)$  if  $l(v) \neq i$  and  $l'(v) = j$  if  $l(v) = i$ .
- The *edge creation*  $\text{Edge}_{i, j}^n$  of an  $n$ -labelled graph  $(G, l)$  is the  $n$ -labelled graph  $(G', l)$  with extra edges between the vertices with label  $i$  and those with label  $j$ .
- The *disjoint union*  $+$  of an  $n$ -labelled graph  $(G, l)$  and an  $n'$ -labelled graph  $(G', l')$  is the  $n + n'$ -labelled graph  $(G + G', l + l')$  given by the disjoint union of graphs and their labelling functions. Note that the labelling function  $l + l'$  reindexes the labels of  $G'$ :  $l + l'(v') := n + l'(v')$  for a vertex  $v'$  of  $G'$ , while  $l + l'(v) := l(v)$  for a vertex  $v$  of  $G$ .

Our treatment of labels slightly differs from the one in [CO00] but equivalent to it up to renaming of labels, and it is closer to the categorical algebra that we will introduce in Section 4.3. To be precise, we should define separately the syntactic operations and their semantics, but, for brevity, we presented them together.

In Section 4.3, we will derive these operations from compositions and monoidal product in a monoidal category where graphs are morphisms. There, the difference between the syntactic operations and the semantic ones is clear: they belong to different, but equivalent, monoidal categories.

A clique decomposition is a syntax tree where the internal nodes are the operations and the leaves are the constants in Definition 2.35.

**Definition 2.36.** A *clique decomposition*  $t \in T_G$  of a graph  $G$  is a term constructed with the operations and constants in Definition 2.35.

$$\begin{array}{ll}
 t & ::= (G) & \text{if } G = \emptyset_1 \text{ or } G = v_1 \\
 & | \text{Rename}^n_{i \rightarrow j}(t') & \text{if } G = \text{Rename}^n_{i \rightarrow j}(G') \text{ and } t' \in T_{G'} \\
 & | \text{Edge}^n_{i,j}(t') & \text{if } G = \text{Edge}^n_{i,j}(G') \text{ and } t' \in T_{G'} \\
 & | t_1 + t_2 & \text{if } G = G_1 + G_2 \text{ and } t_i \in T_{G_i}
 \end{array}$$

*Example 2.37.* The 1-labelled 4-clique is expressed by the term

$$\text{Rename}^2_{2 \rightarrow 1} \text{Edge}^2_{1,2}(\text{Rename}^3_{3 \rightarrow 2} \text{Edge}^3_{2,3}(\text{Rename}^4_{4 \rightarrow 3} \text{Edge}^4_{3,4}(v_1 + v_1 + v_1 + v_1))),$$

that creates 4 vertices and progressively adds edges between them, or by the (simpler) term

$$\text{Rename}^2_{2 \rightarrow 1} \text{Edge}^2_{1,2}(v_1 + \text{Rename}^2_{2 \rightarrow 1} \text{Edge}^2_{1,2}(v_1 + \text{Rename}^2_{2 \rightarrow 1} \text{Edge}^2_{1,2}(v_1 + v_1))),$$

that creates one vertex at a time and adds the edges between each new vertex and all the previous ones.

Assigning a cost to each operation inductively determines a cost for decompositions. The cost of an operation is, intuitively, the number of labels that it needs to handle.

**Definition 2.38.** We assign a cost to each operation,  $w(\text{Rename}^n_{i \rightarrow j}) := n$ ,  $w(\text{Edge}^n_{i,j}) := n$  and  $w(+):= 0$ . The *width* of a clique decomposition  $t$  of  $G$  is the maximum cost of its operations.

$$\begin{array}{ll}
 wd(t) & ::= |V_G| & \text{if } t = (G) \\
 & | \max\{n, wd(t')\} & \text{if } t = \text{Edge}^n_{i,j}(t') \text{ or } t = \text{Rename}^n_{i \rightarrow j}(t') \\
 & | \max\{wd(t_1), wd(t_2)\} & \text{if } t = t_1 + t_2
 \end{array}$$

The *clique width* of a graph  $G$  is the width of a best clique decomposition:

$$clwd(G) := \min_{t \in T_G} wd(t).$$

As with the other graph widths, the clique width of a graph is the cost of a cheapest decomposition. The first term in Example 2.37 costs 4, while the second costs 2 and gives a cheapest decomposition. In fact, in general, cliques (and cographs) have clique width 2, trees have clique width at most 3 [CO00], while  $n$ -grids have clique width  $n + 1$  [GRO0].

**Rank width.** Rank width and rank decompositions were introduced by Oum and Seymour to approximate clique width [Oum05; OS06]. In fact, the two measures are equivalent.

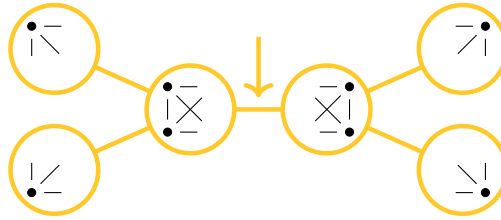
**Theorem 2.39** ([OS06, Proposition 6.3]). *Rank width is equivalent to clique width. More precisely, for a graph  $G$ ,*

$$rwd(G) \leq clwd(G) \leq 2^{rwd(G)+1} - 1.$$

Rank decompositions are similar in spirit to branch decompositions, but, instead of partitioning the edges of a graph, they partition their vertices. A rank decomposition of a graph  $G = (V, E)$  is a tree where the leaves are in bijection with the vertices  $V$  of the graph. If this tree had a root, a rank decomposition would be a recipe for successively splitting the graph in two parts along its edges until both parts contain only one vertex.

**Definition 2.40.** A rank decomposition of a graph  $G$  is a pair  $(Y, r)$  of a subcubic tree  $Y$  and a bijection  $r : \text{leaves}(Y) \rightarrow \text{vertices}(G)$ .

*Example 2.41.* While a branch decomposition cuts a graph along its vertices (Example 2.32), a rank decomposition is, intuitively, a recipe for decomposing a graph into its single-vertex subgraphs by cutting along its edges.



The cost of each cut is given by the rank of the adjacency matrix that represents it. The matrix below corresponds to the cut in the decomposition above indicated by the arrow.

$$\begin{array}{cc}
 \bullet & & \bullet \\
 | & & | \\
 \bullet & & \bullet \\
 | & & | \\
 \bullet & & \bullet
 \end{array}
 \quad
 \text{rk} \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = 1$$

Each edge  $b$  in the tree  $Y$  determines a splitting of the graph: it determines a two partition of the leaves of  $Y$ , which, through  $r$ , determines a two partition  $\{A_b, B_b\}$  of the vertices of  $G$ . This corresponds to a splitting of the graph  $G$  into two subgraphs  $G_1$  and  $G_2$ . Intuitively, the order of an edge  $b$  is the amount of information required to recover  $G$  by joining  $G_1$  and  $G_2$ . Given the partition  $\{A_b, B_b\}$  of the vertices of  $G$ , we can record the edges in  $G$  between  $A_b$  and  $B_b$  in a matrix  $X_b$ . This means that, if  $v_i \in A_b$  and  $v_j \in B_b$ , the entry  $(i, j)$  of the matrix  $X_b$  is the number of edges between  $v_i$  and  $v_j$ . The order of an edge  $b$  is the rank of its corresponding matrix  $X_b$ .

**Definition 2.42.** The order of  $b$  is the rank of the matrix  $X_b$  of the cut corresponding to  $b$ :  $\text{ord}(b) := \text{rk}(X_b)$ .

The cut shown in Example 2.41 corresponds to the edge indicated by the arrow. The order of this edge is 1, which is the rank of the matrix recording the cut. The width of a decomposition is the maximal edge order, and the rank width is the width of the most efficient decomposition. The complete graph on 4 vertices has rank width 1 with minimal decomposition shown in Example 2.41.

**Definition 2.43.** The width of a rank decomposition  $(Y, r)$  of a graph  $G$  is the maximum order of its edges,  $\text{wd}(Y, r) := \max_{b \in \text{edges}(Y)} \text{ord}(b)$ . The rank width of a graph  $G$  is given by the min-max formula:

$$\text{rwd}(G) := \min_{(Y,r)} \text{wd}(Y, r) .$$

The decomposition in Example 2.41 shows that the 4-clique has rank width 1. This holds for  $n$ -cliques in general and they all have rank width 1. As for clique width, the class of grids have unbounded rank width because the  $n$ -grid has rank width  $n - 1$  [Jel10].

## 2.3 Divide-and-conquer algorithms

Divide-and-conquer algorithms rely on the possibility of splitting, or *decomposing*, their inputs into smaller parts according to a given set of operations. The decompositions defined in this section resemble clique decompositions in that they are terms expressing a certain algebraic structure. The advantage of having a term expressing an input structure is that they give a divide-and-conquer algorithm: a “brute-force” algorithm runs on the generating structures to compute partial solutions and these partial solutions are combined according to the decomposition structure. When combining partial solutions is computationally easy and the term expression for the input structure is simple, the divide-and-conquer algorithm is efficient. Problems solved by such divide-and-conquer algorithms are *fixed-parameter tractable*: they can be quickly solved if the term that expresses the input structure has bounded complexity.

This section presents a general technique [CM02; Mak04] for finding divide-and-conquer algorithms for checking formulae of a chosen logic on relational structures and how to apply it to the case of monadic second order logic to obtain Courcelle’s theorems for tree width [Cou92a] and clique width [CO00].

### Checking formulae on relational structures

The problem of checking formulae of a given logic on relational structures is fixed-parameter tractable under two conditions.

1. There is a finite set of generating structures and, for each  $k \in \mathbb{N}$ , a finite set of operations  $\mathcal{O}_k$  to express relational structures of width at most  $k$ .
2. A preservation theorem holds for the chosen operations.

Given some operations and some generating structures, the well-formed terms express relational structures (Definitions 2.47 and 2.48). With Requirement 1, we define the *width* of a relational structure, which is the fixed parameter for the divide-and-conquer algorithm (Definition 2.49). Requirement 2, on the other hand, ensures that the theory of a composite structure can be computed from the theory of the component structures as in Definition 2.46. Assembling the theories of the components amounts to looking up the entries of a table (Definition 2.51) and evaluating a boolean function (Definition 2.45). These tasks do not depend on the relational structure but only on the given logic and can be restricted to check one given formula on the composite structure instead of computing its whole theory. The computation of the look-up table depends on the width of terms and on the initial formula.

Under these conditions running the divide-and-conquer algorithm for a fixed formula depends linearly on the size of the input term but more than exponentially on the width parameter, and the problem of checking a formula on a term for a relational structure is fixed-parameter tractable with parameter the width of input terms (Theorem 2.52).

For the rest of this section, we fix a logic  $\mathcal{L}$ , and consider the class of formulae  $\mathcal{L}(\tau)$  of all those sentences that can be written in  $\mathcal{L}$  using the relational symbols in  $\tau$ . We will write  $\mathcal{L}(\tau, \underline{x})$  for the set of formulae that can be written in  $\mathcal{L}$  using the relational symbols in  $\tau$  and with free variables in  $\underline{x}$ . The theory of a relational structure  $G$  in a logic  $\mathcal{L}$  is the set of sentences in  $\mathcal{L}(\tau)$  that are true in  $G$ .

**Definition 2.44.** For a relational signature  $\tau$ , the *theory* of a set of relational  $\tau$ -structures  $\mathcal{K}$  in a logic  $\mathcal{L}$  is the set of sentences in the logic  $\mathcal{L}(\tau)$  that is true in every  $\tau$ -structure  $G \in \mathcal{K}$ :  $\text{Th}_{\mathcal{L}(\tau)}(\mathcal{K}) := \{\phi \in \mathcal{L}(\tau) : \forall G \in \mathcal{K} G \models \phi\}$ . When the set contains only one structure, we write  $\text{Th}_{\mathcal{L}(\tau)}(G) := \text{Th}_{\mathcal{L}(\tau)}(\{G\})$ .

Given an operation  $o$  and a formula, it is sometimes possible to compute a sequence of formulae, called their *reduction sequence*, whose truth values on components  $G_1, \dots, G_n$  determine the truth value of the original formula on the composite structure  $o(G_1, \dots, G_n)$ .

**Definition 2.45.** For an  $n$ -ary operation  $o$  on  $\tau$ -structures, an  $o$ -*reduction sequence* for a formula  $\phi \in \mathcal{L}(\tau)$  is two pieces of data.

- A list  $(\psi_i^j \mid i = 1, \dots, m, j = 1, \dots, n)$  of formulae  $\psi_i^j \in \mathcal{L}(\tau)$ .
- A boolean function  $b : 2^{m \cdot n} \rightarrow 2$ .

For all  $\tau$ -structures  $G_1, \dots, G_n$ , the values of the formulae  $\psi_i^j$  on  $G_1, \dots, G_n$  and the function  $b$  need to determine the value of  $\phi$  on  $o(G_1, \dots, G_n)$ .

$$o(G_1, \dots, G_n) \models \phi \quad \text{iff} \quad b((G_j \models \psi_i^j)_{i,j}) = 1$$

When the formula  $\phi$  and the operation  $o$  need to be explicit, we will denote the list of formulae  $\psi_i^j$  by  $\text{RedSeq}(\phi, o)$  and the boolean function  $b$  by  $\text{RedBool}(\phi, o)$ .

The operations that always admit reduction sequences are *effectively smooth* [Mak04, Definition 4.1]. For these operations, the theory of a composite structure only depends on the theories of its components. This is a fundamental requirement for the correctness of divide-and-conquer algorithms and corresponds to Requirement 2.

**Definition 2.46.** An  $n$ -ary operation  $o$  is  $\mathcal{L}$ -smooth if the theory  $\text{Th}_{\mathcal{L}(\tau)}(o(G_1, \dots, G_n))$  of the  $\tau$ -structure  $o(G_1, \dots, G_n)$  in  $\mathcal{L}(\tau)$  depends only on  $\text{Th}_{\mathcal{L}(\tau)}(G_1), \dots, \text{Th}_{\mathcal{L}(\tau)}(G_n)$ , for all  $\tau$ -structures  $G_1, \dots, G_n$ . The operation  $o$  is *effectively  $\mathcal{L}$ -smooth* if, for every formula  $\phi \in \mathcal{L}(\tau)$ , there is an algorithm to compute the reduction sequence  $\text{RedSeq}(\phi, o)$  and its associated formula  $\text{RedBool}(\phi, o)$ .

Preservation theorems are the results that show that an operation is  $\mathcal{L}$ -smooth. For first order logic, there are preservation theorems with products and sums of relational structures [Mos52; FV59], while for monadic second order logic, they only hold for sum-like operations [FV59; Fef57; CK09]. Theorems 2.59 and 2.60 in the next section recall these results for the operations in Definitions 2.53 and 2.56.

A finite set of relational structures and a set of operations generate inductively a class of relational structures. Inductive classes are classes of relational structures obtained in this way with a finite set of smooth operations [Mak04, Definition 4.3]. The sets of operations defined in the next section (Definitions 2.53 and 2.56) are infinite, but they are indexed by natural numbers and finite for every fixed index. For every natural number  $k \in \mathbb{N}$ , there is a class of structures generated by the operations with index  $k$ . These are the structures of width at most  $k$ . Classes of relational structure of bounded width are inductive.

**Definition 2.47.** A class  $\mathcal{K}$  of  $\tau$ -structures is  $\mathcal{L}$ -inductive if there are

- a finite *generating* set  $\mathcal{K}_0 \subseteq \mathcal{K}$  of  $\tau$ -structures and
- a finite set  $\mathcal{O}$  of  $\mathcal{L}$ -smooth operations

such that  $\mathcal{K} = \bigcup_{n \in \mathbb{N}} \mathcal{K}_n$ , where  $\mathcal{K}_{n+1} := \{G \text{ } \tau\text{-structure} : \exists G_1, \dots, G_k \in \mathcal{K}_n \exists o \in \mathcal{O} G = o(G_1, \dots, G_k)\}$  is the set of all the  $\tau$ -structures  $A$  that are obtained by applying an operation  $o \in \mathcal{O}$  to some  $\tau$ -structures  $G_1, \dots, G_k \in \mathcal{K}_n$ . The class  $\mathcal{K}$  is *effectively  $\mathcal{L}$ -inductive* if all the operations in  $\mathcal{O}$  are effectively  $\mathcal{L}$ -smooth.

The terms that specify relational structures in terms of operations and generating structures are *decompositions*.

**Definition 2.48.** For an  $\mathcal{L}$ -inductive class  $\mathcal{K}$  of  $\tau$ -structures, an *algebraic decomposition* of a  $\tau$ -structure  $G \in \mathcal{K}$  is a term  $t \in T_G$  constructed from applications of operations  $o \in \mathcal{O}$  to  $\tau$ -structures  $G_0$  in the generating set  $\mathcal{K}_0$ :

$$\begin{array}{ll} t & ::= (G) & \text{if } G \in \mathcal{K}_0, \\ & \mid o(t_1, \dots, t_k) & \text{if } t_i \in T_{G_i} \text{ and } G = o(G_1, \dots, G_k) \text{ with } o \in \mathcal{O} \text{ of arity } k. \end{array}$$

If a decomposition combines the structures  $G_1, \dots, G_l$  with operations  $o_1, \dots, o_n$ , its width is given by the maximum cost,  $\max_{i,j} \{w(o_i), |V_j|\}$ , where we fixed costs  $w(o_i)$  for operations  $o_i$ .

**Definition 2.49.** A *weight function* for an  $\mathcal{L}$ -inductive class of  $\tau$ -structures is a function  $w : \mathcal{O} \rightarrow \mathbb{N}$ . A choice of weight function determines a *width* for algebraic decompositions:

$$\begin{aligned} \text{wd}(t) &:= |V_G| && \text{if } t = (G), \\ &| \max\{w(o), \text{wd}(t_1), \dots, \text{wd}(t_k)\} && \text{if } t = o(t_1, \dots, t_k). \end{aligned}$$

The *size* of a decomposition is the number of its leaves:

$$\begin{aligned} \text{size}(t) &:= 1 && \text{if } t = (G), \\ &| \text{size}(t_1) + \dots + \text{size}(t_k) && \text{if } t = o(t_1, \dots, t_k). \end{aligned}$$

The *algebraic width* of a  $\tau$ -structure  $G$  is the width of a best decomposition:

$$\text{awd}(G) := \min_{t \in T_G} \text{wd}(t).$$

For the sets of operations defined in the next section, Theorems 2.55 and 2.58 characterise the bounded-width classes of relational structures. The size of the decompositions corresponding to these operations is bounded by the number of hyperedges or vertices in the relational structure.

Given a set of effectively smooth operations and a formula, we can compute the set of all reduction sequences generated by the formula and arrange them in a look-up table. The general divide-and-conquer strategy uses this look-up table for combining the partial solutions.

**Definition 2.50.** The  $\mathcal{O}$ -*reduction set*  $\text{Red}(\phi, \mathcal{O})$  of a formula  $\phi \in \mathcal{L}(\tau, \underline{x})$  with respect to a finite set  $\mathcal{O}$  of  $\mathcal{L}$ -smooth operations is the smallest set of formulas in  $\mathcal{L}(\tau, \underline{x})$  that

- contains  $\phi$  and
- is closed under taking  $o$ -reduction sequences  $\text{RedSeq}(-, o)$ , for all operations  $o \in \mathcal{O}$ .

**Definition 2.51.** For a finite set  $\mathcal{O}$  of effectively  $\mathcal{L}$ -smooth operations and a formula  $\phi \in \mathcal{L}(\tau)$ , the *look-up table* of  $\phi$  and  $\mathcal{O}$  is a list

$$\text{Look}(\phi, \mathcal{O}) := (\psi, o, \text{RedSeq}(\psi, o), \text{RedBool}(\psi, o) \mid \psi \in \text{Red}(\phi, \mathcal{O}), o \in \mathcal{O}).$$

When the look-up table  $\text{Look}(\phi, \mathcal{O})$  is finite and the operations are effectively smooth, the table can be computed in finite time. We assume that the logic always gives finite look-up tables, which is true for monadic second order logic [Mak04, Observation 6]. Look-up tables give a way of combining partial solutions and showing fixed-parameter tractability of checking  $\mathcal{L}(\tau)$ -formulae on relational structures [CM02]. The proof that Makowsky presents [Mak04, Theorem 4.21] precomputes all the possible partial solutions, but these can also be computed as needed.

**Theorem 2.52** ([CM02]). *Fix a formula  $\phi \in \mathcal{L}(\tau)$  and an effectively  $\mathcal{L}$ -inductive class  $\mathcal{K}$  of  $\tau$ -structures with respect to a finite set  $\mathcal{O}$  of effectively  $\mathcal{L}$ -smooth operations. Let  $G \in \mathcal{K}$  be a  $\tau$ -structure with a parse term  $d$ . Then, whether the formula  $\phi$  holds in  $G$ ,  $G \models \phi$ , can be decided in time linear in  $\text{size}(d)$ .*

*Proof.* We precompute the look-up table  $\text{Look}(\phi, \mathcal{O})$  in finite time and this computation does not depend on the input structure but only on the fixed formula and operations. Using this look-up table, we run  $\text{Check}(d, \phi)$  (Algorithm 1). This computes  $G_i \models \psi_i^j$  for all the leaves  $G_i$  of the input decomposition  $d$  and combines these partial solutions by looking up on the table  $\text{Look}(\phi, \mathcal{O})$ . Looking up the information in  $\text{Look}(\phi, \mathcal{O})$  takes constant time  $c_0$ , while computing  $G_i \models \psi_i^j$  on a substructure  $G_i$  of size  $n_i$  takes time  $c(n_i)$ , for some more than exponential function  $c : \mathbb{N} \rightarrow \mathbb{N}$ . If the size of the decomposition is  $n$  and the maximum size of the substructures  $G_i$  is  $k$ , the computation takes  $\mathcal{O}(c(k) \cdot n)$ .  $\square$

**Algorithm 1:** Check( $d, \phi$ )

---

**Data:** a term  $d$  for a structure  $G$  and a formula  $\phi$   
**Result:** whether the structure  $G$  satisfies  $\phi$   
**if**  $d = (G)$  **then**  
  | compute  $t := G \models \phi$  by brute force  
**else if**  $d = o(d_1, \dots, d_n)$  for some  $o \in \mathcal{O}$  **then**  
  | look up the reduction sequence  $(\psi_i^j)_{i=1, \dots, m}^{j=1, \dots, n} := \text{RedSeq}(\phi, o)$   
  | look up the function  $b := \text{RedBool}(\phi, o)$   
  **for**  $i = 1, \dots, m$  **and**  $j = 1, \dots, n$  **do**  
    | compute  $t_i^j := \text{Check}(d_i, \psi_i^j)$   
  **end**  
  compute  $t := b((t_i^j)_{i,j})$   
**return**  $t$

---

**Monadic second order logic of graphs**

Monadic second order (MSO) logic is the fragment of second order logic where quantification is only allowed on unary predicates, i.e. sets of variables. This section recalls Courcelle's theorems for tree width and clique width [Cou92a; CO00] in Corollaries 2.63 and 2.64. Their proof strategy relies on showing the assumptions for applying Theorem 2.52:

- (1) Definitions 2.53 and 2.56 recall the decomposition algebras for tree width introduced by Bauderon and Courcelle [BC87; Cou90] and for rank width introduced by Courcelle and Kanté [CK09], and Theorems 2.55 and 2.58 recall that their MSO-inductive classes are those of bounded tree width [Cou92a] and rank width [CK07].
- (2) Theorems 2.59 and 2.60 recall the Feferman-Vaught-Mostowski [FV59; Fef57] and the Courcelle-Kanté [CK09] preservation theorems.

The operations for tree width and rank width join two structures by merging some of their parts. Structures are given an additional piece of information to specify which parts are allowed to be merged with other structures. These are called *constants* for the tree width decomposition algebra and *labels* for the rank width decomposition algebra.

**Definition 2.53.** A relational  $\tau$ -structure *with  $n$  constants*, for a natural number  $n \in \mathbb{N}$ , is a pair  $(G, c)$  of a structure  $G$  together with a function  $c : \{1, \dots, n\} \rightarrow V$ .

- The generating structures are the empty structure with no constants,  $\emptyset$ , and, for every relational symbol  $R \in \tau$ , the structure  $e_R$  with  $\alpha_R$  vertices that are all related by  $R$  and are all constants.
- The *disjoint union* of relational structures  $(G, c)$  and  $(H, d)$  with  $m$  and  $n$  constants is a relational structure  $(G + H, c + d)$  with  $m + n$  constants and universe  $A + B$ , where the relations are interpreted as disjoint unions:  $R^{G+H} = R^G \sqcup R^H$ .
- The *redefinition of constants*  $\text{Relab}_f^n$  of a relational structure  $(G, c)$  with  $n$  constants with a function  $f : \{1, \dots, m\} \rightarrow \{1, \dots, n\}$ , is the relational structure  $(G, fc)$  with  $m$  constants.
- The *fusion of constants*  $i$  and  $j$ , with  $0 < i < j \leq n+1$ , on a relational structure  $(G, c)$  with  $n+1$  constants gives a relational structure  $\text{Fuse}_{i,j}^n(G, c)$  with  $n$  constants where:
  - The universe  $V /_{c(i)=c(j)}$  is the set  $V$  quotiented by the equivalence relation  $c(i) = c(j)$ ;
  - The interpretation  $R^{\text{Fuse}_{i,j}^n(G)}$  of the relation  $R$  is the subset of  $V /_{c(i)=c(j)}$  that corresponds to the subset  $R^G$  quotiented by  $c(i) = c(j)$ ;



- The function for the constants  $\text{Fuse}_{i,j}^n c$  is the reindexing of the function  $c$  as  $\text{Fuse}_{i,j}^n c(k) = c(k)$  if  $k < j$  and  $\text{Fuse}_{i,j}^n c(k) = c(k+1)$  if  $k \geq j$ .

- The *addition of constant  $i$* ,  $\text{Vert}_i^n$ , on a relational structure  $(G, c)$  with  $n \geq i - 1$  constants is the relational structure  $(G + \{v\}, c')$  with  $n + 1$  constants  $c' : n + 1 \rightarrow V + \{v\}$  defined as  $c'(j) = c(j)$  for  $j < i$ ,  $c'(i) = v$  and  $c'(j) = c(j - 1)$  for  $j \geq i$ .

We assign a cost to these operations,  $w(+):= 0$ ,  $w(\text{Relab}_f^n):= n$ ,  $w(\text{Fuse}_{i,j}^n):= n$  and  $w(\text{Vert}_i^n):= n$ , and obtain a corresponding notion of width for decompositions.

*Example 2.54.* The graph formed by two 3-cliques joined along a vertex



is expressed by the term  $\text{Fuse}_{1,2}^2(t + t)$ , where  $t$  is a term for the 3-clique with one constant:

$$t = \text{Fuse}_{1,2}^2 \text{Relab}_i^3 \text{Fuse}_{2,3}^4 (\mathbf{e} + \text{Relab}_i^3 \text{Fuse}_{2,3}^4 (\mathbf{e} + \mathbf{e})),$$

that creates an edge at a time and joins its endpoints with the existing edges. The function  $\iota : \{1, 2\} \rightarrow \{1, 2, 3\}$  indicates the inclusion of the set with two elements into the set with three elements.

The operations for relational structures recalled above are slightly different from the original ones [Cou90], but define the same complexity measure [CM02; Mak04] and are more similar to the categorical algebra that we will introduce in Section 4.1. They define a graph width that is equivalent to tree width [Cou92a] and, as a consequence of Theorem 2.30, to branch width as well.

**Theorem 2.55** ([Cou92a, Theorem 2.2]). *For a relational  $\tau$ -structure  $(G, c)$  with constants, the algebraic width given by the operations of disjoint union and fusion of constants (Definition 2.53) is linearly related to its tree width:*

$$\text{twd}(G) \leq \text{awd}(G, c) \leq \max\{2 \cdot \text{twd}(G), \text{twd}(G) + \gamma(\tau), \gamma(G)\}.$$

Rank width and clique width are defined for graphs and so they are the operations that characterise them [CK07]. These are defined for graphs where the vertices can have multiple labels and these labels can be linearly modified.

**Definition 2.56.** An  $n$ -labelled graph  $(G, B)$  is a graph  $G$  on  $k$  vertices with a matrix  $B \in \text{Mat}_2(k, n)$  assigning to each vertex some of the labels  $\{1, \dots, n\}$ .

- The generating structures are the empty 1-coloured graph,  $\emptyset_1$ , and graph  $v_1$  with a single 1-coloured vertex.
- The *linear recolouring*  $\text{Recol}_M$  of an  $n$ -labelled graph  $(G, B)$  by an  $n$  by  $m$  matrix  $M \in \text{Mat}_2(n, m)$  is the  $m$ -labelled graph  $(G, B \cdot M)$ , where the colours have been modified by the matrix  $M$ .
- The *bilinear product*  $+_{M,P,N}$  of two labelled graphs,  $(G, B)$  with  $m$  labels and  $(H, C)$  with  $n$  labels, by the matrices  $M \in \text{Mat}_2(m, l)$ ,  $N \in \text{Mat}_2(n, l)$  and  $P \in \text{Mat}_2(m, n)$ , is the  $l$ -labelled graph  $(G +_P H, \begin{pmatrix} B \cdot M \\ C \cdot N \end{pmatrix})$ , where  $G +_P H$  is the graph obtained from  $G$  and  $H$  by adding an edge  $\{i, j\}$  between the vertex  $i$  of  $G$  and the vertex  $j$  of  $H$  for every non-zero entry  $(i, j)$  of  $P$ . This operation adds the edges specified by  $P$  and recolours the vertices of  $G$  and  $H$  with  $M$  and  $N$ .

We assign a cost to the operations,  $w(\text{Recol}_M):= n$  and  $w(+_{M,P,N}):= \max\{m, n\}$ , and obtain a corresponding notion of width for decompositions.

*Example 2.57.* The 1-labelled 4-clique is expressed by the term

$$\text{Recol}_{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}(v_1 +_{1,1,1} \text{Recol}_{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}(v_1 +_{1,1,1} \text{Recol}_{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}(v_1 +_{1,1,1} v_1))),$$

that creates one vertex at a time and connects it to all existing vertices. The recolouring by the matrix  $\rightarrow c := \begin{pmatrix} 1 \\ 1 \end{pmatrix}$  assigns the same colour to all existing vertices. Note that the structure of this term resembles the structure of the second clique term in Example 2.37 for the same graph.

The operations above are similar in spirit to those of clique width (Definition 2.35) and, in fact, they define an equivalent width measure, rank width [CK07]. All these operations are derived from the categorical structure of graphs presented in Section 4.3.

**Theorem 2.58** ([CK07, Theorem 3.4]). *For a graph  $(G, B)$  with  $n$  labels, the algebraic width given by linear recolouring and bilinear product (Definition 2.56) is at least its rank width:*

$$\text{rwd}(G) \leq \text{awd}(G, B).$$

All these operations preserve monadic second order formulae. More precisely, the preservation theorem for disjoint union is known as Feferman-Vaugh-Mostowski preservation theorem [Fef57; FV59], while the preservation theorem for the fuse operation as defined above is by Courcelle and Makowsky [CM02, Lemma 5.2]. The proof of this statement relies on Ehrenfeucht-Fraïssé games [Fra55; Fra57; Ehr57; Ehr61]. For a reference, the preservation theorems can be found in Makowsky's review [Mak04]: for the disjoint union as Theorems 1.5 and 1.6, while for the fuse operation as Proposition 3.6.

**Theorem 2.59** ([FV59; CM02]). *The disjoint union and the fuse operation of  $\tau$ -structures with sources are effectively MSO-smooth operations.*

The preservation theorem for the rank width operations [CK09] is similar to that for clique width operations [CMR00] and relies on a result that shows that all quantifier free operations are effectively MSO-smooth [Cou92b, Theorem 3.4].

**Theorem 2.60** ([CK09, Proposition 3.2]). *Linear recolouring and bilinear product of graphs with labels are effectively MSO-smooth operations.*

This results allow us to compute the reduction set of MSO formulae and use it to run Algorithm 1 as described in Theorem 2.52 on MSO-inductive classes of relational structures. As consequences of Theorems 2.55 and 2.58 to 2.60, bounded tree width and bounded clique width classes of relational structures are, indeed, MSO-inductive.

**Theorem 2.61** ([BC87; Cou90]). *Classes of relational structures with sources of bounded tree width are effectively MSO-inductive with respect to disjoint union and the fuse operation. The same is true for classes of bounded branch width.*

**Theorem 2.62** ([CK09]). *Classes of graphs with labels of bounded clique width are effectively MSO-inductive with respect to linear recolouring and bilinear product. The same is true for classes of bounded rank width.*

Theorems 2.59 and 2.61 show that the assumptions of Theorem 2.52 hold for MSO logic and the operations of disjoint union and fusion of sources, which gives Courcelle's theorem for tree width [Cou92a, Proposition 3.1], while Theorems 2.60 and 2.62 show them for the operations for rank width [CMR00, Theorem 4].

We assume that the input graph is given as a term as we do not deal with the problem of finding efficient decompositions in this work. For tree width, it is known that the term can be computed in linear time [Bod93a], while, for clique width, it can be approximated [OS06].

**Corollary 2.63** ([Cou92a]). *For a formula  $\phi$  in the monadic second order logic of relational  $\tau$ -structures, the problem of checking  $\phi$  on an input structure of tree width at most  $k$  is linear in the number of its vertices.*

**Corollary 2.64** ([CMR00; CO00]). *For a formula  $\phi$  in the monadic second order logic of graphs, the problem of checking  $\phi$  on an input graph of clique width at most  $k$  is linear in the number of its vertices.*

## Chapter 3

# Monoidal Width

Monoidal width measures the structural complexity of morphisms in monoidal categories, and is the central definition of this work. Monoidal width takes from tree width and rank width to capture their algorithmic properties. The structural complexity of graphs, measured by tree and rank widths, gives an upper bound to the computational cost of checking a certain class of properties on graphs. Similarly, the structural complexity of morphisms in monoidal categories, measured by monoidal width, gives an upper bound to the computational cost of divide-and-conquer algorithms on monoidal categories.

Monoidal width depends on *monoidal decompositions* as tree width and rank width depend on tree and rank decompositions. A decomposition is a recipe for dividing a morphism, or a graph, into smaller parts with given operations. This can be done in different ways, using different operations in different orders. Some operations are more costly than others, which causes some decompositions to be more efficient than others and divide-and-conquer algorithms on some decompositions run faster than on others. Decompositions that use cheap operations are more efficient.

The operations for monoidal decompositions are the categorical composition and the monoidal product. Typically, compositions represent information or resource sharing, which makes them costly. On the other hand, monoidal products represent juxtaposition, which is usually cheap.

Monoidal decompositions are like algebraic decompositions for morphisms in monoidal categories where the choice of monoidal category fixes the operations.

Monoidal decompositions may seem more restricted than the algebraic decompositions introduced in Section 2.3. However, on the one hand, the flexibility of the choice of categorical algebra makes up for this restriction and it allows us to capture tree width and clique width as particular cases. On the other hand, there are advantages to this restriction as it gives canonicity to some of the numerous possible choices of operations that define equivalent width measures. As shown in the previous chapter, the operations that determine clique width are equivalent to those that determine rank width, in the sense that they determine equivalent width measures. Similarly, there are slightly different operations that all define tree width. The next chapter shows how all these operations are derivable from compositions and monoidal products in two different monoidal categories of graphs.

### 3.1 Decompositions in monoidal categories

A monoidal decomposition describes a process as sequential and parallel compositions of smaller processes. Explicitly, a monoidal decomposition is a syntax tree in the language of monoidal categories: internal nodes are compositions or monoidal products, and leaves are morphisms that, when assembled according to the operations in the decomposition, give the original morphism.

**Definition 3.1.** A monoidal decomposition  $d \in D_f$  of a morphism  $f : A \rightarrow B$  in a monoidal category  $\mathcal{C}$  is a syntax tree that uses the composition  $\circ$  and the monoidal product  $\otimes$  in  $\mathcal{C}$  as operations.

$$\begin{aligned}
 d & ::= (f) \\
 & \mid (d_1 \otimes d_2) && \text{if } d_1 \in D_{f_1}, d_2 \in D_{f_2} \text{ and } f = f_1 \otimes f_2 \\
 & \mid (d_1 \circ_C d_2) && \text{if } d_1 \in D_{f_1}, d_2 \in D_{f_2} \text{ and } f = f_1 \circ_C f_2
 \end{aligned}$$

The trivial decomposition,  $(f) \in D_f$ , is always a possibility, but usually not the best one, as it can cost more than other decompositions that break  $f$  into smaller components.

*Example 3.2.* Let  $f : 1 \rightarrow 2$  and  $g : 2 \rightarrow 1$  be morphisms in a prop. A monoidal decomposition of  $f \circ (f \otimes f) \circ (g \otimes g) \circ g$  can be described by vertical and horizontal cuts in the string diagram of the morphism (Figure 3.1). Vertical cuts represent compositions, while horizontal cuts represent monoidal products.

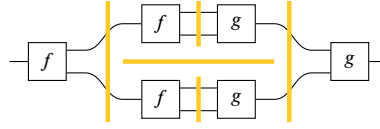


Figure 3.1: A monoidal decomposition represented with cuts in a string diagram.

Figure 3.1 encodes all the information of a monoidal decomposition but the order in which compositions and monoidal products are associated. Choosing the order in which compositions and monoidal products are performed, we obtain a formal expression of the decomposition in Figure 3.1.

$$(f \circ_2 - (((f \circ_2 - g) \otimes - (f \circ_2 - g)) \circ_2 - g)),$$

We will avoid writing decompositions in this form whenever possible.

The cost of a monoidal decomposition bounds the running time of a divide-and-conquer algorithm on this decomposition, and depends on the operations and morphisms that label its internal nodes and leaves. More precisely, it depends on a weight assigned to the operations and morphisms that appear in the decomposition, in a way that we describe below.

Each morphism has a *weight*. The running time of a divide-and-conquer algorithm on the trivial decomposition  $(f)$  depends, usually more than exponentially, on the weight of the morphism  $f$ , as it amounts to running the brute-force algorithm on  $f$ .

**Definition 3.3.** A *weight function*  $w : \text{Mor}(\mathcal{C}) \rightarrow \mathbb{N}^1$  for a monoidal category  $\mathcal{C}$  is a function that assigns a natural number to each morphism of  $\mathcal{C}$  such that

1.  $w(f \circ_B g) \leq w(f) + w(g) + w(B)$ , for  $f : A \rightarrow B$  and  $g : B \rightarrow C$ ; and
2.  $w(f \otimes g) \leq w(f) + w(g)$ .

The weight function extends to objects of  $\mathcal{C}$  by taking the weight of identity morphisms,  $w(A) := w(1_A)$ .

The two conditions on the weight function intuitively capture the behaviour of the running time of the brute-force algorithm on morphisms: the difference between running it on a composition  $f \circ_B g$  and on the two morphisms  $f$  and  $g$  separately depends only on the boundary  $B$  of the composition; the running time on a monoidal product  $f \otimes g$  depends only on the running time on the separate components  $f$  and  $g$ .

<sup>1</sup>We indicate with  $\text{Mor}(\mathcal{C})$  the set of morphisms of a small category  $\mathcal{C}$ . If the category  $\mathcal{C}$  is essentially small, we can still define a weight function for  $\mathcal{C}$  by defining it on its equivalent small category.

Given the weight of morphisms, we can assign a weight to the operations of a monoidal category. The weight of a composition along an object  $A$  is  $w(A) := w(\mathbb{1}_A)$ , while the weight of a monoidal product is 0. These determine the *width* of a decomposition by taking the maximum of the weights of operations and morphisms appearing in the decomposition.

**Definition 3.4.** The *width* of a monoidal decomposition  $d \in D_f$  of a morphism  $f : A \rightarrow B$  in a monoidal category  $C$  with a weight function  $w$  is defined inductively below.

$$\begin{aligned} \text{wd}(d) := & w(f) && \text{if } d = (f) \\ & \max\{\text{wd}(d_1), \text{wd}(d_2)\} && \text{if } d = (d_1 - \otimes - d_2) \\ & \max\{\text{wd}(d_1), w(C), \text{wd}(d_2)\} && \text{if } d = (d_1 - \circ_C - d_2) \end{aligned}$$

The *size* of the monoidal decomposition  $d$  is the number of its nodes.

$$\begin{aligned} \text{size}(d) := & 1 && \text{if } d = (f) \\ & \text{size}(d_1) + 1 + \text{size}(d_2) && \text{if } d = (d_1 - \otimes - d_2) \text{ or } d = (d_1 - \circ_C - d_2) \end{aligned}$$

Thanks to the inequalities in Definition 3.3, the weight of a morphism is bounded by the product of the size and the width of any of its decompositions.

**Lemma 3.5.** Let  $d \in D_f$  be a monoidal decomposition of a morphism  $f : A \rightarrow B$  in a monoidal category  $C$ . Then,

$$w(f) \leq \text{wd}(d) \cdot \text{size}(d).$$

*Proof.* This is easily shown by induction on  $d$ . If  $d = (f)$  is a leaf, then its width coincides with the weight of  $f$ ,  $\text{wd}(d) := w(f)$ , and its size is 1. If  $d = (d_1 - \circ_B - d_2)$  or  $d = (d_1 - \otimes - d_2)$ , we bound the weight of  $f$  applying the inequalities of Definition 3.3 and the induction hypothesis.

$$\begin{aligned} w(f) & & w(f) \\ \leq w(f_1) + w(f_2) + w(B) & & \leq w(f_1) + w(f_2) \\ \leq \text{wd}(d_1) \text{size}(d_1) + \text{wd}(d_2) \text{size}(d_2) + w(B) & & \leq \text{wd}(d_1) \text{size}(d_1) + \text{wd}(d_2) \text{size}(d_2) \\ \leq \max\{\text{wd}(d_1), w(B), \text{wd}(d_2)\} & & \leq \max\{\text{wd}(d_1), \text{wd}(d_2)\} \\ \cdot (\text{size}(d_1) + \text{size}(d_2) + 1) & & \cdot (\text{size}(d_1) + \text{size}(d_2) + 1) \\ = \text{wd}(d) \cdot \text{size}(d) & & = \text{wd}(d) \cdot \text{size}(d) \end{aligned}$$

□

The width of a decomposition is not influenced by the order in which the operations appear, but only by their costs. This means that all the different monoidal decompositions corresponding to the cuts in Figure 3.1 have the same width and this representation can be used without any consequences.

*Example 3.6.* The width of the decomposition in Example 3.2, if we assume that  $w(f) = w(g) = 2$ , is 2. In fact, compositions are along at most 2 wires, and the morphisms at the leaves all weight 2.

The *monoidal width* of a morphism is the width of a cheapest decomposition, and gives a bound for the running time of a divide-and-conquer algorithm on the given morphism.

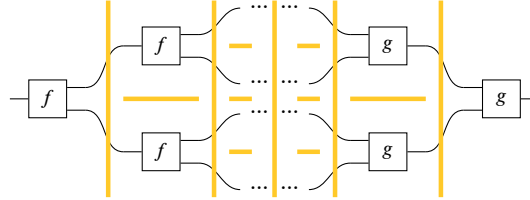
**Definition 3.7.** The *monoidal width* of a morphism  $f$  in a monoidal category  $C$  with a weight function  $w$  is the width of a cheapest decomposition:

$$\text{mwd}(f) := \min_{d \in D_f} \text{wd}(d).$$

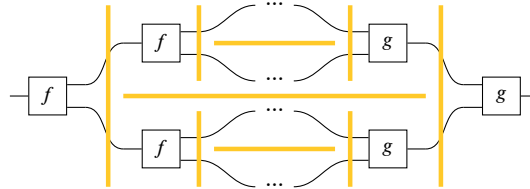
*Example 3.8.* With the morphisms  $f$  and  $g$  as in Example 3.2, we define a family of morphisms  $h_n : 1 \rightarrow 1$  inductively:

- $h_0 := f \circ_2 g$ ;
- $h_{n+1} := f \circ_2 (h_n \otimes h_n) \circ_2 g$ .

Each  $h_n$  has a monoidal decomposition of width  $2^n$  where the first node is the composition along the  $2^n$  wires in the middle.



However, the monoidal decomposition below shows that  $\text{mwd}(h_n) \leq 2$  for any  $n$ .

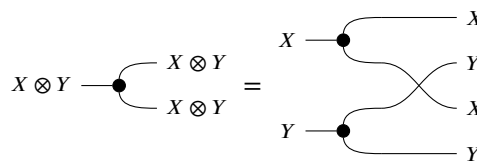


### 3.2 Categories with copy

A simple case study for monoidal decompositions are the copy morphisms of symmetric monoidal categories with coherent copying. We bound their monoidal width, a result that is useful to compute the width in props with biproducts (Section 3.3) and prove the more complex bounds in Chapters 5 and 6.

**Definition 3.9.** A symmetric monoidal category  $\mathcal{C}$  has *coherent copying* if there is a class of *copiable* objects  $\Delta_{\mathcal{C}} \subseteq \text{Obj}(\mathcal{C})$  such that

- $X, Y \in \Delta_{\mathcal{C}}$  iff  $X \otimes Y \in \Delta_{\mathcal{C}}$ ;
- every object  $X \in \Delta_{\mathcal{C}}$  is endowed with a *copy* morphism  $\text{copy}_X : X \rightarrow X \otimes X$ ;
- the copy morphisms are *coherent*: for every  $X, Y \in \Delta_{\mathcal{C}}$ ,  $\text{copy}_{X \otimes Y} = (\text{copy}_X \otimes \text{copy}_Y) \circ (\mathbb{1}_X \otimes \sigma_{X,Y} \otimes \mathbb{1}_Y)$ .



For props with coherent copy, we assume that the weight of copy morphisms, symmetries and identities is given by  $w(\text{copy}_X) := 2 \cdot w(X)$ ,  $w(\sigma_{X,Y}) := w(X) + w(Y)$  and  $w(\mathbb{1}_X) := w(X)$ . Note that, on these morphisms, this weight function satisfies the conditions in Definition 3.3.

*Example 3.10.* Any cartesian prop has coherent copying, where the copy morphisms are the universal ones given by the cartesian structure:  $\text{copy}_n := \langle \mathbb{1}_n, \mathbb{1}_n \rangle : n \rightarrow n + n$ . The monoidal width of the copy morphism on  $n$  is bounded by  $n + 1$ . This is shown more generally in Lemma 3.11, but the idea of the proof can be exemplified in this case. Let  $\gamma_{n,m} := (\text{copy}_n \otimes \mathbb{1}_m) \circ (\mathbb{1}_n \otimes \sigma_{n,m}) : n + m \rightarrow n + m + n$  be the morphism in Figure 3.2. We can decompose  $\gamma_{n,m}$  in terms of  $\gamma_{n-1,m+1}$  (in the dashed box in Figure 3.2), the copy morphism

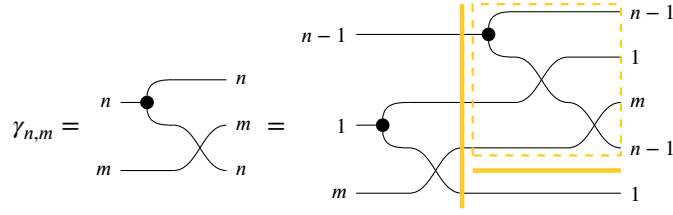


Figure 3.2: Decomposing copy morphisms.

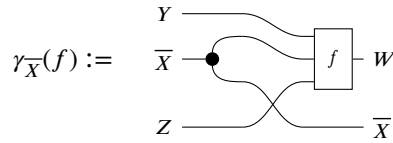
$\dashv\!\!\!\dashv_1$  and the symmetry  $\sigma_{1,1}$ , by cutting along at most  $n + 1 + m$  wires:

$$\gamma_{n,m} = (\mathbb{1}_{n-1} \otimes ((\dashv\!\!\!\dashv_1 \otimes \mathbb{1}_1) \circ (\mathbb{1}_1 \otimes \sigma_{1,1}))) \circ_{n+1+m} (g_{n-1,m+1} \otimes \mathbb{1}_1).$$

By induction, we decompose  $\dashv\!\!\!\dashv_n = \gamma_{n,0}$  cutting along only  $n + 1$  wires. In particular, this means that  $\text{mwd}(\dashv\!\!\!\dashv_n) \leq n + 1$ .

The following result generalises the reasoning in Example 3.10.

**Lemma 3.11.** *Let  $\mathcal{C}$  be a symmetric monoidal category with coherent copying and  $d \in D_f$  be a monoidal decomposition of a morphism  $f : Y \otimes \bar{X} \otimes Z \rightarrow W$ , with  $\bar{X} := X_1 \otimes \dots \otimes X_n$ . Then we can construct a monoidal decomposition  $C_{\bar{X}}(d)$  of the morphism  $\gamma_{\bar{X}}(f) := (\mathbb{1}_Y \otimes \dashv\!\!\!\dashv_{\bar{X}} \otimes \mathbb{1}_Z) \circ (\mathbb{1}_{Y \otimes \bar{X}} \otimes \sigma_{\bar{X},Z}) \circ (f \otimes \mathbb{1}_{\bar{X}})$*

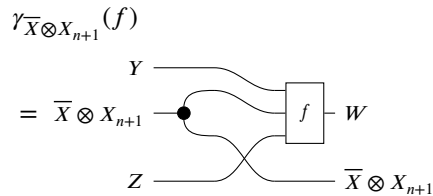


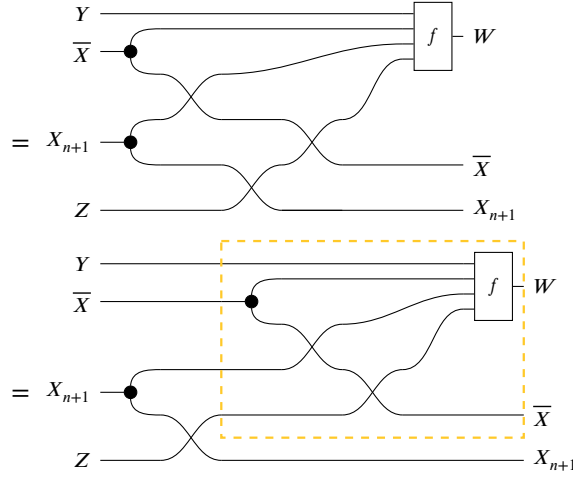
of bounded width:

$$\text{wd}(C_{\bar{X}}(d)) \leq \max\{\text{wd}(d), \text{w}(Y) + \text{w}(Z) + (n + 1) \cdot \max_{i=1,\dots,n} \text{w}(X_i)\}.$$

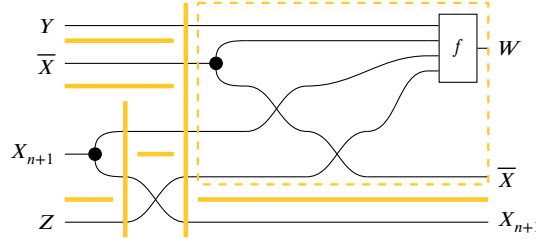
*Proof.* Proceed by induction on the number  $n$  of objects being copied. If  $n = 0$ , then we are done because we can keep the decomposition  $d : C_f(d) := d$ .

Suppose that the statement is true for any  $f' : Y \otimes \bar{X} \otimes Z' \rightarrow W$  and let  $f : Y \otimes \bar{X} \otimes X_{n+1} \otimes Z \rightarrow W$ . Then we can rewrite  $\gamma_{\bar{X} \otimes X_{n+1}}(f)$  using coherence of the copy morphisms  $\dashv\!\!\!\dashv$  and the properties of the symmetries  $\sigma$ .





Consider  $\gamma_{\bar{X}}(f) := (\mathbb{1} \otimes \bar{\hookrightarrow}_{\bar{X}} \otimes \mathbb{1}) \circ (\mathbb{1} \otimes \sigma) \circ (f \otimes \mathbb{1})$ , the morphism in the dashed box. By the induction hypothesis, there is a monoidal decomposition  $C_{\bar{X}}(d)$  of  $\gamma_{\bar{X}}(f)$  with bounded width:  $\text{wd}(C_{\bar{X}}(d)) \leq \max\{\text{wd}(d), \text{w}(Y) + \text{w}(X_{n+1} \otimes Z) + (n+1) \cdot \max_{i=1, \dots, n} \text{w}(X_i)\}$ . Using this decomposition, we can define a monoidal decomposition  $C_{\bar{X} \otimes X_{n+1}}(d)$  of  $\gamma_{\bar{X} \otimes X_{n+1}}(f)$  as shown below.



Note that the only cut that matters is the longest vertical one, the composition node along  $Y \otimes \bar{X} \otimes X_{n+1} \otimes Z \otimes X_{n+1}$ , because all the other cuts are cheaper. The cost of this cut is  $\text{w}(Y) + \text{w}(Z) + 2 \cdot \text{w}(X_{n+1}) + \text{w}(\bar{X}) = \text{w}(Y) + \text{w}(Z) + \text{w}(X_{n+1}) + \sum_{i=1}^{n+1} \text{w}(X_i)$ . With this observation and applying the induction hypothesis, we can compute the width of the decomposition  $C_{\bar{X} \otimes X_{n+1}}(d)$ .

$$\begin{aligned}
& \text{wd}(C_{\bar{X} \otimes X_{n+1}}(d)) \\
&= \max\{\text{w}(\mathbb{1}_{Y \otimes \bar{X}}), \text{w}(\bar{\hookrightarrow}_{X_{n+1}}), \text{w}(\mathbb{1}_Z), \text{w}(\mathbb{1}_{X_{n+1}}), \text{w}(\sigma_{X_{n+1}, Z}), \text{wd}(C_{\bar{X}}(d)), \\
&\quad \text{w}(Y \otimes \bar{X} \otimes Z \otimes X_{n+1}), \text{w}(X_{n+1} \otimes Z \otimes X_{n+1})\} \\
&\leq \max\{\text{w}(Y) + \text{w}(Z) + \text{w}(X_{n+1}) + \sum_{i=1}^{n+1} \text{w}(X_i), \text{wd}(C_{\bar{X}}(d))\} \\
&\leq \max\{\text{w}(Y) + \text{w}(Z) + (n+2) \cdot \max_{i=1, \dots, n+1} \text{w}(X_i), \text{wd}(C_{\bar{X}}(d))\} \\
&\leq \max\{\text{w}(Y) + \text{w}(Z) + (n+2) \cdot \max_{i=1, \dots, n+1} \text{w}(X_i), \\
&\quad \text{wd}(d), \text{w}(Y) + \text{w}(X_{n+1} \otimes Z) + (n+1) \cdot \max_{i=1, \dots, n} \text{w}(X_i)\} \\
&= \max\{\text{w}(Y) + \text{w}(Z) + (n+2) \cdot \max_{i=1, \dots, n+1} \text{w}(X_i), \text{wd}(d)\}
\end{aligned}$$



□

### 3.3 Categories with biproducts

This section shows another simple example of monoidal decompositions. In props with biproducts, morphisms have a rank which is related to their monoidal width. An example of such props is the category of matrices<sup>2</sup>.

*Example 3.12.* The category  $\text{Mat}_R$  of matrices over a semiring  $R$  is a prop where the monoidal product is a biproduct. Its morphisms  $n \rightarrow m$  are  $m$  rows by  $n$  columns matrices with entries in the semiring  $R$  and the biproduct of matrices,  $A \oplus B := \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$ , is the monoidal product.

By the string diagrammatic formulation of Fox's theorem [Fox76], every object in a bicartesian prop has natural commutative monoid and cocommutative comonoid structures. This structures are fundamental for the proofs in this section.

**Theorem 3.13.** *A symmetric monoidal category  $C$  is cartesian if and only if every object  $A$  is equipped with a cocommutative comonoid structure and this structure is natural and uniform, whose structure morphisms and equations are in Figure 3.3.*

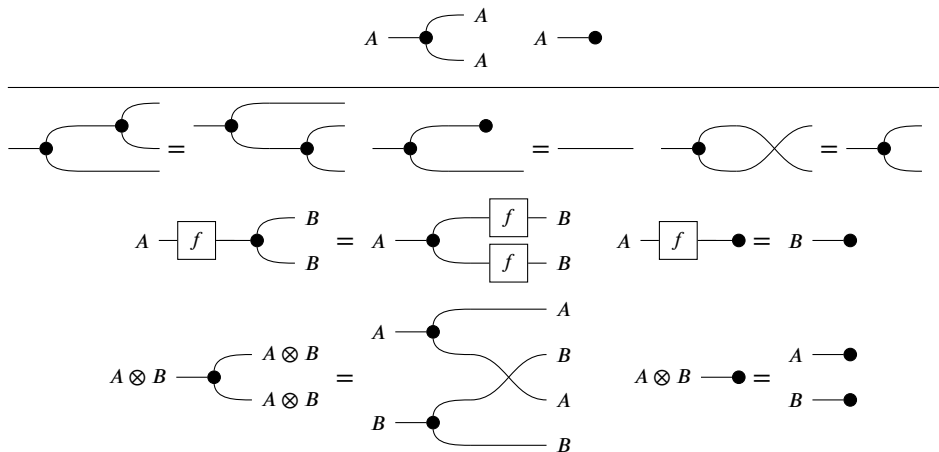


Figure 3.3: Structure and equations for a natural and uniform cocommutative comonoid.

The results in this section hold for monoidal categories where the monoidal product is the biproduct and whose objects are a unique factorisation monoid. To help readability, some results are stated for the particular case of props, but they apply to, for example, coloured props as well. When the monoidal product is the biproduct, then, in particular, the monoidal unit is the zero object. Then, there is only one scalar: the only morphism  $I \rightarrow I$  is the identity. In some sense, this means that the interesting part of a morphism happens on the boundary and a reasonable choice of weight function for these categories only keeps track of the complexity of the boundaries.

<sup>2</sup>We thank JS Lemay for suggesting to generalise this result for matrices to categories with biproducts.

**Definition 3.14.** For a prop  $\mathcal{P}$ , define a weight function  $w : \mathcal{A} \rightarrow \mathbb{N}$  as  $w(g) := \max\{m, n\}$ , for  $g : n \rightarrow m$  in  $\mathcal{P}$ . For a monoidal category  $\mathcal{C}$  where the objects are a unique factorisation monoid, define the *dimension*  $|X|$  of an object  $X$  to be the number of factors in its unique  $\otimes$ -factorisation  $X = X_1 \otimes \dots \otimes X_k$ ,  $|X| := k$ . A weight function for  $\mathcal{C}$  is  $w : \mathcal{A} \rightarrow \mathbb{N}$  as  $w(g) := \max\{|X|, |Y|\}$ , for  $g : X \rightarrow Y$  in  $\mathcal{C}$ .

This definition satisfies the conditions for a weight function.

**Lemma 3.15.** *In a monoidal category whose objects are a unique factorisation monoid, the function  $w$  in Definition 3.14 satisfies the conditions for a weight function in Definition 3.3.*

*Proof.* For morphisms  $f : X \rightarrow Y$ ,  $g : Y \rightarrow Z$  and  $f' : X' \rightarrow Y'$  in  $\mathcal{C}$ , let  $X = X_1 \otimes \dots \otimes X_l$ ,  $Y = Y_1 \otimes \dots \otimes Y_m$ ,  $Z = Z_1 \otimes \dots \otimes Z_n$ ,  $X' = X'_1 \otimes \dots \otimes X'_{l'}$  and  $Y' = Y'_1 \otimes \dots \otimes Y'_{m'}$  be the unique  $\otimes$ -factorisations of  $X, Y, Z, X'$  and  $Y'$ . We compute and bound their weights.

$$\begin{array}{ll} w(f \circ g) & w(f \otimes f') \\ := \max\{l, n\} & := \max\{l + l', m + m'\} \\ \leq \max\{l, m, n\} + m & \leq \max\{l + l', l + m', l' + m, m + m'\} \\ \leq \max\{l, m\} + \max\{m, n\} + m & = \max\{l, m\} + \max\{l', m'\} \\ =: w(f) + w(g) + m & =: w(f) + w(f') \end{array}$$

□

The proof strategy consists in finding a standard shape of decomposition and show that it is minimal. When a morphism  $f$  can be written as a monoidal product  $f = f_1 \otimes \dots \otimes f_k$  of morphisms of smaller weight, the decompositions that use this factorisation are more efficient (Proposition 3.19). Under the assumptions above, every morphism has a unique  $\otimes$ -factorisation (Lemma 3.20) and a minimal decomposition must use this factorisation.

$$f = f_1 \otimes \dots \otimes f_k = \begin{array}{c} \begin{array}{|c|} \hline u_1 \\ \hline \end{array} \begin{array}{|c|} \hline v_1 \\ \hline \end{array} \\ \hline \begin{array}{|c|} \hline u_2 \\ \hline \end{array} \begin{array}{|c|} \hline v_2 \\ \hline \end{array} \\ \vdots \\ \begin{array}{|c|} \hline u_k \\ \hline \end{array} \begin{array}{|c|} \hline v_k \\ \hline \end{array} \end{array}$$

Every factor  $f_i$  can be minimally split as a composition  $f_i = u_i \circ_{r_i} v_i$  and give a decomposition of  $f$  of width at least  $\max_{i=1, \dots, k} r_i$ . We show that each  $u_i$  and  $v_i$  can be further decomposed and their monoidal width is at most  $r_i + 1$ . This compound decomposition is minimal and bounds the monoidal width of  $f$  as  $\max_{i=1, \dots, k} r_i \leq \text{mwd}(f) \leq \max_{i=1, \dots, k} r_i + 1$ .

The shape of the minimal decomposition above shows that minimal vertical cuts play an important role in computing monoidal width. Following the characterisation of rank for matrices, we define the rank of morphisms as their minimal vertical cut.

**Lemma 3.16** ([PO99]). *Let  $A : n \rightarrow m$  in  $\text{Mat}_{\mathbb{N}}$ . Then  $\min\{k \in \mathbb{N} : A = B \circ_k C\} = \text{rk}(A)$ .*

**Definition 3.17.** The *rank* of a morphism  $f : n \rightarrow m$  in a prop  $\mathcal{P}$  is its minimal vertical cut:

$$\text{rk}(f) := \min\{k \in \mathbb{N} : f = g \circ_k h\}.$$

Similarly, for a morphism  $f : X \rightarrow Y$  in a monoidal category  $\mathcal{C}$ , whose objects are a unique factorisation monoid, its *rank* is its minimal vertical cut:

$$\text{rk}(f) := \min\{k \in \mathbb{N} : f = g \circ_{\mathcal{C}} h \wedge |C| = k\}.$$

The first step for computing monoidal width is to show that, whenever possible, decompositions should start with a  $\otimes$  node. This result needs a technical lemma: discarding outputs or inputs of a morphism cannot increase its width.

**Lemma 3.18.** *Let  $f : n \rightarrow m$  in a prop  $\mathbf{P}$  where  $0$  is both initial and terminal and  $d \in D_f$ . Let  $f_D := f \circ (\mathbb{1}_{m-k} \otimes \bullet_k)$  and  $f_Z := (\mathbb{1}_{n-k} \otimes \circ_k) \circ f$ , with  $k \leq m$  and  $k \leq n$ , respectively.*

$$f_D := n \text{ --- } \boxed{f} \text{ --- } \bullet \text{ --- } m-k, \quad f_Z := n-k \text{ --- } \circ \text{ --- } \boxed{f} \text{ --- } m.$$

Then there are monoidal decompositions  $\mathcal{D}(d) \in D_{f_D}$  and  $\mathcal{Z}(d) \in D_{f_Z}$  of bounded width,  $\text{wd}(\mathcal{D}(d)) \leq \text{wd}(d)$  and  $\text{wd}(\mathcal{Z}(d)) \leq \text{wd}(d)$ .

*Proof.* We show the inequality for  $f_D$  by induction on the decomposition  $d$ . The inequality for  $f_Z$  follows from the fact that the same proof applies to  $\mathbf{P}^{\text{op}}$ . If the decomposition has only one node,  $d = (f)$ , then we define  $\mathcal{D}(d) := (f_D)$  and obtain that

$$\text{wd}(\mathcal{D}(d)) := \max\{n, m-k\} \leq \max\{n, m\} =: \text{wd}(d).$$

If the decomposition starts with a composition node,  $d = (d_1 \circ_j d_2)$ , then  $f = f_1 \circ_j f_2$ , where  $d_i$  is a monoidal decomposition of  $f_i$ .

$$n \text{ --- } \boxed{f} \text{ --- } \bullet \text{ --- } m-k = n \text{ --- } \boxed{f_1} \text{ --- } \boxed{f_2} \text{ --- } \bullet \text{ --- } m-k$$

By induction hypothesis, there is a monoidal decomposition  $\mathcal{D}(d_2)$  of  $f_2 \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k)$  such that  $\text{wd}(\mathcal{D}(d_2)) \leq \text{wd}(d_2)$ . We use this decomposition to define a decomposition  $\mathcal{D}(d) := (d_1 \circ_j \mathcal{D}(d_2))$  of  $f_D$ . Then,  $\mathcal{D}(d)$  is a monoidal decomposition of  $f \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k)$  because  $f \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k) = f_1 \circ_j f_2 \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k)$  and its width is bounded.

$$\text{wd}(\mathcal{D}(d)) := \max\{\text{wd}(d_1), j, \text{wd}(\mathcal{D}(d_2))\} \leq \max\{\text{wd}(d_1), j, \text{wd}(d_2)\} =: \text{wd}(d)$$

If the decomposition starts with a tensor node,  $d = (d_1 \otimes d_2)$ , then  $f = f_1 \otimes f_2$ , with  $d_i$  monoidal decomposition of  $f_i : n_i \rightarrow m_i$ . There are two possibilities: either  $k \leq m_2$  or  $k > m_2$ . If  $k \leq m_2$ , then  $f \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k) = f_1 \otimes (f_2 \circ_j (\mathbb{1}_{m_2-k} \otimes \bullet_k))$ .

$$n \text{ --- } \boxed{f} \text{ --- } \bullet \text{ --- } m-k = \begin{array}{c} n_1 \text{ --- } \boxed{f_1} \text{ --- } m_1 \\ n_2 \text{ --- } \boxed{f_2} \text{ --- } \bullet \text{ --- } m_2-k \end{array}$$

By induction hypothesis, there is a monoidal decomposition  $\mathcal{D}(d_2)$  of  $f_2 \circ_j (\mathbb{1}_{m_2-k} \otimes \bullet_k)$  such that  $\text{wd}(\mathcal{D}(d_2)) \leq \text{wd}(d_2)$ . Then, we can use this decomposition to define a decomposition  $\mathcal{D}(d) := (d_1 \otimes \mathcal{D}(d_2))$  of  $f_D$  whose width is bounded.

$$\text{wd}(\mathcal{D}(d)) := \max\{\text{wd}(d_1), \text{wd}(\mathcal{D}(d_2))\} \leq \max\{\text{wd}(d_1), \text{wd}(d_2)\} =: \text{wd}(d)$$

If  $k > m_2$ , then  $f \circ_j (\mathbb{1}_{m-k} \otimes \bullet_k) = (f_1 \circ_j (\mathbb{1}_{m_1-k+m_2} \otimes \bullet_{k-m_2})) \otimes (f_2 \circ_j \bullet_{m_2})$ .

$$n \text{ --- } \boxed{f} \text{ --- } \bullet \text{ --- } m-k = \begin{array}{c} n_1 \text{ --- } \boxed{f_1} \text{ --- } \bullet \text{ --- } m_1-k+m_2 \\ n_2 \text{ --- } \boxed{f_2} \text{ --- } \bullet \text{ --- } m_2 \end{array}$$

By induction hypothesis, there are monoidal decompositions  $D(d_i)$  of  $f_1 \circ (\mathbb{1}_{m_1-k+m_2} \otimes \bullet_{k-m_2})$  and  $f_2 \circ \bullet_{m_2}$  such that  $\text{wd}(D(d_i)) \leq \text{wd}(d_i)$ . Then, we can use these decompositions to define a monoidal decomposition  $D(d) := (D(d_1) \otimes D(d_2))$  of  $f_D$  of bounded width.

$$\text{wd}(D(d)) := \max\{\text{wd}(D(d_1)), \text{wd}(D(d_2))\} \leq \max\{\text{wd}(d_1), \text{wd}(d_2)\} =: \text{wd}(d)$$

□

As a consequence, decompositions that start with a  $\otimes$  node are more efficient.

**Proposition 3.19.** *Let  $f : n \rightarrow m$  be a morphism in a prop  $\mathcal{P}$  and  $d' = (d'_1 \circ \bullet_k - d'_2) \in D_f$  be a decomposition of  $f$ . Suppose there are  $f_1 : n_1 \rightarrow m_1$  and  $f_2 : n_2 \rightarrow m_2$  such that  $f = f_1 \otimes f_2$ . Then, there is  $d = (d_1 \otimes - d_2) \in D_f$  such that  $\text{wd}(d) \leq \text{wd}(d')$ .*

*Proof.* Since the monoidal unit is the zero object,  $f_1 = (\mathbb{1} \otimes \circ_{-n_1}) \circ f \circ (\mathbb{1} \otimes \bullet_{m_1})$  and  $f_2 = (\circ_{-n_2} \otimes \mathbb{1}) \circ f \circ (\bullet_{m_2} \otimes \mathbb{1})$ . By Lemma 3.18, there are monoidal decompositions  $d_1 = \mathcal{Z}_1(D_1(d'))$  and  $d_2 = \mathcal{Z}_2(D_2(d'))$  of  $f_1$  and  $f_2$  with bounded width,  $\text{wd}(d_i) \leq \text{wd}(d')$ . Then, the decomposition  $d := (d_1 \otimes - d_2)$  is a monoidal decomposition of  $f$  and

$$\begin{aligned} \text{wd}(d) \\ &:= \max\{\text{wd}(d_1), \text{wd}(d_2)\} \\ &\leq \text{wd}(d') \end{aligned}$$

□

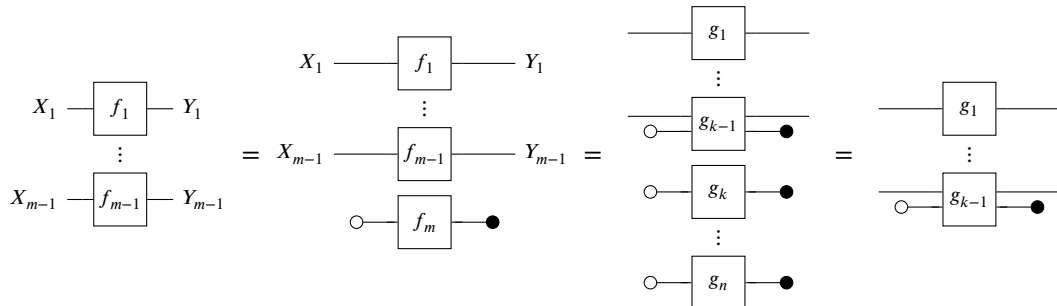
In monoidal categories where the monoidal unit is a zero object and the objects are a unique factorisation monoid, morphisms have a unique  $\otimes$ -decomposition.

**Lemma 3.20.** *Let  $\mathcal{C}$  be a monoidal category whose monoidal unit  $0$  is a zero object, and whose objects are a unique factorisation monoid. Then any morphism  $f$  in  $\mathcal{C}$  has a unique  $\otimes$ -decomposition.*

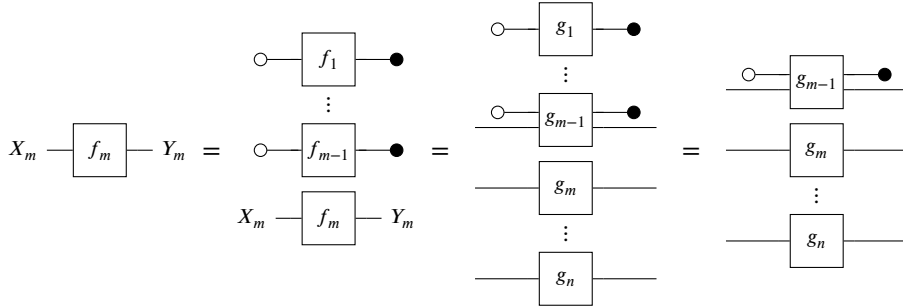
*Proof.* Suppose that  $f : X \rightarrow Y$  has two  $\otimes$ -decompositions  $f = f_1 \otimes \dots \otimes f_m = g_1 \otimes \dots \otimes g_n$  with  $f_i : X_i \rightarrow Y_i$  and  $g_j : Z_j \rightarrow W_j$  that are non  $\otimes$ -decomposable. Suppose  $m \leq n$  and proceed by induction on  $m$ .

If  $m = 0$ , then  $X = 0$  is the empty monoidal product, and  $f = \mathbb{1}_0$  and  $g_i = \mathbb{1}_0$  for every  $i = 1, \dots, n$  must be identities on  $0$  because  $0$  is both initial and terminal.

For the induction step, suppose that  $\bar{f} := f_1 \otimes \dots \otimes f_{m-1}$  has a unique  $\otimes$ -decomposition. Let  $A_1 \otimes \dots \otimes A_\alpha$  and  $B_1 \otimes \dots \otimes B_\beta$  be the unique  $\otimes$ -decompositions of  $X_1 \otimes \dots \otimes X_m = Z_1 \otimes \dots \otimes Z_n$  and  $Y_1 \otimes \dots \otimes Y_m = W_1 \otimes \dots \otimes W_n$ , respectively. Then, there are  $x \leq \alpha$  and  $y \leq \beta$  such that  $A_1 \otimes \dots \otimes A_x = X_1 \otimes \dots \otimes X_{m-1}$  and  $B_1 \otimes \dots \otimes B_y = Y_1 \otimes \dots \otimes Y_{m-1}$ . Then, we can rewrite  $\bar{f}$  in terms of  $g_i$ s, for some  $k \leq n$ :



By induction hypothesis,  $\bar{f}$  has a unique  $\otimes$ -decomposition, thus it must be that  $k = m$ , for every  $i < m - 1$   $f_i = g_i$  and  $f_{m-1} = (\mathbb{1} \otimes \circ-) \circledast g_k \circledast (\mathbb{1} \otimes -\bullet)$  because  $g_i$  are not  $\otimes$ -decomposable. Then, we can express  $f_m$  in terms of  $g_m, \dots, g_n$ :



By hypothesis,  $f_m$  is not  $\otimes$ -decomposable and  $m \leq n$ . Thus,  $n = m$ ,  $f_{m-1} = g_{m-1}$  and  $f_m = g_m$ . □

These results show that a minimal monoidal decomposition of  $f = f_1 \otimes \dots \otimes f_k$  can be obtained from minimal monoidal decompositions of  $f_i$ .

**Corollary 3.21.** *Let  $f = f_1 \otimes \dots \otimes f_k$  be the unique  $\otimes$ -decomposition of a morphism  $f$  in a monoidal category where the monoidal unit is a zero object and the objects are a unique factorisation monoid. Then, a minimal monoidal decomposition of  $f$  is  $d = (d_1 - \otimes - (d_2 - \otimes - \dots - d_k))$ , for minimal decompositions  $d_i$  of  $f_i$ .*

How do we find minimal decompositions of the factors  $f_i$ ? Since they cannot be  $\otimes$ -factored further, their minimal decompositions will start with a composition node. When this composition node is minimal, it corresponds to the rank and we obtain  $\text{mwd}(f) \geq \max_i \{\text{rk}(f_i)\}$ . For the upper bound, we show that every  $f_i$  can be decomposed with width at most  $\text{rk}(f_i) + 1$ . The unpleasant  $+1$  in this bound comes from the difference between the weight and the minimal boundary of the morphisms  $\circ-$  and  $-\bullet$ , and from the  $+1$  in the bound of the monoidal width of copy morphisms in Lemma 3.11.

**Proposition 3.22.** *The monoidal width of a morphism  $f : n \rightarrow m$  in a bicartesian prop  $\mathbb{P}$  is bounded by its domain and codomain:  $\text{mwd}(f) \leq \min\{m, n\} + 1$ .*

*Proof.* We proceed by induction on  $k = \max\{m, n\}$ . There are three base cases.

- If  $n = 0$ , then  $f = \circ-m$  because 0 is initial by hypothesis. Then,  $\text{mwd}(f) = \text{mwd}(\otimes_m \circ-) \leq w(\circ-) = \max\{1, 1\} \leq \min\{0, 1\} + 1$ .
- If  $m = 0$ , then  $f = -\bullet_n$  because 0 is terminal by hypothesis. Then,  $\text{mwd}(f) = \text{mwd}(\otimes_n -\bullet) \leq w(-\bullet) = \max\{1, 1\} \leq \min\{0, 1\} + 1$ .
- If  $m = n = 1$ , then  $\text{mwd}(f) \leq w(f) = \max\{1, 1\} \leq \min\{1, 1\} + 1$  by definition of the weight function.

For the induction steps, suppose that the statement is true for any  $f' : n' \rightarrow m'$  with  $\max\{m', n'\} < k = \max\{m, n\}$  and  $\min\{m', n'\} \geq 1$ . There are three possibilities.

1. If  $0 < n < m = k$ , then  $f$  can be decomposed as shown below because  $\dashv\vdash_{n+1}$  is uniform and morphisms are copiable because  $\mathbb{P}$  is cartesian by hypothesis.

$$\begin{aligned}
& n \text{ --- } \boxed{f} \text{ --- } m \\
&= n \text{ --- } \boxed{f} \text{ --- } \begin{array}{l} m-1 \\ 1 \end{array} \\
&= n \text{ --- } \boxed{f} \text{ --- } \begin{array}{l} \bullet \text{ --- } m-1 \\ \bullet \text{ --- } 1 \end{array} \\
&= n \text{ --- } \bullet \text{ --- } \boxed{f} \text{ --- } \bullet \text{ --- } \boxed{f} \text{ --- } \begin{array}{l} m-1 \\ 1 \end{array}
\end{aligned}$$

This corresponds to  $f = \dashv\vdash_n \circ (\mathbb{1}_n \otimes h_1) \circ \circ_{n+1} (h_2 \otimes \mathbb{1}_1)$ , where  $h_1 := f \circ (\dashv\vdash_{m-1} \otimes \mathbb{1}_1) : n \rightarrow 1$  and  $h_2 := f \circ (\mathbb{1}_{m-1} \otimes \dashv\vdash_1) : n \rightarrow m-1$ .

Then,  $\text{mwd}(f) \leq \max\{\text{mwd}(\dashv\vdash_n \circ (\mathbb{1}_n \otimes h_1)), n+1, \text{mwd}(h_2 \otimes \mathbb{1}_1)\}$ . So, we want to bound the monoidal width of the two morphisms appearing in the formula above. For the first morphism, we apply the induction hypothesis because  $h_1 : n \rightarrow 1$  and  $1, n < k$  and we apply Lemma 3.11. For the second morphism, we apply the induction hypothesis because  $h_2 : n \rightarrow m-1$  and  $n, m-1 < k$ .

$$\begin{array}{ll}
\text{mwd}(\dashv\vdash_n \circ (\mathbb{1}_n \otimes h_1)) & \text{mwd}(h_2 \otimes \mathbb{1}_1) \\
\leq \text{(by Lemma 3.11)} & = \text{(by Definition 3.4)} \\
\max\{\text{mwd}(h_1), n+1\} & \text{mwd}(h_2) \\
\leq \text{(by induction hypothesis)} & \leq \text{(by induction hypothesis)} \\
\max\{\min\{n, 1\} + 1, n+1\} & \min\{n, m-1\} + 1 \\
= \text{(because } 0 < n) & = \text{(because } n \leq m-1) \\
n+1 & n+1
\end{array}$$

Then,  $\text{mwd}(f) \leq n+1 = \min\{m, n\} + 1$  because  $n < m$ .

2. If  $0 < m < n = k$ , we can apply Case 1 to  $\mathbb{P}^{\text{op}}$  with the same assumptions on the set of atoms because  $\mathbb{P}^{\text{op}}$  is also bicartesian. We obtain that  $\text{mwd}(f) \leq m+1 = \min\{m, n\} + 1$  because  $m < n$ .
3. If  $0 < m = n = k$ ,  $f$  can be decomposed as in Case 1 (or Case 2) and, instead of applying the induction hypothesis to bound  $\text{mwd}(h_1)$  and  $\text{mwd}(h_2)$ , one applies Case 2 (or Case 1). Then,  $\text{mwd}(f) \leq m+1 = \min\{m, n\} + 1$  because  $m = n$ . □

**Lemma 3.23.** *The monoidal width of a morphism  $f : n \rightarrow m$  in a bicartesian prop  $\mathbb{P}$  is bounded by its rank:  $\text{mwd} f \leq \text{rk}(f) + 1$ . Moreover, if  $f$  is not  $\otimes$ -decomposable, i.e. there are no  $f_1, f_2$  both distinct from  $f$  such that  $f = f_1 \otimes f_2$ , then also  $\text{mwd} f \geq \text{rk}(f)$ .*

*Proof.* For the first inequality, observe that there is a monoidal decomposition  $d = ((g) \dashv\vdash_k \dashv\vdash (h))$  of  $f$  attaining the minimum of  $k = \text{rk}(f)$ . By Proposition 3.22, there are monoidal decompositions  $d_1$  and  $d_2$  of  $g$  and  $h$  whose width is bounded by their boundaries and, as a consequence, by the rank of  $f$ .

$$\begin{array}{ll}
\text{wd}(d_1) & \text{wd}(d_2) \\
\leq \min\{n, k\} + 1 & \leq \min\{k, m\} + 1
\end{array}$$

$$= k + 1$$

$$= k + 1$$

By definition of monoidal width and weight function,

$$\begin{aligned} \text{mwd}(f) & \\ &\leq \text{wd}(d) \\ &:= \max\{\text{wd}(d_1), k, \text{wd}(d_2)\} \\ &\leq \max\{k + 1, k, k + 1\} \\ &= \text{rk}(f) + 1 \end{aligned}$$

For the second inequality, suppose that there are no non-trivial  $f_1, f_2$  such that  $f = f_1 \otimes f_2$ . This means that there are no monoidal decompositions of  $f$  that start with a monoidal product node,  $(d_1 - \otimes - d_2)$ , and that all monoidal decompositions of  $f$  must either start with a composition node,  $(d_1 - \circ_k - d_2)$ , or be a leaf,  $(f)$ . Then,

$$\begin{aligned} \text{mwd}(f) & \\ &:= \min_{d \in D_f} \text{wd}(d) \\ &\geq \min\{k \in \mathbb{N} : f = g \circ_k h\} \\ &=: \text{rk}(f) \end{aligned}$$

□

From Corollary 3.21 and Lemma 3.23, we construct a minimal monoidal decomposition of morphisms in props with a zero object.

**Theorem 3.24.** *Let  $f$  be a morphism in a prop  $P$  where  $0$  is a zero object. Then,  $f$  has a unique  $\otimes$ -decomposition  $f = f_1 \otimes \dots \otimes f_k$  and its monoidal width is, up to 1, the maximum of the ranks of its factors,  $\max_{i=1, \dots, k} \text{rk}(f_i) \leq \text{mwd}(f) \leq \max_{i=1, \dots, k} \text{rk}(f_i) + 1$ .*

*Proof.* By Lemma 3.23, there are monoidal decompositions  $d_i$  of  $f_i$  of rank-bounded width,  $\text{wd}(d_i) \leq \text{rk}(f_i) + 1$ . We use these to define a decomposition  $d$  of  $f$ ,  $d = (d_1 - \otimes - \dots - (d_{k-1} - \otimes - d_k))$ , whose width is  $\text{wd}(d) := \max_{i=1, \dots, k} \text{wd}(d_i) \leq \max_{i=1, \dots, k} \text{rk}(f_i) + 1$ .

By Lemma 3.20, the factors  $f_i$  are not  $\otimes$ -decomposable. Then, the decompositions  $d_i$  are minimal and  $\text{mwd}(f_i) = \text{wd}(d_i) \geq \text{rk}(f_i)$ . By Proposition 3.19, the decomposition  $d$  is also minimal and  $\text{mwd}(f) \geq \text{wd}(d) = \max_{i=1, \dots, k} \text{rk}(f_i)$ . □





## Chapter 4

# Interlude: Two Perspectives on Graphs

Graphs and their homomorphisms form a monoidal category (Example 2.3), but not the one we will be concerned with. Our interest is in decomposing graphs as morphisms and we will instantiate monoidal width in two categorical algebras of graphs. Cospans of graphs are a well-known algebra for composing graphs along some shared vertices. Section 4.1 recalls cospans of hypergraphs and relational structures, and their syntactic presentation based on special Frobenius monoids [RSW05; BSS18]. Section 4.3 introduces the less-known algebra of graphs where the boundaries are “dangling edges” [CS15; DHS21] that allow graphs to be composed by connecting their boundary edges. Here, adjacency matrices encode the connectivity information of graphs and the syntactic presentation of this monoidal category of graphs relies on that of matrices [Zan15; Bon+19b], which we recall in Section 4.2.

These categorical algebras give canonical choices for the operations defining tree width and clique width, which we recalled in Sections 2.2 and 2.3. We derive these operations from compositions and monoidal products in cospans of hypergraphs and graphs with dangling edges, respectively.

### 4.1 Cospans of hypergraphs and relational structures

Cospans give an algebraic structure to compose systems along shared boundaries. Together with their dual algebra of spans, they are natural examples of Katis, Sabadini and Walters’ bicategories of processes [KSW97a], where cospans and spans of sets and graphs model transition systems and automata [KSW97b; Kat+00; KSW04; RSW04]. Gadducci and Heckel’s axiomatisation of double pushout graph rewriting also relies on cospans for adding boundaries to graphs [GH97; GHL99]. More recently, cospans of graphs and variations of them have been applied to modelling “open” processes like Petri nets [Fon15; BP17; BM20] and Markov processes [BFP16; CHP17].

In most of these applications, the boundaries do not retain all the computational information of the part of the system they refer to, so the boundary objects are, usually, simpler than the objects that model systems. Thus, the algebra of cospans is often restricted to a full subcategory on “simple” or “discrete” objects. This restriction can be mathematically justified with decorated [Fon15] and structured [FS07] cospans, or with free feedback monoidal categories [Bon+19a; Di+23], but, for this work, the most appropriate perspective is the characterisation of discrete cospans of graphs as a free Frobenius monoid with an additional generator [RSW05]. A very similar syntactic characterisation works more generally for discrete cospans of relational structures [BSS18]. This section reviews the category of relational structures, cospans of them and their syntactic presentation (Section 4.1). As anticipated in Example 2.22, graphs and hypergraphs are instances of relational structures where the relational signature specifies the adjacency relations between vertices. Morphisms of relational structures are functions preserving the relations and, in the case of graphs

and hypergraph, these are the usual graph and hypergraph homomorphisms.

**Definition 4.1.** For a relational signature  $\tau$ , a *relational  $\tau$ -structure*  $G$  is a finite set  $V$  with an  $\alpha_R$ -ary relation  $R^G \subseteq V^{\alpha_R}$  for each relational symbol  $R$  of arity  $\alpha_R$  in the signature  $\tau$ . A *morphism* of relational  $\tau$ -structures  $h : G \rightarrow H$  is a function  $h : V_G \rightarrow V_H$  that respects the relations: for all relational symbols  $(R, \alpha_R) \in \tau$  and all lists of elements  $v_1, \dots, v_{\alpha_R} \in V$ ,

$$R^G(v_1, \dots, v_{\alpha_R}) \Rightarrow R^H(h(v_1), \dots, h(v_{\alpha_R})).$$

Relational structures and their morphisms form a monoidal category, where disjoint union gives the monoidal structure (Proposition 4.6). This category can be described concisely as a comma category [Law63].

*Remark 4.2.* Relational  $\tau$ -structures and their morphisms are the objects and morphisms of the comma category  $(\mathbb{1} \downarrow \mathbf{T})$  for the identity functor and the functor  $\mathbf{T} : \text{FinSet} \rightarrow \text{FinSet}$  defined by the pullback below.

$$\begin{array}{ccc} \mathbf{T}(V) & \longrightarrow & V^* \\ \downarrow & \lrcorner & \downarrow \text{length} \\ \tau & \xrightarrow{\alpha} & \mathbb{N} \end{array}$$

Explicitly, elements of  $\mathbf{T}(V)$  are pairs  $(R, (v_1, \dots, v_{\alpha_R}))$  of a relational symbol  $R$  and a list of length  $\alpha_R$  of elements  $v_1, \dots, v_{\alpha_R} \in V$ . A relational structure is a function  $G : E_G \rightarrow \mathbf{T}(V_G)$  and a morphism  $h : G \rightarrow H$  is a pair of functions  $h_E : E_G \rightarrow E_H$  and  $h_V : V_G \rightarrow V_H$  such that  $G \circ h_V = h_E \circ H$ .

$$\begin{array}{ccc} E_G & \xrightarrow{h_E} & E_H \\ G \downarrow & & \downarrow H \\ \mathbf{T}(V_G) & \xrightarrow{\mathbf{T}(h_V)} & \mathbf{T}(V_H) \end{array}$$

**Proposition 4.3.** *Relational  $\tau$ -structures and their morphisms form a category  $\text{Struct}_\tau$ .*

*Proof.* As detailed in Remark 4.2,  $\text{Struct}_\tau$  is also the comma category  $(\mathbb{1} \downarrow \mathbf{T})$  of the identity functor  $\mathbb{1}_{\text{FinSet}}$  and the functor  $\mathbf{T} : \text{FinSet} \rightarrow \text{FinSet}$ . For a reference, see [Mac78, Section II.6].  $\square$

Intuitively, a cospan is a system together with two boundary maps that identify the subsystems that can communicate with the environment. Composition of cospans allows them to be composed along common substructures.

**Definition 4.4.** A *cospan* in a category  $\mathbf{C}$  is a pair of morphisms, the *legs*  $f : X \rightarrow E$  and  $g : Y \rightarrow E$ , in  $\mathbf{C}$  that share the same codomain  $E$ , the *head*.

Cospans form a monoidal category when the base category has finite colimits [Bén67].

**Proposition 4.5.** *When  $\mathbf{C}$  has finite colimits, cospans form a symmetric monoidal category  $\text{Cospan}(\mathbf{C})$  whose objects are the objects of  $\mathbf{C}$  and morphisms are cospans in  $\mathbf{C}$ . More precisely, a morphism  $X \rightarrow Y$  in  $\text{Cospan}(\mathbf{C})$  is an equivalence class of cospans  $f : X \rightarrow E \leftarrow Y : g$ , up to isomorphism of the head of the cospan. The composition of  $f : X \rightarrow E \leftarrow Y : g$  and  $h : Y \rightarrow F \leftarrow Z : l$  is given by the pushout of  $g$  and  $h$ . The monoidal product is given by component-wise coproducts.*

Relational structures have finite colimits and there is a category of cospans of them.

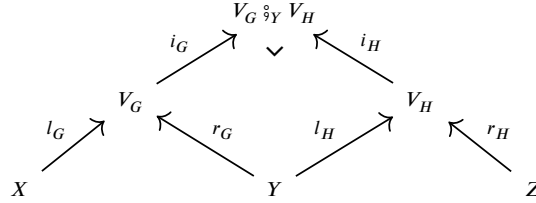
**Proposition 4.6.** *The category  $\text{Struct}_\tau$  has all finite colimits.*

*Proof.* A comma category  $(\mathbf{S} \downarrow \mathbf{T})$  for two functors  $\mathbf{S} : \mathbf{C} \rightarrow \mathbf{E}$  and  $\mathbf{T} : \mathbf{D} \rightarrow \mathbf{E}$  has all finite colimits if  $\mathbf{C}$  and  $\mathbf{D}$  have all finite colimits and the functor  $\mathbf{S}$  preserves them (see [RB88, Section 5.2] for a proof). In our case,  $\mathbf{C} = \mathbf{D} = \mathbf{FinSet}$ , which has all finite colimits and  $\mathbf{S} = \mathbb{1}$  is the identity functor, which preserves colimits. Then,  $\mathbf{Struct}_\tau$  has all finite colimits.  $\square$

This result ensures that we can consider the monoidal category of cospans of relational structures. As mentioned at the beginning of this section, the boundaries do not need to carry all the computational information of a relational structure, but it is sufficient that they record which vertices are accessible from the environment. Thus, we restrict to discrete cospans of relational structures, the full subcategory of cospans on discrete objects, i.e. sets. The legs of such a cospan point to some vertices in the relational structure that are called *sources* as they play a similar role to the sources for graphs in Bauderon and Courcelle's work [BC87].

**Definition 4.7.** The category  $\mathbf{sStruct}_\tau$  of *relational structures with sources* is the full subcategory of the monoidal category  $\mathbf{Cospan}(\mathbf{Struct}_\tau)$  on discrete structures  $D : \emptyset \rightarrow X$ . Explicitly, morphisms are cospans of functions  $l : X \rightarrow V \leftarrow Y : r$  with an apex  $\tau$ -structure  $G : E_G \rightarrow \mathbf{T}(V_G)$ .

Explicitly, the composition of two morphisms  $l_G : X \rightarrow V_G \leftarrow Y : r_G$  and  $l_H : Y \rightarrow V_H \leftarrow Z : r_H$  in  $\mathbf{sStruct}_\tau$  is the morphism  $l : X \rightarrow V_G \circledast_Y V_H \leftarrow Z : r$  defined by the pushout of  $r_G$  and  $l_H$ .



The apex of the cospan,  $V_G \circledast_Y V_H$ , is the relational structure obtained by joining  $V_G$  and  $V_H$  and identifying the vertices that are the images of the same element of the boundary  $Y$ . The legs of the composite cospan extend the legs of the original cospans:  $l := l_G \circledast_Y i_G$  and  $r := r_H \circledast_Y i_H$ . The monoidal product of two morphisms  $l : X \rightarrow V \leftarrow Y : r$  and  $l' : X' \rightarrow V' \leftarrow Y' : r'$  is their component-wise coproduct:  $l + l' : X + X' \rightarrow V + V' \leftarrow Y + Y' : r + r'$ .

Chapter 5 is dedicated to showing that monoidal width in the category  $\mathbf{sStruct}_\tau$  is equivalent to tree width. Since the tree width of a relational structure is the same as the tree width of its underlying hypergraph, it is sufficient to prove that monoidal width in the category of discrete cospans of hypergraphs is equivalent to tree width.

**Definition 4.8.** The category  $\mathbf{Cospan}(\mathbf{UHGraph})_*$  has sets as objects and discrete cospans of hypergraphs as morphisms. It is equivalent to the category  $\mathbf{sStruct}_{\tau_{hyp}}$  of discrete cospans of relational structures on the relational signature  $\tau_{hyp}$  for hypergraphs.

### A syntax for relational structures

The skeleton of the category  $\mathbf{Cospan}(\mathbf{FinSet})$  of cospans of finite sets and functions is isomorphic to the prop generated by a special Frobenius monoid [Lac04, Section 5.4], whose generators and equations are in Figure 4.1. The syntactic presentation of discrete cospans of relational structures builds on this characterisation and only adds a generator for each relational symbol  $R$  in the relational signature  $\tau$ .

**Definition 4.9.** The category  $\mathbf{sFrob}$  is the prop generated by a special Frobenius monoid, whose generators and equations are in Figure 4.1.

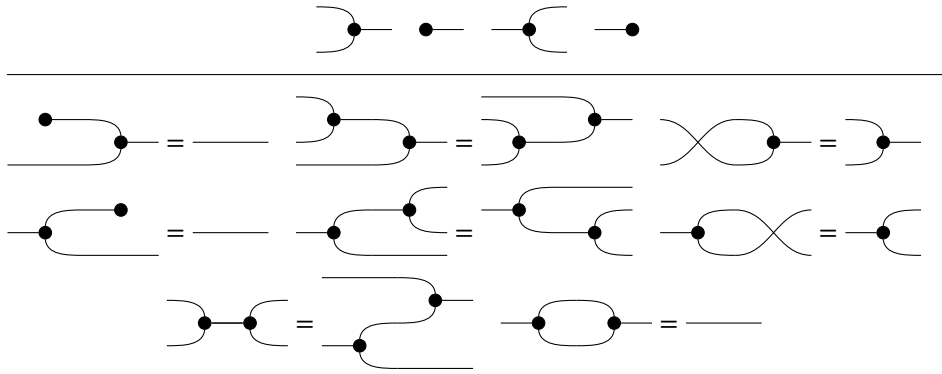


Figure 4.1: Generators and equations for a special Frobenius monoid.

**Proposition 4.10** ([Lac04]). *The skeleton of  $\text{Cospan}(\text{FinSet})$  is isomorphic to the prop  $\text{sFrob}$  generated by a special Frobenius monoid.*

The prop of relational structures with sources is obtained by freely adding a generator  $e_R : \alpha_R \rightarrow 0$  for each  $(R, \alpha_R) \in \tau$  to the prop  $\text{sFrob}$ .

**Definition 4.11.** Given a relational signature  $\tau$ , the category  $\text{LHedge}_\tau$  is the free prop generated by a “labelled hyperedge” generator  $e_R : \alpha_R \rightarrow 0$  for every relational symbol  $R$  of arity  $\alpha_R$  in the signature  $\tau$  (Figure 4.2).

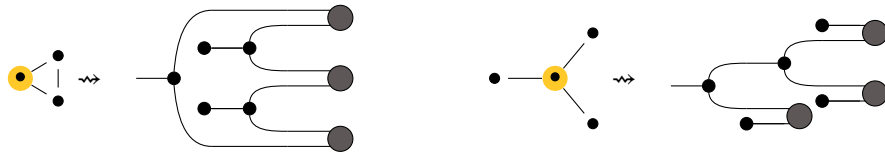
$$\alpha_R \vdots \textcircled{R} \quad \text{for every } (R, \alpha_R) \in \tau$$

Figure 4.2: The labelled hyperedge generators.

**Definition 4.12.** For a relational signature  $\tau$ , the prop  $\text{sFrob}_\tau := \text{sFrob} + \text{LHedge}_\tau$  is the coproduct of the prop  $\text{sFrob}$  generated by a special Frobenius monoid and the prop  $\text{LHedge}_\tau$  generated by the labelled hyperedges in  $\tau$ .

The relational signature for graphs  $\tau_{gr}$  contains a single symbol  $\text{---}\bullet$  and morphisms in  $\text{sFrob}_{\tau_{gr}}$  are graphs with sources.

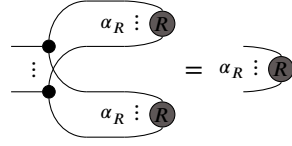
*Example 4.13.* The 3-clique with one source and the 3-star with one source are morphisms  $1 \rightarrow 0$  in  $\text{sFrob}_{\tau_{gr}}$ .



**Remark 4.14.** We can impose additional equations to  $\text{sFrob}_\tau$  to constrain the behaviour of some relational symbols. For a symmetric relational symbol  $R$ , we impose that  $p \circ e_R = e_R$ , for every permutation  $p$  of the  $\alpha_R$  inputs of  $e_R$ .

$$\textcircled{p} \circ \alpha_R \vdots \textcircled{R} = \alpha_R \vdots \textcircled{R}$$

If we want to impose that there may not be parallel edges of the same type  $R$ , we add that  $\alpha_R \circ (e_R \otimes e_R) = e_R$ .



The prop  $\text{sFrob}_\tau$  is a syntax for relational structures with sources [BSS18]. This result relies on previous characterisations of the category of discrete cospans of graphs with Frobenius monoids [GH97; GHL99; RSW05].

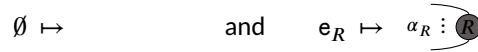
**Theorem 4.15** ([BSS18, Theorem 31]). *The category  $\text{sStruct}_\tau$  of  $\tau$ -structures with sources is isomorphic to the free special Frobenius prop  $\text{sFrob}_\tau$  on the signature  $\tau$ .*

### The operations for tree width

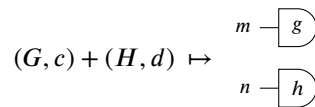
This section takes the operations for tree width of Definition 2.53 introduced by Bauderon and Courcelle [BC87; Cou90] and examines them through a categorical lens. We derive these operations from compositions and monoidal products in the category  $\text{sFrob}_\tau$  of relational structures with sources. This correspondence defines inductively a function from structures with  $n$  constants to morphisms of type  $n \rightarrow 0$  in  $\text{sFrob}_\tau$ , which maps a structure  $(G, c)$  with  $n$  constants to the morphism  $g : n \rightarrow 0$  in  $\text{sFrob}_\tau$  that corresponds to the discrete cospan of structures  $g = c : n \rightarrow G \leftarrow 0 : i^1$ .

The categorical structure clarifies the relationships between all the slightly different versions of the operations for tree width [BC87; Cou90; CM02]. While it is not difficult to check, with their usual definitions, that these different variations are equivalent, this becomes even more apparent when seen from the categorical perspective. This perspective also gives canonicity to one choice: the operations that define tree width are composition and monoidal product in the monoidal category of relational structures with sources. Chapter 5 is devoted to prove this in detail.

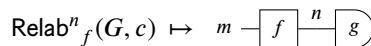
The generating structures of the algebraic tree decompositions correspond to specific morphisms in  $\text{sFrob}_\tau$ . The empty structure with no constants  $\emptyset$  is the identity morphism on the monoidal unit  $1_I$ , and the structure  $e_R$  with  $\alpha_R$  constants is the generator  $e_R : \alpha_R \rightarrow 0$ .



The operations are derived from the categorical structure. The disjoint union  $(G, c) + (H, d)$  of structures  $(G, c)$  with  $m$  constants and  $(H, d)$  with  $n$  constants is their monoidal product as morphisms  $g \otimes h : m+n \rightarrow 0$ .

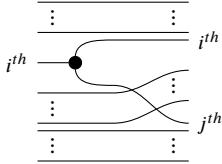


The redefinition of constants  $\text{Relab}_f^n(G, c)$  by a function  $f$  is obtained by precomposing the corresponding morphism  $g$  with the cospan  $f : m \rightarrow n \leftarrow n : !$ . This cospan is composed only of the monoid operations, i.e. it is covariantly lifted from the function  $f : m \rightarrow n$ .

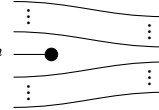


<sup>1</sup>We indicate with  $i_A : 0 \rightarrow A$  the unique morphism from the initial object  $0$  to an object  $A$ . Similarly, we indicate with  $!_A : A \rightarrow 1$  the unique morphism from an object  $A$  to the terminal object  $1$

Similarly, the fusion of the constants  $i$  and  $j$ ,  $\text{Fuse}_{i,j}^n(G, c)$ , is obtained by precomposing with the cospan  $d_{i,j}^{\text{op}} = \mathbb{1} : n \rightarrow n \leftarrow n+1 : d_{i,j}$ . This cospan is contravariantly lifted from the function  $d_{i,j} : n+1 \rightarrow n$  defined as  $d_{i,j}(k) = k$  if  $k < j$ ,  $d_{i,j}(j) = i$  and  $d_{i,j}(k) = k-1$  if  $k > j$ . The cospan  $d_{i,j}^{\text{op}}$  is composed only of symmetries and a copy morphism that joins the  $i^{\text{th}}$  and  $j^{\text{th}}$  outputs.

$$\text{Fuse}_{i,j}^n(G, c) \mapsto n+1 \text{ --- } \boxed{d_{i,j}^{\text{op}}} \text{ --- } n \text{ --- } \boxed{g} \quad \text{where } d_{i,j}^{\text{op}} :=$$


The addition of a constant  $i$ ,  $\text{Vert}_i^n(G, c)$  is also a precomposition. We compose the cospan  $a_i^{\text{op}} = \mathbb{1} : n+1 \rightarrow n+1 \leftarrow n : a_i$  with the morphism  $g$  that corresponds to the structure  $(G, c)$ . As with the fusion of constants, the cospan  $a_i^{\text{op}}$  is contravariantly lifted from the function  $a_i : n \rightarrow n+1$  defined as  $a_i(k) = k$  if  $k < i$  and  $a_i(k) = k+1$  if  $k \geq i$ . The cospan  $a_i^{\text{op}}$  is composed only of identities and one discard morphism on the  $i^{\text{th}}$  input.

$$\text{Vert}_i^n(G, c) \mapsto n \text{ --- } \boxed{a_i^{\text{op}}} \text{ --- } n+1 \text{ --- } \boxed{g} \quad \text{where } a_i^{\text{op}} :=$$


The operations of redefinition, fusion and addition of constants together are as expressive as the operation of precomposition with edge-less morphisms in  $\text{sFrob}_r$ . In fact, these operations can construct all morphisms  $n \rightarrow 0$  in the monoidal category of relational structures with sources.

## 4.2 Matrices

Matrices over the natural numbers are often used to encode the adjacency relation of graphs and are the basis for the graph algebra presented in Section 4.3. This section recalls Proposition 4.18, a result that characterises the algebra of matrices in terms of the generators and equations of a bialgebra (Figure 4.3). The characterisation of the algebra of graphs in Section 4.3, Theorem 4.44, relies on this result.

Matrices are the morphisms of a prop.

**Definition 4.16.** The *category of matrices*  $\text{Mat}_{\mathbb{N}}$  is the prop whose morphisms  $n \rightarrow m$  are  $m$  by  $n$  matrices. Composition is the usual product of matrices and the monoidal product is the biproduct of matrices  $A \oplus B := \begin{pmatrix} A & 0 \\ 0 & B \end{pmatrix}$ .

### A syntax for matrices

The syntax for the prop of matrices is given by a commutative monoid  $(\oplus, \circ)$ , interpreted as adding and zero, and a cocommutative comonoid  $(\dashv, \dashv)$ , interpreted as copying and discarding. These interact according to the laws of a bialgebra.

**Definition 4.17.** The prop  $\text{Bialg}$  is freely generated by a bialgebra, whose generators and equations are given in Figure 4.3.

The free prop generated by a bialgebra is isomorphic to the prop of matrices. Proofs of this result can be found in Zanasi's PhD thesis [Zan15, Proposition 3.9] and in [BSZ17, Proposition 3.7].

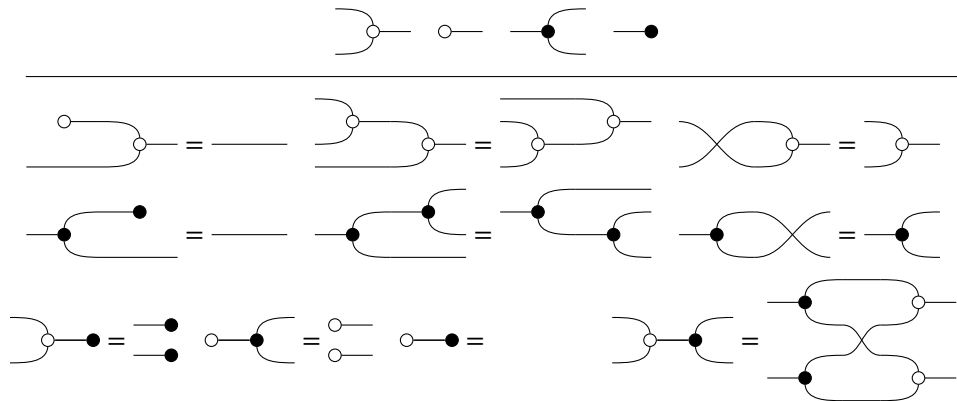
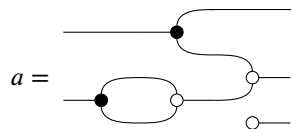


Figure 4.3: Generators and equations of a bialgebra.

**Proposition 4.18** ([Zan15]). *There is an isomorphism of categories  $\mathbf{Mat} : \mathbf{Bialg} \rightarrow \mathbf{Mat}_{\mathbb{N}}$ .*

Every morphism  $f : n \rightarrow m$  in  $\mathbf{Bialg}$  corresponds to a matrix  $A = \mathbf{Mat}(f) \in \mathbf{Mat}_{\mathbb{N}}(m, n)$ : we can read the  $(i, j)$ -entry of  $A$  off the diagram of  $f$  by counting the number of paths directed from the  $j$ -th input to the  $i$ -th output. These paths do not include paths that “go back” through a multiplication or comultiplication node.

*Example 4.19.* The matrix  $A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 0 \end{pmatrix} \in \mathbf{Mat}_{\mathbb{N}}(3, 2)$  corresponds to the morphism  $a : 2 \rightarrow 3$  below. The columns are the inputs and the rows are the outputs: the two distinct paths from the second input to the second output and the absence of paths from the same input to the third output are recorded by a 2 in the entry  $(2, 2)$  and a 0 in the entry  $(3, 2)$  of the matrix  $A$ .



*Remark 4.20.* By Theorem 3.24, the monoidal width of a matrix  $A = A_1 \oplus \dots \oplus A_b$  is the maximal rank of its blocks,

$$\text{mwd}(A) = \max_{i=1, \dots, b} \text{rk}(A_i),$$

because the monoidal unit  $0$  is also a zero object.

### 4.3 Graphs with dangling edges

This section introduces the prop of *graphs with dangling edges*. Morphisms represent graphs with additional “dangling edges” and composition joins two graphs by connecting their dangling edges. We define this algebra explicitly (Definition 4.25) and give an equivalent syntactic presentation (Definition 4.42). We show their isomorphism by finding a normal form for morphisms in the syntactic presentation. The diagram below summarises the proof strategy: Proposition 4.30 shows that the prop of graphs with dangling edges,  $\mathbf{MGraph}$ , is the coproduct of a prop of adjacency matrices,  $\mathbf{MAAdj}$ , and a prop of bounded permutations,  $\mathbf{boundP}$ ; Theorem 4.39 and Proposition 4.41 give equivalent syntactic descriptions of adjacency matrices,  $\mathbf{Adj}$ , and bounded

permutations,  $\text{Vert}$ , based on the bialgebra characterisation of matrices; finally, the syntactic presentation of graphs with dangling edges is defined as their coproduct,  $\text{BGraph} := \text{Adj} + \text{Vert}$ .

$$\begin{array}{ccccc}
 \text{MAdj} & \xrightarrow{i_1} & \text{MGraph} & \xleftarrow{i_2} & \text{boundP} \\
 \text{Theorem 4.39} \downarrow & & \downarrow \text{Theorem 4.44} & & \downarrow \text{Proposition 4.41} \\
 \text{Adj} & \longrightarrow & \text{BGraph} & \longleftarrow & \text{Vert}
 \end{array}$$

The algebra of graphs with dangling edges relies on adjacency matrices to encode the connectivity of vertices. These are matrices quotiented by an equivalence relation that captures that there are different ways of expressing the same connectivity information: if there are two edges between vertices  $i$  and  $j$  of a graph  $G$ , then this can be recorded in the entry  $(i, j)$  or  $(j, i)$  as long as their sum is 2.

**Definition 4.21.** An *adjacency matrix*  $[G]$  is an equivalence class of square matrices  $G \in \text{Mat}_{\mathbb{N}}(m, m)$  over the natural numbers, where the equivalence relation is  $[G] = [H]$  iff  $G + G^{\top} = H + H^{\top}$ .

Adjacency matrices on  $m$  vertices are the morphisms  $0 \rightarrow m$  of a prop where generic morphisms represent adjacency matrices “with inputs”. These are an adjacency matrix together with a matrix of compatible dimensions that connects the inputs to the adjacency matrix. This prop is defined in [CS15], where it gives an algebra for simple graphs. Our graph algebra captures multi-graphs but follows a similar idea.

**Proposition 4.22** ([CS15]). *There is a prop  $\text{MAdj}$  where morphisms  $\alpha : n \rightarrow m$  are pairs  $\alpha = (B, [G])$  of an  $m$  by  $n$  matrix  $B \in \text{Mat}_{\mathbb{N}}(m, n)$  and an  $m$  by  $m$  adjacency matrix  $[G]$ .*

*Proof.* The composition of two morphisms  $(B, [G]) : n \rightarrow m$  and  $(C, [H]) : m \rightarrow l$  is defined as  $(B, [G]) \circ (C, [H]) := (C \cdot B, [C \cdot G \cdot C^{\top} + H]) : n \rightarrow l$ . The identity on  $n$  is  $(1_n, [0])$ . The monoidal product on objects is addition, while on morphisms it is the component-wise biproduct of matrices,  $(B, [G]) \otimes (B', [G']) := (B \oplus B', [G \oplus G'])$ , with monoidal unit  $0$ . Composition is well-defined on equivalence classes of adjacency matrices. Suppose  $(B, [G]) = (B, [G'])$  and  $(C, [H]) = (C, [H'])$ . This means that  $G + G^{\top} = G' + (G')^{\top}$  and  $H + H^{\top} = H' + (H')^{\top}$ .

$$\begin{aligned}
 & (CGC^{\top} + H) + (CGC^{\top} + H)^{\top} \\
 &= CGC^{\top} + CG^{\top}C^{\top} + H + H^{\top} \\
 &= C(G + G^{\top})C^{\top} + H + H^{\top} \\
 &= C(G' + (G')^{\top})C^{\top} + H' + (H')^{\top} \\
 &= CG'C^{\top} + C(G')^{\top}C^{\top} + H' + (H')^{\top} \\
 &= (CG'C^{\top} + H') + (CG'C^{\top} + H')^{\top}
 \end{aligned}$$

Then, composition preserves equivalence of adjacency matrices.

$$\begin{aligned}
 & (B, [G]) \circ (C, [H]) \\
 &:= (C \cdot B, [C \cdot G \cdot C^{\top} + H]) \\
 &= (C \cdot B, [C \cdot G' \cdot C^{\top} + H']) \\
 &:= (B, [G']) \circ (C, [H'])
 \end{aligned}$$

For the monoidal product, it is easier to see that it preserves equivalence of adjacency matrices because, if  $G + G^{\top} = G' + (G')^{\top}$  and  $H + H^{\top} = H' + (H')^{\top}$ , then  $(G \oplus H) + (G \oplus H)^{\top} = (G' \oplus H') + (G' \oplus H')^{\top}$ .



For  $(A, [F]) : p \rightarrow n$ ,  $(B, [G]) : n \rightarrow m$  and  $(C, [H]) : m \rightarrow l$ , we show that composition is associative.

$$\begin{aligned}
& ((A, [F]) \circ (B, [G])) \circ (C, [H]) && (A, [F]) \circ ((B, [G]) \circ (C, [H])) \\
& = (BA, [BFB^T + G]) \circ (C, [H]) && = (A, [F]) \circ (CB, [CGC^T + H]) \\
& = (CBA, [C(BFB^T + G)C^T + H]) && = (CBA, [CBF(CB)^T + CGC^T + H]) \\
& = (CBA, [CBF(CB)^T + CGC^T + H])
\end{aligned}$$

For  $(B, [G]) : n \rightarrow m$ , we show that composition is unital.

$$\begin{aligned}
& (B, [G]) \circ (\mathbb{1}_m, [0]) && (\mathbb{1}_n, [0]) \circ (B, [G]) \\
& = (\mathbb{1}_m \cdot B, [\mathbb{1}_m \cdot G \cdot \mathbb{1}_m^T + 0]) && = (B \cdot \mathbb{1}_n, [B \cdot 0 \cdot B^T + G]) \\
& = (B, [G]) && = (B, [G])
\end{aligned}$$

For  $(B, [G]) : n \rightarrow m$ ,  $(C, [H]) : m \rightarrow l$ ,  $(B', [G']) : n' \rightarrow m'$  and  $(C', [H']) : m' \rightarrow l'$ , we show that the monoidal product preserves their composition.

$$\begin{aligned}
& ((B, [G]) \otimes (B', [G'])) \circ ((C, [H]) \otimes (C', [H'])) \\
& = (B \oplus B', [G \oplus G']) \circ (C \oplus C', [H \oplus H']) \\
& = ((C \oplus C')(B \oplus B'), [(C \oplus C')(G \oplus G')(C \oplus C')^T + (H \oplus H')]) \\
& = ((CB) \oplus (C'B'), [(CGC^T) \oplus (C'G'C'^T) + (H \oplus H')]) \\
& = ((CB) \oplus (C'B'), [(CGC^T + H) \oplus (C'G'(C')^T + H')]) \\
& = (CB, [CGC^T + H]) \otimes (C'B', [C'G'(C')^T + H']) \\
& = ((B, [G]) \circ (C, [H])) \otimes ((B', [G']) \circ (C', [H']))
\end{aligned}$$

The monoidal product preserves identities.

$$\begin{aligned}
& (\mathbb{1}_n, [0]) \oplus (\mathbb{1}_{n'}, [0]) \\
& = (\mathbb{1}_n \oplus \mathbb{1}_{n'}, [0 \oplus 0]) \\
& = (\mathbb{1}_{n+n'}, [0])
\end{aligned}$$

The monoidal product is associative and unital because the objects are natural numbers and the monoidal product is addition.  $\square$

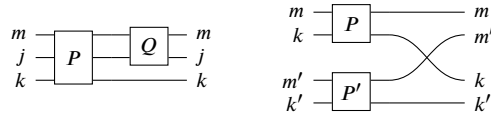
The ordering of vertices in a graph is immaterial, but adjacency matrices fix one. Graphs are adjacency matrices where the vertices can be arbitrarily permuted, so they are obtained by adding to the prop of adjacency matrices the possibility of permuting some of the wires, those connected to the vertices. We introduce the prop of bounded permutations to capture this aspect: morphisms are permutations where some of the outputs can be freely permuted.

**Definition 4.23.** A bounded permutation  $p = (k, P)$  is a pair of a natural number  $k \in \mathbb{N}$  and a permutation matrix  $P \in \text{Mat}_{\mathbb{N}}(m+k, m+k)$ . Two bounded permutations  $p = (k, P)$  and  $q = (k, Q)$  are equivalent if there is a permutation  $\sigma \in \text{Mat}_{\mathbb{N}}(k, k)$  such that  $P = \begin{pmatrix} \mathbb{1}_m & 0 \\ 0 & \sigma \end{pmatrix} \cdot Q$ .

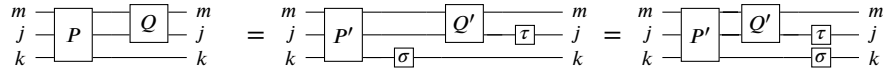
In a bounded permutation  $(k, P)$ , the number  $k$  gives the number of outputs that are “bounded” and can, thus, be permuted without changing the morphism. Bounded permutations are the morphisms of a prop.

**Proposition 4.24.** *Bounded permutations form a prop  $\text{boundP}$  where morphisms  $p : m + k \rightarrow m$  are equivalence classes of bounded permutations  $p = (k, P)$ .*

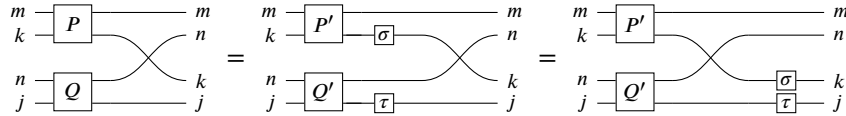
*Proof.* The composition of two bounded permutations  $(k, P) : m + j + k \rightarrow m + j$  and  $(j, Q) : m + j \rightarrow m$  is defined as  $(k, P) \circ (j, Q) := (k + j, (Q \oplus \mathbb{1}_k) \cdot P)$ , and the identity morphism on  $m$  is  $(0, \mathbb{1}_m) : m \rightarrow m$ . The monoidal product on objects is addition, the monoidal unit is 0 and, for two bounded permutations  $(k, P) : m + k \rightarrow m$  and  $(k', P') : m' + k' \rightarrow m'$ , their monoidal product is  $(k, P) \otimes (k', P') := (k + k', (\mathbb{1}_m \oplus \sigma_{k, m'} \oplus \mathbb{1}_{k'}) \cdot (P \oplus P'))$ , where  $\sigma_{k, m'}$  is the permutation matrix that swaps the first  $k$  inputs with the remaining  $m'$  inputs. Thanks to the string diagrammatic syntax for matrices, the permutation matrices associated to a composition and a monoidal product are, in string diagrams,



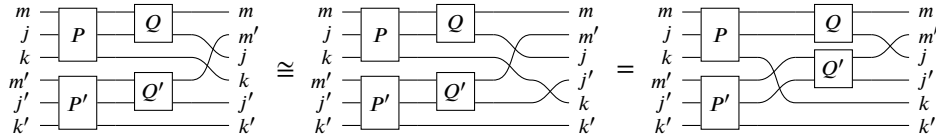
With this, it easy to see that composition is associative and unital. Composition is well-defined because, if  $(k, P) = (k, P')$  and  $(j, Q) = (j, Q')$ , then  $P = (\mathbb{1}_{m+j} \oplus \sigma) \cdot P'$ ,  $Q = (\mathbb{1}_m \oplus \tau) \cdot Q'$ ,



and  $(\mathbb{1}_k \oplus Q') \cdot P' = (\mathbb{1}_m \oplus \tau \oplus \sigma) \cdot ((\mathbb{1}_k \oplus Q') \cdot P')$ . So  $(k, P) \circ (j, Q) = (k, P') \circ (j, Q')$ . The monoidal product is also well-defined because, if  $(k, P) = (k, P')$  and  $(j, Q) = (j, Q')$ , then  $P = (\mathbb{1}_m \oplus \sigma) \cdot P'$ ,  $Q = (\mathbb{1}_n \oplus \tau) \cdot Q'$ ,



and  $(\mathbb{1}_m \oplus \sigma_{k, n} \oplus \mathbb{1}_j) \cdot (P \oplus Q) = (\mathbb{1}_{m+n} \oplus \sigma \oplus \tau) \cdot (\mathbb{1}_m \oplus \sigma_{k, n} \oplus \mathbb{1}_j) \cdot (P' \oplus Q')$ . So  $(k, P) \otimes (j, Q) = (k, P') \otimes (j, Q')$ . The monoidal product is a functor:  $((k, P) \otimes (k', P')) \circ ((j, Q) \otimes (j', Q')) = ((k, P) \circ (j, Q)) \otimes ((k', P') \circ (j', Q'))$  because their matrices are equivalent up to permuting the “bounded wires”.



The monoidal product is strictly associative and unital because, on objects, it is addition of natural numbers.  $\square$

Graphs with dangling edges inherit the algebra of adjacency matrices and mix it with that of bounded permutations. In fact, Proposition 4.30 shows that the prop  $\text{MGraph}$  of graphs with dangling edges is the coproduct of  $\text{MAdj}$  and  $\text{boundP}$ . Graphs with dangling edges have three connectivity points: the left and right boundaries, and the vertices. These are connected between each other and themselves with five matrices.

**Definition 4.25.** Graphs with dangling edges are tuples  $g = ([G], L, R, P, [S])$ , where each matrix encodes part of the edges:

- $G \in \text{Mat}_{\mathbb{N}}(k, k)$  the edges of the graph, with  $k$  the number of vertices;

- $L \in \text{Mat}_{\mathbb{N}}(k, n)$  the dangling edges to the left boundary;
- $R \in \text{Mat}_{\mathbb{N}}(k, m)$  the dangling edges to the right boundary;
- $P \in \text{Mat}_{\mathbb{N}}(m, n)$  the passing edges from the left to the right boundary; and
- $S \in \text{Mat}_{\mathbb{N}}(m, m)$  the edges from the right boundary to itself.

Two graphs with dangling edges  $g = ([G], L, R, P, [S])$  and  $g' = ([G'], L', R', P', [S'])$  are equivalent if there is a permutation matrix  $\sigma \in \text{Mat}_{\mathbb{N}}(k, k)$  such that

$$g' = ([\sigma \cdot G \cdot \sigma^\top], \sigma \cdot L, \sigma \cdot R, P, [S]).$$

The equivalence relation of graphs with dangling edges captures that the order of the vertices is immaterial. Graphs with dangling edges can be composed and are the morphisms of a prop.

**Proposition 4.26.** *Graphs with dangling edges form a prop MGraph where morphisms  $g : n \rightarrow m$  are equivalence classes of graphs with dangling edges  $g = ([G], L, R, P, [S])$  as in Definition 4.25.*

*Proof.* Given two graphs with dangling edges  $g : n \rightarrow m$  and  $h : m \rightarrow l$ , with  $g = ([G], L_g, R_g, P_g, [S_g])$  and  $h = ([H], L_h, R_h, P_h, [S_h])$ , their composition  $g \circ h : n \rightarrow l$  is

$$\left( \left[ \begin{pmatrix} G & R_g L_h^\top \\ 0 & H + L_h S_g L_h^\top \end{pmatrix} \right], \begin{pmatrix} L_g \\ L_h P_g \end{pmatrix}, \begin{pmatrix} R_g P_h^\top \\ R_h + L_h (S_g + S_g^\top) P_h^\top \end{pmatrix}, P_h P_g, [S_h + P_h S_g P_h^\top] \right).$$

Composition is associative.

$$\begin{aligned} & (f \circ g) \circ h \\ &= \left( \left[ \begin{pmatrix} F & R_f L_g^\top \\ 0 & G + L_g S_f L_g^\top \end{pmatrix} \right], \begin{pmatrix} L_f \\ L_g P_f \end{pmatrix}, \begin{pmatrix} R_f P_g^\top \\ R_g + L_g (S_f + S_f^\top) P_g^\top \end{pmatrix}, P_g P_f, [S_g + P_g S_f P_g^\top] \right) \circ h \\ &= \left( \left[ \begin{pmatrix} F & R_f L_g^\top & R_f P_g^\top L_h^\top \\ 0 & G + L_g S_f L_g^\top & (R_g + L_g (S_f + S_f^\top) P_g^\top) L_h^\top \\ 0 & 0 & H + L_h (S_g + P_g S_f P_g^\top) L_h^\top \end{pmatrix} \right], \begin{pmatrix} L_f \\ L_g P_f \end{pmatrix}, \begin{pmatrix} R_f P_g^\top P_h^\top \\ (R_g + L_g (S_f + S_f^\top) P_g^\top) P_h^\top \\ R_h + L_h (S_g + P_g S_f P_g^\top + S_g^\top + P_g S_f^\top P_g^\top) P_h^\top \end{pmatrix}, \right. \\ & \quad \left. P_h P_g P_f, [S_h + P_h (S_g + P_g S_f P_g^\top) P_h^\top] \right) \\ &= \left( \left[ \begin{pmatrix} F & R_f L_g^\top & R_f P_g^\top L_h^\top \\ 0 & G + L_g S_f L_g^\top & (R_g + L_g S_f P_g^\top) L_h^\top \\ 0 & L_h P_g S_f L_g^\top & H + L_h (S_g + P_g S_f P_g^\top) L_h^\top \end{pmatrix} \right], \begin{pmatrix} L_f \\ L_g P_f \end{pmatrix}, \begin{pmatrix} R_f P_g^\top P_h^\top \\ R_g P_h^\top + L_g (S_f + S_f^\top) P_g^\top P_h^\top \\ R_h + L_h (S_g + S_g^\top + P_g (S_f + S_f^\top) P_g^\top) P_h^\top \end{pmatrix}, \right. \\ & \quad \left. P_h P_g P_f, [S_h + P_h S_g P_h^\top + P_h P_g S_f P_g^\top P_h^\top] \right) \\ &= f \circ \left( \left[ \begin{pmatrix} G & R_g L_h^\top \\ 0 & H + L_h S_g L_h^\top \end{pmatrix} \right], \begin{pmatrix} L_g \\ L_h P_g \end{pmatrix}, \begin{pmatrix} R_g P_h^\top \\ R_h + L_h (S_g + S_g^\top) P_h^\top \end{pmatrix}, P_h P_g, [S_h + P_h S_g P_h^\top] \right) \\ &= f \circ (g \circ h) \end{aligned}$$

Composition is unital.

$$\begin{aligned} & g \circ \mathbb{1}_m \\ &= \left( \left[ \begin{pmatrix} G & R_g \mathbb{1}_m \\ \mathbb{1}_m & S_g \mathbb{1}_m \end{pmatrix} \right], \begin{pmatrix} L_g \\ \mathbb{1}_m P_g \end{pmatrix}, \begin{pmatrix} R_g \mathbb{1}_m \\ \mathbb{1}_m + \mathbb{1}_m (S_g + S_g^\top) \mathbb{1}_m \end{pmatrix}, \mathbb{1}_m P_g, [\mathbb{0} + \mathbb{1}_m S_g \mathbb{1}_m] \right) \end{aligned}$$

$$\begin{aligned}
&= ([G], L_g, R_g, P_g, [S_g]) \\
&= g \\
&\mathbb{1}_n \circ g \\
&= \left( \left[ \begin{pmatrix} \binom{\cdot}{\cdot} & \mathbb{1}_n L_g^\top \\ \text{in } G+L_g & 0 L_g^\top \end{pmatrix} \right], \left( \begin{matrix} \mathbb{1}_n \\ L_g \mathbb{1}_n \end{matrix} \right), \left( \begin{matrix} \mathbb{1}_n P_g^\top \\ R_g+L_g & 0 P_g^\top \end{matrix} \right), P_g \mathbb{1}_n, [S_g + P_g 0 P_g^\top] \right) \\
&= ([G], L_g, R_g, P_g, [S_g]) \\
&= g
\end{aligned}$$

Given two morphisms  $g = ([G], L, R, P, [S])$  and  $g' = ([G'], L', R', P', [S'])$ , their monoidal product is

$$g \otimes g' := ([G \oplus G'], L \oplus L', R \oplus R', P \oplus P', [S \oplus S']).$$

The monoidal product is functorial.

$$\begin{aligned}
&(g \circ h) \otimes (g' \circ h') \\
&= \left( \left[ \begin{pmatrix} G & R_g L_h^\top \\ 0 & H+L_h S_g L_h^\top \end{pmatrix} \right], \left( \begin{matrix} L_g \\ L_h P_g \end{matrix} \right), \left( \begin{matrix} R_g P_h^\top \\ R_h+L_h(S_g+S_g^\top) P_h^\top \end{matrix} \right), P_h P_g, [S_h + P_h S_g P_h^\top] \right) \\
&\quad \otimes \left( \left[ \begin{pmatrix} G' & R'_g (L'_h)^\top \\ 0 & H'+L'_h S'_g (L'_h)^\top \end{pmatrix} \right], \left( \begin{matrix} L'_g \\ L'_h P'_g \end{matrix} \right), \left( \begin{matrix} R'_g (P'_h)^\top \\ R'_h+L'_h(S'_g+(S'_g)^\top)(P'_h)^\top \end{matrix} \right), P'_h P'_g, [S'_h + P'_h S'_g (P'_h)^\top] \right) \\
&= \left( \left[ \begin{pmatrix} G & R_g L_h^\top \\ 0 & H+L_h S_g L_h^\top \end{pmatrix} \oplus \begin{pmatrix} G' & R'_g (L'_h)^\top \\ 0 & H'+L'_h S'_g (L'_h)^\top \end{pmatrix} \right], \right. \\
&\quad \left. \left( \begin{matrix} L_g \\ L_h P_g \end{matrix} \right) \oplus \left( \begin{matrix} L'_g \\ L'_h P'_g \end{matrix} \right), \left( \begin{matrix} R_g P_h^\top \\ R_h+L_h(S_g+S_g^\top) P_h^\top \end{matrix} \right) \oplus \left( \begin{matrix} R'_g (P'_h)^\top \\ R'_h+L'_h(S'_g+(S'_g)^\top)(P'_h)^\top \end{matrix} \right), \right. \\
&\quad \left. P_h P_g \oplus P'_h P'_g, [S_h + P_h S_g P_h^\top] \oplus [S'_h + P'_h S'_g (P'_h)^\top] \right) \\
&= \left( \left[ \begin{pmatrix} G & 0 & R_g L_h^\top & 0 \\ 0 & G' & 0 & R'_g (L'_h)^\top \\ 0 & 0 & H+L_h S_g L_h^\top & 0 \\ 0 & 0 & 0 & H'+L'_h S'_g (L'_h)^\top \end{pmatrix} \right] \tau^\top, \tau \begin{pmatrix} L_g & 0 \\ 0 & L'_g \\ L_h P_g & 0 \\ 0 & L'_h P'_g \end{pmatrix}, \tau \begin{pmatrix} R_g P_h^\top & 0 \\ 0 & R'_g (P'_h)^\top \\ R_h+L_h(S_g+S_g^\top) P_h^\top & 0 \\ 0 & R'_h+L'_h(S'_g+(S'_g)^\top)(P'_h)^\top \end{pmatrix} \right) \\
&\quad \left( \begin{pmatrix} P_h P_g & 0 \\ 0 & P'_h P'_g \end{pmatrix}, \left[ \begin{pmatrix} S_h+P_h S_g P_h^\top & 0 \\ 0 & S'_h+P'_h S'_g (P'_h)^\top \end{pmatrix} \right] \right) \\
&\cong \left( \left[ \begin{pmatrix} G & 0 & R_g L_h^\top & 0 \\ 0 & G' & 0 & R'_g (L'_h)^\top \\ 0 & 0 & H+L_h S_g L_h^\top & 0 \\ 0 & 0 & 0 & H'+L'_h S'_g (L'_h)^\top \end{pmatrix} \right], \left( \begin{matrix} L_g & 0 \\ 0 & L'_g \\ L_h P_g & 0 \\ 0 & L'_h P'_g \end{matrix} \right), \begin{pmatrix} R_g P_h^\top & 0 \\ 0 & R'_g (P'_h)^\top \\ R_h+L_h(S_g+S_g^\top) P_h^\top & 0 \\ 0 & R'_h+L'_h(S'_g+(S'_g)^\top)(P'_h)^\top \end{pmatrix}, \right. \\
&\quad \left. \left( \begin{pmatrix} P_h P_g & 0 \\ 0 & P'_h P'_g \end{pmatrix}, \left[ \begin{pmatrix} S_h+P_h S_g P_h^\top & 0 \\ 0 & S'_h+P'_h S'_g (P'_h)^\top \end{pmatrix} \right] \right) \\
&= \left( \left[ \begin{pmatrix} G & 0 \\ 0 & G' \end{pmatrix} \right], \left( \begin{matrix} L_g & 0 \\ 0 & L'_g \end{matrix} \right), \left( \begin{matrix} R_g & 0 \\ 0 & R'_g \end{matrix} \right), \left( \begin{matrix} P_g & 0 \\ 0 & P'_g \end{matrix} \right), \left[ \begin{pmatrix} S_g & 0 \\ 0 & S'_g \end{pmatrix} \right] \right)
\end{aligned}$$

$$\begin{aligned} & \circlearrowleft \left( \left[ \begin{pmatrix} H & 0 \\ 0 & H' \end{pmatrix} \right], \begin{pmatrix} L_h & 0 \\ 0 & L'_h \end{pmatrix}, \begin{pmatrix} R_h & 0 \\ 0 & R'_h \end{pmatrix}, \begin{pmatrix} P_h & 0 \\ 0 & P'_h \end{pmatrix}, \left[ \begin{pmatrix} S_h & 0 \\ 0 & S'_h \end{pmatrix} \right] \right) \\ & = (g \otimes g') \circlearrowleft (h \otimes h') \end{aligned}$$

where  $\tau = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$  permutes the order of the vertices.  $\square$

Proposition 4.30 shows the universal property of MGraph as a coproduct. The intermediate results in Lemmas 4.27 to 4.29 define the inclusions and show the factorisation system of MGraph. The inclusions indicate that adjacency matrices and bounded permutations are graphs with dangling edges of a particular shape.

**Lemma 4.27.** *There are two homomorphisms of props  $\iota_1 : \text{MAAdj} \rightarrow \text{MGraph}$  and  $\iota_2 : \text{boundP} \rightarrow \text{MGraph}$ .*

*Proof.* The inclusions are identity on objects and, on morphisms, are defined by

$$\begin{aligned} \iota_1 : \text{MAAdj} &\rightarrow \text{MGraph} & \iota_2 : \text{boundP} &\rightarrow \text{MGraph} \\ (B, [G]) &\mapsto ([()], !, !, B, [G]) & (k, P) &\mapsto ([\mathbb{0}_k], P_2, \mathbb{0}, P_1, [\mathbb{0}]) \end{aligned}$$

where  $P = \begin{pmatrix} P_1 \\ P_2 \end{pmatrix}$ , with  $P_1 \in \text{Mat}_{\mathbb{N}}(m, m+k)$  and  $P_2 \in \text{Mat}_{\mathbb{N}}(k, m+k)$ . These are homomorphisms of props. They respect composition.

$$\begin{aligned} \iota_1(B, [G]) \circlearrowleft \iota_1(C, [H]) & & \iota_2(k, P) \circlearrowleft \iota_2(j, Q) \\ := ([()], !, !, B, [G]) \circlearrowleft ([()], !, !, C, [H]) & & := ([\mathbb{0}_k], P_2, \mathbb{0}, P_1, [\mathbb{0}]) \circlearrowleft ([\mathbb{0}_j], Q_2, \mathbb{0}, Q_1, [\mathbb{0}]) \\ = ([()], !, !, CB, [H + CGC^T]) & & = ([\mathbb{0}_{k+j}], \begin{pmatrix} P_2 \\ Q_2 P_1 \end{pmatrix}, \mathbb{0}, Q_1 P_1, [\mathbb{0}]) \\ =: \iota_1(CB, [H + CGC^T]) & & \cong ([\mathbb{0}_{k+j}], \begin{pmatrix} Q_2 P_1 \\ P_2 \end{pmatrix}, \mathbb{0}, Q_1 P_1, [\mathbb{0}]) \\ = \iota_1((B, [G]) \circlearrowleft (C, [H])) & & =: \iota_2(k+j, \begin{pmatrix} Q_1 P_1 \\ P_2 \end{pmatrix}) \\ & & = \iota_2(k+j, \begin{pmatrix} Q \\ \mathbb{1}_k \end{pmatrix} P) \\ & & = \iota_2((k, P) \circlearrowleft (j, Q)) \end{aligned}$$

They respect identities.

$$\begin{aligned} \iota_1(\mathbb{1}_n, [\mathbb{0}]) & & \iota_2(\mathbb{0}, \mathbb{1}_n) \\ := ([()], !, !, \mathbb{1}_n, [\mathbb{0}]) & & := ([()], !, !, \mathbb{1}_n, [\mathbb{0}]) \\ = \mathbb{1}_n & & = \mathbb{1}_n \end{aligned}$$

They respect the monoidal product.

$$\begin{aligned} \iota_1(B, [G]) \otimes \iota_1(B', [G']) & & \iota_2(k, P) \otimes \iota_2(k', P') \\ := ([()], !, !, B, [G]) \otimes ([()], !, !, B', [G']) & & := ([\mathbb{0}_k], P_2, \mathbb{0}, P_1, [\mathbb{0}]) \otimes ([\mathbb{0}_{k'}], P'_2, \mathbb{0}, P'_1, [\mathbb{0}]) \\ = ([()], !, !, B \oplus B', [G \oplus G']) & & = ([\mathbb{0}_{k+k'}], P_2 \oplus P'_2, \mathbb{0}, P_1 \oplus P'_1, [\mathbb{0}]) \\ =: \iota_1(B \oplus B', [G \oplus G']) & & =: \iota_2(k+k', \begin{pmatrix} P_1 \oplus P'_1 \\ P_2 \oplus P'_2 \end{pmatrix}) \end{aligned}$$

$$\begin{aligned}
&= \iota_1((B, [G]) \otimes (B', [G'])) &&= \iota_2(k + k', (\mathbb{1}_m \oplus \sigma_{k,m'} \oplus \mathbb{1}_{k'})) \binom{P}{P'} \\
& &&= \iota_2((k, P) \otimes (k', P'))
\end{aligned}$$

□

The inclusions of MAdj and boundP into MGraph characterise all morphisms.

**Lemma 4.28.** *Morphisms  $g : n \rightarrow m$  in MGraph split as  $g = \iota_1(a) \circ \iota_2(v)$ , for some  $a : n \rightarrow l$  in MAdj and  $v : l \rightarrow m$  in boundP, uniquely up to permutations.*

*Proof.* A morphism  $g = ([G], L, R, P, [S])$  with  $k$  vertices in  $\text{MGraph}(n, m)$  splits as a composition.

$$\begin{aligned}
g &= ([G], L, R, P, [S]) \\
&= ([()], \iota_n, \iota_{m+k}, \binom{P}{L}, \left[ \begin{pmatrix} S & 0 \\ R & G \end{pmatrix} \right]) \circ ([0_k], (\mathbb{0} | \mathbb{1}_k), \mathbb{0}, (\mathbb{1}_m | \mathbb{0}), [0_m]) \\
&= \iota_1\left(\binom{P}{L}, \left[ \begin{pmatrix} S & 0 \\ R & G \end{pmatrix} \right]\right) \circ \iota_2(k, \mathbb{1}_{m+k}) \\
&= \iota_1(a) \circ \iota_2(v)
\end{aligned}$$

Suppose that the same morphism  $g$  splits as  $g = \iota_1(B, [T]) \circ \iota_2(k', P_\tau) = \iota_1(a') \circ \iota_2(v')$  as well. We show that there is a permutation  $\tau$  such that  $a = a' \circ \tau$  and  $v' = \tau \circ v$ .

Then,  $P_\tau \in \text{Mat}_{\mathbb{N}}(m' + k', m' + k')$  is the matrix corresponding to a permutation  $\tau$  and  $m' = m$  because  $(k', P_\tau) : m' + k' \rightarrow m'$ ,  $g : n \rightarrow m$  and their codomains must coincide. This permutation matrix splits along its rows as  $P_\tau = \begin{pmatrix} P_1 \\ P_2 \end{pmatrix}$ , with  $P_1 \in \text{Mat}_{\mathbb{N}}(m, m + k')$  and  $P_2 \in \text{Mat}_{\mathbb{N}}(k', m + k')$  and the second factor of  $g$  splits with the permutation  $\tau$ :  $(k', P_\tau) = (\mathbb{0}, P_\tau) \circ (k', \mathbb{1}_{m+k'}) = \tau \circ (k', \mathbb{1}_{m+k'})$ .

$$\begin{aligned}
g &= \iota_1(B, [T]) \circ \iota_2(k', P_\tau) \\
&= \iota_1(B, [T]) \circ \iota_2(\tau \circ (k', \mathbb{1}_{m+k'})) \\
&= \iota_1(B, [T]) \circ \tau \circ \iota_2(k', \mathbb{1}_{m+k'}) \\
&= \iota_1((B, [T]) \circ \tau) \circ \iota_2(k', \mathbb{1}_{m+k'}) \\
&= \iota_1(P_\tau B, [P_\tau T P_\tau^\top]) \circ \iota_2(k', \mathbb{1}_{m+k'}) \\
&= ([()], \iota_n, \iota_{m+k'}, P_\tau B, [P_\tau T P_\tau^\top]) \circ ([0_{k'}], (\mathbb{0} | \mathbb{1}_{k'}), \mathbb{0}, (\mathbb{1}_m | \mathbb{0}), [0_m]) \\
&= ([P_2 T P_2^\top], P_2 B, P_2(T + T^\top)P_1^\top, P_1 B, [P_1 T P_1^\top])
\end{aligned}$$

Then, we can rewrite the components of  $g$  in terms of  $P_\tau$ ,  $B$  and  $T$ .

$$\begin{aligned}
[G] &= [P_2 T P_2^\top] \\
L &= P_2 B \\
R &= P_2(T + T^\top)P_1^\top \\
P &= P_1 B \\
[S] &= [P_1 T P_1^\top]
\end{aligned}$$

As a consequence,  $k = k'$  and we can relate the two factorisations.

$$\begin{aligned}
&\binom{P}{L} && \left[ \begin{pmatrix} S & 0 \\ R & G \end{pmatrix} \right] \\
&= \begin{pmatrix} P_1 B \\ P_2 B \end{pmatrix} &&= \left[ \begin{pmatrix} P_1 T P_1^\top & 0 \\ P_2(T + T^\top)P_1^\top & P_2 T P_2^\top \end{pmatrix} \right]
\end{aligned}$$

$$\begin{aligned}
&= P_\tau B &= \left[ \begin{pmatrix} P_1 T P_1^\top & P_1 T P_2^\top \\ P_2 T P_1^\top & P_2 T P_2^\top \end{pmatrix} \right] \\
& &= [P_\tau T P_\tau^\top]
\end{aligned}$$

Then,  $((\begin{smallmatrix} P \\ L \end{smallmatrix}), [\begin{smallmatrix} S & 0 \\ R & G \end{smallmatrix}])) = (P_\tau B, [P_\tau T P_\tau^\top]) = (B, [T]) \circlearrowright \tau$ .  $\square$

By Theorem 2.16, this result means that MGraph is a composite prop. The next result ensures that MGraph is, in particular, the coproduct of MAdj and boundP.

**Lemma 4.29.** For any two prop morphisms  $\mathbf{a} : \text{MAdj} \rightarrow \mathbf{P}$  and  $\mathbf{v} : \text{boundP} \rightarrow \mathbf{P}$ ,

$$\mathbf{v}(k, P) \circlearrowright \mathbf{a}(B, [S]) = \mathbf{a}((B \oplus \mathbb{1}_k)P, [S \oplus \mathbb{0}_k]) \circlearrowright \mathbf{v}(k, \mathbb{1}_{m+k}).$$

*Proof.* We compute the composition using that  $\mathbf{a}$  and  $\mathbf{v}$  are prop morphisms. We use the red functor boxes for  $\mathbf{v}$  and the blue ones for  $\mathbf{a}$ . We indicate with the costate  $k$  the morphism  $(k, \mathbb{1}_k)$  in boundP, with  $b$  the morphism  $(B, [S])$  in MAdj, and with  $\sigma_P$  the permutation in MAdj, boundP or P corresponding to the permutation matrix  $P$ .

$$\begin{aligned}
&\mathbf{v}(k, P) \circlearrowright \mathbf{a}(B, [S]) \\
&= \mathbf{v}(\sigma_P \circlearrowright (\mathbb{1}_n \otimes (k, \mathbb{1}_k))) \circlearrowright \mathbf{a}(B, [S]) \\
&= \text{Diagram 1} \\
&= \text{Diagram 2} \\
&= \text{Diagram 3} \\
&= \text{Diagram 4} \\
&= \text{Diagram 5} \\
&= \text{Diagram 6} \\
&= \mathbf{a}(\sigma_P \circlearrowright ((B, [S]) \otimes \mathbb{1}_k)) \circlearrowright \mathbf{v}(\mathbb{1}_m \otimes (k, \mathbb{1}_k)) \\
&= \mathbf{a}((B \oplus \mathbb{1}_k)P, [S \oplus \mathbb{0}_k]) \circlearrowright \mathbf{v}(k, \mathbb{1}_{m+k})
\end{aligned}$$

$\square$

With these results, we can show the universal property of MGraph.

**Proposition 4.30.** The prop of graphs with dangling edges is the coproduct of the prop of adjacency matrices and that of bounded permutations:  $\text{MGraph} \cong \text{MAdj} + \text{boundP}$ .

*Proof.* By Lemma 4.28, we can apply the result on composition of props [Lac04, Theorem 4.6], recalled in Theorem 2.16, to the prop  $\mathbf{MGraph}$  to obtain that it is the composition of  $\mathbf{MAAdj}$  and  $\mathbf{boundP}$  via a distributive law  $\lambda : (\iota_2(v) \mid \iota_1(a)) \mapsto (\iota_1(\hat{a}) \mid \iota_2(\hat{v}))$ . In particular, for any two prop morphisms  $\mathbf{a} : \mathbf{MAAdj} \rightarrow \mathbf{P}$  and  $\mathbf{v} : \mathbf{boundP} \rightarrow \mathbf{P}$  such that  $\mathbf{v}(v) \circledast \mathbf{a}(a) = \mathbf{a}(\hat{a}) \circledast \mathbf{v}(\hat{v})$ , there is a unique prop morphism  $\mathbf{h} : \mathbf{MGraph} \rightarrow \mathbf{P}$  such that  $\mathbf{a} = \iota_1 \circledast \mathbf{h}$  and  $\mathbf{v} = \iota_2 \circledast \mathbf{h}$ . By Lemma 4.29, any two prop morphisms  $\mathbf{a} : \mathbf{MAAdj} \rightarrow \mathbf{P}$  and  $\mathbf{v} : \mathbf{boundP} \rightarrow \mathbf{P}$  satisfy  $\mathbf{v}(v) \circledast \mathbf{a}(a) = \mathbf{a}(\hat{a}) \circledast \mathbf{v}(\hat{v})$ , which means that any two such morphisms define a unique prop morphism  $\mathbf{h} : \mathbf{MGraph} \rightarrow \mathbf{P}$  such that  $\mathbf{a} = \iota_1 \circledast \mathbf{h}$  and  $\mathbf{v} = \iota_2 \circledast \mathbf{h}$ . This is equivalent to say that  $\mathbf{MGraph}$  satisfies the universal property of the coproduct of  $\mathbf{MAAdj}$  and  $\mathbf{boundP}$ .  $\square$

### A syntax for graphs with dangling edges

We give a syntactic presentation of graphs with dangling edges by giving syntactic presentations of its components  $\mathbf{MAAdj}$  and  $\mathbf{boundP}$ . The string diagrammatic syntax for adjacency matrices relies on the characterisation of matrices as a bialgebra but needs the addition of a “cup” generator  $\subset : 0 \rightarrow 2$  (Figure 4.4) that captures the equivalence relation of adjacency matrices (Lemma 4.36). Theorem 4.39 shows that  $\mathbf{Adj}$  is a syntactic presentation of  $\mathbf{MAAdj}$ .

**Definition 4.31.** The prop  $\mathbf{Adj}$  is presented by the generators and equations in Figures 4.3 and 4.4.

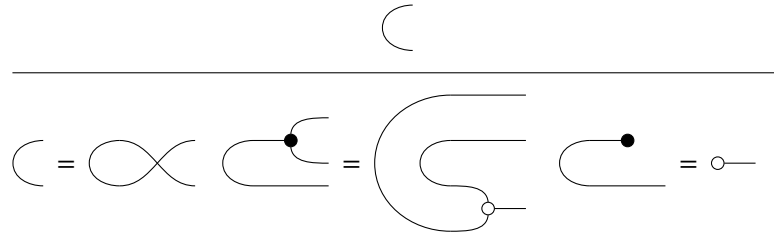


Figure 4.4: Additional generator and equations for the prop of adjacency matrices (Figure 4.3 contains the rest of generators and equations).

As recalled in Section 2.1, presenting a prop with generators and equations corresponds to taking a coequaliser in the category  $\mathbf{Prop}$  of props and their morphisms. The generators and equations in Figure 4.4 indicate that  $\mathbf{Adj}$  is the coequaliser of two prop morphisms  $\mathbf{s}, \mathbf{t} : \mathbf{A} \rightarrow \mathbf{Bialg} + \mathbf{Cup}$ . The prop  $\mathbf{A}$  is freely generated by two morphisms  $a : 0 \rightarrow 3$  and  $b : 0 \rightarrow 1$ , while the prop  $\mathbf{Cup}$  is presented by a cup morphism  $\subset : 0 \rightarrow 2$  and quotiented by the first equation in Figure 4.4. The prop morphisms are defined inductively by their images on the generators of  $\mathbf{A}$ .

$$\begin{array}{ll}
 \mathbf{s}(a) := \text{diagram of two strands with a dot on the top strand} & \mathbf{t}(a) := \text{diagram of two strands with a loop and a dot on the top strand} \\
 \mathbf{s}(b) := \text{diagram of one strand with a dot} & \mathbf{t}(b) := \text{diagram of one strand with a circle}
 \end{array} \tag{4.1}$$

The isomorphism between the props  $\mathbf{Adj}$  and  $\mathbf{MAAdj}$  is proven in [CS15, Theorem 4.2] by defining a prop morphism  $\mathbf{Adj} \rightarrow \mathbf{MAAdj}$  inductively and showing that it is an isomorphism. We rely on the same arguments but give a slightly different proof. We show that  $\mathbf{MAAdj}$  also satisfies the universal property of the coequaliser



of  $s$  and  $t$ . The isomorphism  $\phi : \text{MAdj} \rightarrow \text{Adj}$  defined in Theorem 4.39 captures the normal form of morphisms in  $\text{Adj}$ .

$$\phi : (B, [G]) \mapsto \text{---} \begin{array}{c} \boxed{B} \\ \text{---} \circ \text{---} \\ \text{---} \circ \text{---} \\ \boxed{G} \\ \text{---} \circ \text{---} \\ \text{---} \circ \text{---} \end{array} \text{---} \quad (4.2)$$

This notation implicitly relies on the isomorphism  $\text{Bialg} \cong \text{Mat}_{\mathbb{N}}$  of Proposition 4.18: a box  $\boxed{A}$  indicates the image of the matrix  $A$  under the isomorphism  $\text{Mat}^{-1} : \text{Mat}_{\mathbb{N}} \rightarrow \text{Bialg}$ .

The proof that  $\text{MAdj}$  is the coequaliser of  $s$  and  $t$  first constructs a candidate coequaliser map  $\mathbf{q} : \text{Bialg} + \text{Cup} \rightarrow \text{MAdj}$  and then shows the universal property for it. The prop morphism  $\mathbf{q}$  is defined as a coproduct map of prop morphisms  $\mathbf{b} : \text{Bialg} \rightarrow \text{MAdj}$  and  $\mathbf{c} : \text{Cup} \rightarrow \text{MAdj}$ . The morphism  $\mathbf{b}$  is the composition of the isomorphism  $\text{Mat} : \text{Bialg} \rightarrow \text{Mat}_{\mathbb{N}}$  and the prop morphism  $\mathbf{j} : \text{Mat}_{\mathbb{N}} \rightarrow \text{MAdj}$  described in Lemma 4.32.

**Lemma 4.32.** *There is a morphism of props  $\mathbf{j} : \text{Mat}_{\mathbb{N}} \rightarrow \text{MAdj}$  from the prop of matrices to that of adjacency matrices defined by  $\mathbf{j}(A) := (A, [0])$ .*

*Proof.* We check that  $\mathbf{j}$  preserves compositions, identities and monoidal products.

$$\begin{array}{lll} \mathbf{j}(A) \mathbin{\text{;}} \mathbf{j}(B) & \mathbf{j}(1_n) & \mathbf{j}(A) \otimes \mathbf{j}(A') \\ := (A, [0]) \mathbin{\text{;}} (B, [0]) & := (1_n, [0_n]) & := (A, [0]) \otimes (A', [0]) \\ := (BA, [B0B^T + 0]) & = 1_n & := (A \oplus A', [0 \oplus 0]) \\ = (BA, [0]) & & = (A \oplus A', [0]) \\ =: \mathbf{j}(A \mathbin{\text{;}} B) & & =: \mathbf{j}(A \oplus A') \end{array}$$

□

There is a morphism in  $\text{MAdj}$  that behaves like the cup  $\subset$ .

**Lemma 4.33.** *There is a morphism of props  $\mathbf{c} : \text{Cup} \rightarrow \text{MAdj}$  defined by*

$$\mathbf{c}(\subset) := (i_2, \left[ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right]).$$

*Proof.* We define the mapping  $\mathbf{c}$  on the generator as  $\mathbf{c}(\subset) := (i_2, \left[ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right])$ , which, using the isomorphism  $\text{Mat} : \text{Bialg} \rightarrow \text{Mat}_{\mathbb{N}}$ , becomes

$$\mathbf{c}(\subset) = \left( \text{Mat} \left( \begin{array}{c} \circ \text{---} \\ \circ \text{---} \end{array} \right), \left[ \text{Mat} \left( \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \circ \text{---} \end{array} \right) \right] \right).$$

The image on the rest of the morphisms of  $\text{Cup}$  is defined inductively, so we need to check that the equation of commutativity of the cup holds. We use the equivalence relation of adjacency matrices for  $\begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \sim \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ .

$$\begin{aligned} & \mathbf{c}(\subset \mathbin{\text{;}} \infty) \\ & := \mathbf{c}(\subset) \mathbin{\text{;}} \mathbf{c}(\infty) \\ & = \left( \text{Mat} \left( \begin{array}{c} \circ \text{---} \\ \circ \text{---} \end{array} \right), \left[ \text{Mat} \left( \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \circ \text{---} \end{array} \right) \right] \right) \mathbin{\text{;}} \left( \text{Mat} \left( \begin{array}{c} \infty \\ \infty \end{array} \right), \left[ \text{Mat} \left( \begin{array}{c} \bullet \circ \text{---} \\ \bullet \circ \text{---} \end{array} \right) \right] \right) \\ & := \left( \text{Mat} \left( \begin{array}{c} \infty \\ \infty \end{array} \right), \left[ \text{Mat} \left( \begin{array}{c} \infty \\ \infty \end{array} \right) + \text{Mat} \left( \begin{array}{c} \bullet \circ \text{---} \\ \bullet \circ \text{---} \end{array} \right) \right] \right) \\ & = \left( \text{Mat} \left( \begin{array}{c} \circ \text{---} \\ \circ \text{---} \end{array} \right), \left[ \text{Mat} \left( \begin{array}{c} \text{---} \bullet \text{---} \\ \text{---} \circ \text{---} \end{array} \right) \right] \right) \\ & = (i_2, \left[ \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \right]) \end{aligned}$$

$$\begin{aligned}
&= (i_2, \left[ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \right]) \\
&=: \mathbf{c}(\cup)
\end{aligned}$$

□

The prop morphism  $\mathbf{q} : \mathbf{Bialg} + \mathbf{Cup} \rightarrow \mathbf{MAAdj}$  is the coproduct map of the prop morphisms  $\mathbf{b}$  and  $\mathbf{c}$  defined in Lemmas 4.32 and 4.33. The morphism  $\mathbf{q}$  also coequalises  $\mathbf{s}$  and  $\mathbf{t}$ .

**Proposition 4.34.** *The coproduct map  $\mathbf{q} := [\mathbf{b}, \mathbf{c}]$  is a coequalising prop morphism  $\mathbf{q} : \mathbf{Bialg} + \mathbf{Cup} \rightarrow \mathbf{MAAdj}$  of the pair  $\mathbf{s}, \mathbf{t} : \mathbf{A} \rightarrow \mathbf{Bialg} + \mathbf{Cup}$ .*

*Proof.* The prop morphism  $\mathbf{b} : \mathbf{Bialg} \rightarrow \mathbf{MAAdj}$  is defined as the composition of the isomorphism  $\mathbf{Mat} : \mathbf{Bialg} \rightarrow \mathbf{Mat}_{\mathbb{N}}$ , recalled in Proposition 4.18, with the morphism  $\mathbf{j} : \mathbf{Mat}_{\mathbb{N}} \rightarrow \mathbf{MAAdj}$ , defined in Lemma 4.32. The prop morphism  $\mathbf{c} : \mathbf{Cup} \rightarrow \mathbf{MAAdj}$  is defined in Lemma 4.33. We show that their coproduct map  $\mathbf{q}$  is a coequalising morphism of  $\mathbf{s}$  and  $\mathbf{t}$  by computing the images of  $\mathbf{s} \circ \mathbf{q}$  and  $\mathbf{t} \circ \mathbf{q}$  on both the morphisms  $a : 0 \rightarrow 2$  and  $b : 0 \rightarrow 1$  of the prop  $\mathbf{A}$ . For both computations, we use the definition of  $\mathbf{q}$  as a coproduct map.

$$\begin{aligned}
\mathbf{q}(\mathbf{s}(a)) &:= \mathbf{q} \left( \text{Diagram: a cup with a dot on the top wire} \right) \\
&= \mathbf{q} \left( \text{Diagram: a cup} \right) \circledast \mathbf{q} \left( \text{Diagram: a cup with a dot on the top wire} \right) \\
&= \mathbf{c} \left( \text{Diagram: a cup} \right) \circledast \mathbf{b} \left( \text{Diagram: a cup with a dot on the top wire} \right) \\
&:= \left( \mathbf{Mat} \left( \text{Diagram: two wires} \right), \left[ \mathbf{Mat} \left( \text{Diagram: two wires with a dot} \right) \right] \right) \circledast \left( \mathbf{Mat} \left( \text{Diagram: a cup} \right), \left[ \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \\
&= \left( \mathbf{Mat} \left( \text{Diagram: two wires with a dot} \right), \left[ \mathbf{Mat} \left( \text{Diagram: two wires with a dot} \right) + \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \\
&= \left( \mathbf{Mat} \left( \text{Diagram: three wires} \right), \left[ \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \\
&= \left( \mathbf{Mat} \left( \text{Diagram: three wires} \right), \left[ \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) + \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \\
&= \left( \mathbf{Mat} \left( \text{Diagram: three wires} \right), \left[ \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \circledast \left( \mathbf{Mat} \left( \text{Diagram: a cup} \right), \left[ \mathbf{Mat} \left( \text{Diagram: three wires with a dot} \right) \right] \right) \\
&= \mathbf{c} \left( \text{Diagram: a cup} \right) \circledast \mathbf{b} \left( \text{Diagram: a cup with a dot on the top wire} \right) \\
&= \mathbf{q} \left( \text{Diagram: a cup} \right) \circledast \mathbf{q} \left( \text{Diagram: a cup with a dot on the top wire} \right)
\end{aligned}$$

$$\begin{aligned}
 &= \mathbf{q} \left( \text{Diagram: a cup with a line entering from the right, looping back to the left, and exiting from the right} \right) \\
 &=: \mathbf{q}(\mathbf{t}(a)) \\
 \mathbf{q}(s(b)) &:= \mathbf{q} \left( \text{Diagram: a cup with a line entering from the right, looping back to the left, and exiting from the left} \right) \\
 &= \mathbf{q} \left( \text{Diagram: a cup} \right) ; \mathbf{q} \left( \text{Diagram: a line with a dot on the left} \right) \\
 &= \mathbf{c} \left( \text{Diagram: a cup} \right) ; \mathbf{b} \left( \text{Diagram: a line with a dot on the left} \right) \\
 &= \left( \text{Mat} \left( \text{Diagram: a cup} \right), \left[ \text{Mat} \left( \text{Diagram: a line with a dot on the left} \right) \right] \right) ; \left( \text{Mat} \left( \text{Diagram: a line with a dot on the left} \right), \left[ \text{Mat} \left( \text{Diagram: a line with a dot on the right} \right) \right] \right) \\
 &= \left( \text{Mat} \left( \text{Diagram: a cup with a dot on the left} \right), \left[ \text{Mat} \left( \text{Diagram: a line with a dot on the left} \right) + \text{Mat} \left( \text{Diagram: a line with a dot on the right} \right) \right] \right) \\
 &= \left( \text{Mat} \left( \text{Diagram: a line with a dot on the left} \right), \left[ \text{Mat} \left( \text{Diagram: a line with a dot on the right} \right) \right] \right) \\
 &= \mathbf{b} \left( \text{Diagram: a line with a dot on the right} \right) \\
 &= \mathbf{q} \left( \text{Diagram: a line with a dot on the right} \right) \\
 &=: \mathbf{q}(\mathbf{t}(b))
 \end{aligned}$$

□

We show that the prop morphism  $\mathbf{q}$  satisfies the universal property of coequalisers. For every coequalising prop morphism  $\mathbf{p} : \mathbf{Bialg} + \mathbf{Cup} \rightarrow \mathbf{P}$  of the pair  $s, t : \mathbf{A} \rightarrow \mathbf{Bialg} + \mathbf{Cup}$ , Proposition 4.38 defines a candidate extension  $\bar{\mathbf{p}} : \mathbf{MA}dj \rightarrow \mathbf{P}$  of  $\mathbf{p}$  to  $\mathbf{MA}dj$ . Theorem 4.39 concludes by showing that  $\bar{\mathbf{p}}$  is the unique extension of  $\mathbf{p}$  along  $\mathbf{q}$ . For constructing the candidate extension  $\bar{\mathbf{p}}$  we need to investigate some properties of the coequalising morphism  $\mathbf{p}$  that are consequences of the cup axioms in Figure 4.4. Those equations imply that the cup quotients by transposition.

This equation holds in  $\mathbf{Adj}$  and also in the image of any coequalising morphism of  $s$  and  $t$ .

**Lemma 4.35.** For any coequalising morphism of props  $\mathbf{p} : \mathbf{Bialg} + \mathbf{Cup} \rightarrow \mathbf{P}$  of the pair  $s, t : \mathbf{A} \rightarrow \mathbf{Bialg} + \mathbf{Cup}$  in Equation (4.1),

$$\mathbf{p} \left( \text{Diagram: a cup with a box labeled A on the left} \right) = \mathbf{p} \left( \text{Diagram: a cup with a box labeled A^T on the right} \right). \tag{4.3}$$

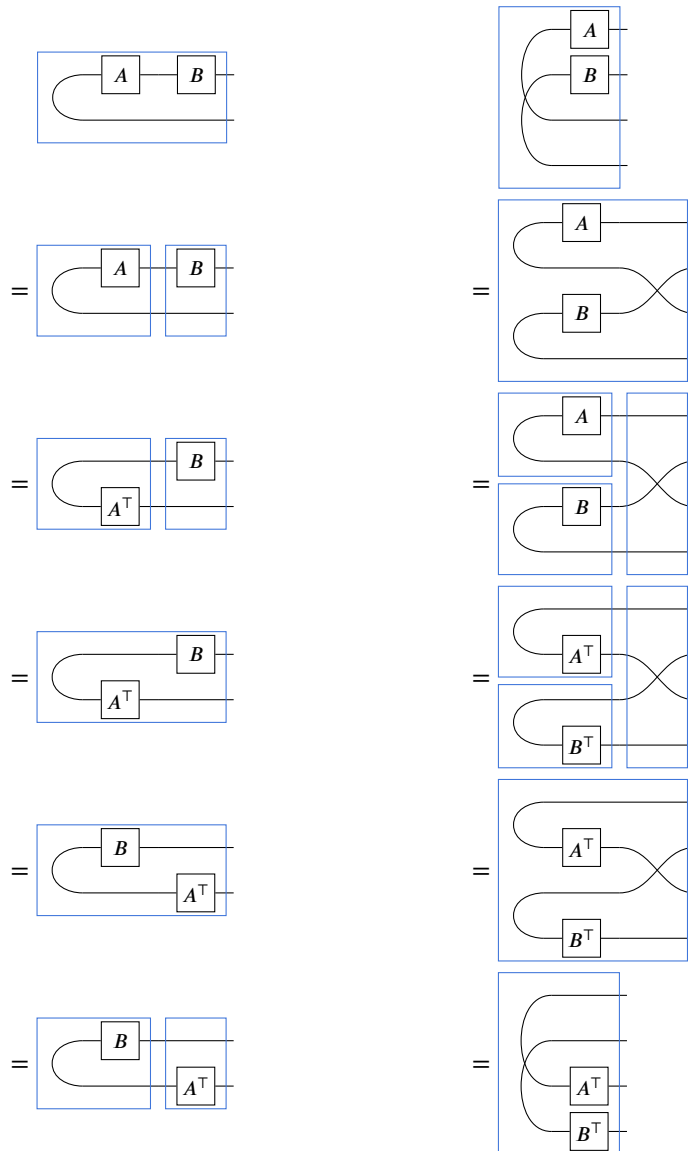
*Proof.* By Proposition 4.18, every morphism  $A : n \rightarrow m$  in  $\mathbf{Mat}_{\mathbb{N}}$  can be written as compositions and monoidal products of finitely many of its generators. These generators are the images under the isomorphism  $\mathbf{Mat} : \mathbf{Bialg} \rightarrow \mathbf{Mat}$  of the generators in Figure 4.3. By these considerations, the proof can proceed by structural induction on the morphisms. For the base cases, Equation (4.3) holds for the bialgebra generators

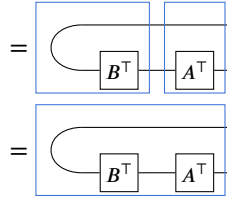
because  $\mathbf{p}$  is a coequalising morphism for  $s$  and  $t$ ,  $s \circ \mathbf{p} = t \circ \mathbf{p}$ .

$$\mathbf{p} \left( \begin{array}{c} \bullet \\ \text{---} \end{array} \right) = \mathbf{p}(s(a)) = \mathbf{p}(t(a)) = \mathbf{p} \left( \begin{array}{c} \text{---} \\ \circ \end{array} \right)$$

$$\mathbf{p} \left( \begin{array}{c} \bullet \\ \text{---} \end{array} \right) = \mathbf{p}(s(b)) = \mathbf{p}(t(b)) = \mathbf{p}(\circ \text{---})$$

The two remaining equations follow by commutativity of the cup. For the inductive steps, suppose that Equation (4.3) holds for  $A : n \rightarrow m$ ,  $B : m \rightarrow l$  and  $A' : n' \rightarrow m'$ . We show that it holds for  $A \circ B$  and for  $A \oplus A'$ . We indicate  $\mathbf{p}$  with a blue functor box.





□

A consequence of this result is that equality in the prop of adjacency matrices captures the equivalence relation of adjacency matrices. Recalling Definition 4.21, two adjacency matrices are equivalent,  $[G] = [H]$ , if and only if they are equal up to transposition,  $G + G^T = H + H^T$ . In string diagrams, this is

$$[G] = [H] \quad \text{iff} \quad \text{loop}(G) = \text{loop}(H),$$

and holds in  $\text{Adj}$  and the image of any coequalising prop morphism of  $\mathbf{s}$  and  $\mathbf{t}$ .

**Lemma 4.36.** For two adjacency matrices  $[A]$  and  $[B]$ , and any coequalising morphism of props  $\mathbf{p} : \text{Bialg} + \text{Cup} \rightarrow \mathbf{P}$  of the pair  $\mathbf{s}, \mathbf{t} : A \rightarrow \text{Bialg} + \text{Cup}$  in Equation (4.1),

$$\text{if} \quad [A] = [B] \quad \text{then} \quad \mathbf{p} \left( \text{loop}(A) \right) = \mathbf{p} \left( \text{loop}(B) \right).$$

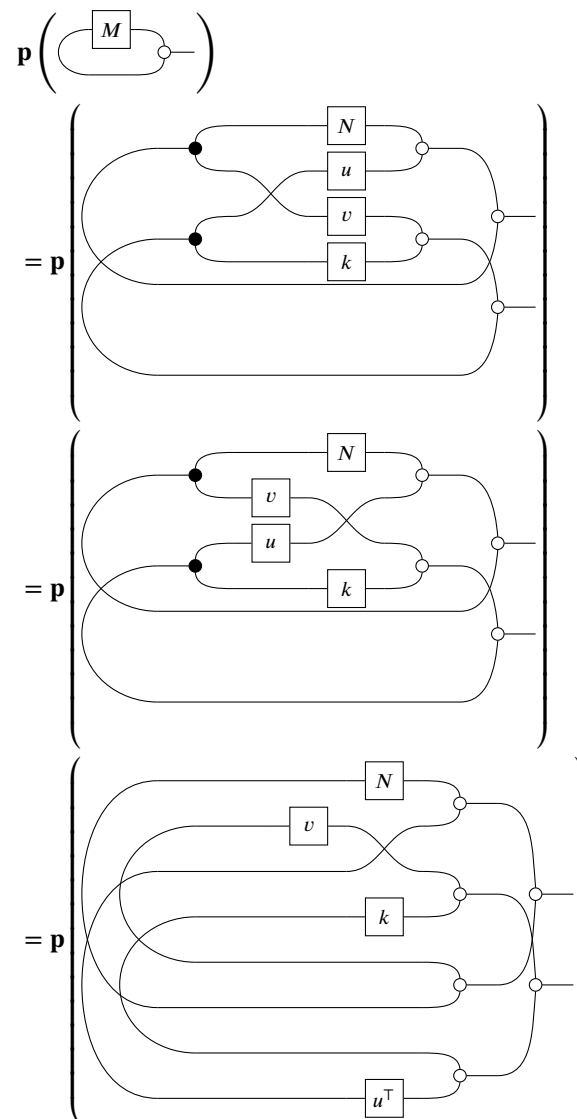
*Proof.* Proceed by induction on the size  $n$  of the matrices. For  $n = 0$ , there is only one morphism  $0 \rightarrow 0$  in  $\text{Mat}_{\mathbb{N}}$  so the statement is trivially true. For the induction step, suppose that the statement is true for any two  $n$  by  $n$  matrices  $A'$  and  $B'$  and consider two  $n + 1$  by  $n + 1$  matrices  $A = \begin{pmatrix} A' & a \\ a' & i \end{pmatrix}$  and  $B = \begin{pmatrix} B' & b \\ b' & j \end{pmatrix}$ . Notice that the two matrices are equivalent,  $[A] = [B]$ , if and only if  $[A'] = [B']$ ,  $a' + a^T = b' + b^T$  and  $i = j$ , because  $2 \cdot i = 2 \cdot j$  implies  $i = j$ . By induction hypothesis,  $[A'] = [B']$ ,  $a' + a^T = b' + b^T$  and  $i = j$  imply the corresponding equalities in the image of  $\mathbf{p}$ .

$$\begin{aligned} \mathbf{p} \left( \text{loop}(A) \right) &= \mathbf{p} \left( \text{loop}(B) \right) \\ \mathbf{p} \left( \begin{array}{c} \bullet \\ \text{loop}(A') \\ \bullet \end{array} \right) &= \mathbf{p} \left( \begin{array}{c} \bullet \\ \text{loop}(B') \\ \bullet \end{array} \right) \\ \mathbf{p} \left( \text{loop}(i) \right) &= \mathbf{p} \left( \text{loop}(j) \right) \end{aligned}$$

By functoriality of  $\mathbf{p}$ , we obtain

$$\mathbf{p} \left( \begin{array}{c} \text{loop}(A') \\ \text{loop}(a') \\ \text{loop}(a^T) \\ \text{loop}(i) \end{array} \right) = \mathbf{p} \left( \begin{array}{c} \text{loop}(B') \\ \text{loop}(b') \\ \text{loop}(b^T) \\ \text{loop}(j) \end{array} \right).$$

By the bialgebra axioms (Figure 4.3) and Lemma 4.35, we can do the rewrites below for any  $n + 1$  by  $n + 1$  square matrix  $M = \begin{pmatrix} N & u \\ v & k \end{pmatrix}$ .





$$= \text{Diagram: A box labeled } H \text{ with a loop and an output line.}$$

Thanks to Lemmas 4.35 and 4.36, the mapping  $\phi : \text{MAdj} \rightarrow \text{Adj}$  given in Equation (4.2) is a prop morphism. More generally, for every coequalising prop morphism  $\mathbf{p} : \text{Bialg} + \text{Cup} \rightarrow \mathbf{P}$  of the pair  $s, \mathbf{t} : A \rightarrow \text{Bialg} + \text{Cup}$ , these results allow us to define a candidate extension  $\bar{\mathbf{p}} : \text{MAdj} \rightarrow \mathbf{P}$ .

**Proposition 4.38.** Any coequalising prop morphism  $\mathbf{p} : \text{Bialg} + \text{Cup} \rightarrow \mathbf{P}$  of the pair  $s, \mathbf{t} : A \rightarrow \text{Bialg} + \text{Cup}$  in Equation (4.1) induces a prop morphism  $\bar{\mathbf{p}} : \text{MAdj} \rightarrow \mathbf{P}$  given by

$$\bar{\mathbf{p}}(B, [G]) := \mathbf{p} \left( \text{Diagram: A box labeled } G \text{ with a box labeled } B \text{ on top, a loop, and an output line.} \right).$$

*Proof.* By Lemma 4.36 and functoriality of  $\mathbf{p}$ , the assignment  $\bar{\mathbf{p}}$  is well-defined on equivalence classes of adjacency matrices: if  $(B, [G]) = (B, [H])$ , then  $\bar{\mathbf{p}}(B, [G]) = \bar{\mathbf{p}}(B, [H])$  because

$$\mathbf{p} \left( \text{Diagram: } G \text{ with } B \text{ on top} \right) = \mathbf{p} \left( \text{Diagram: } H \text{ with } B \text{ on top} \right).$$

Applying Lemma 4.35, we check that  $\bar{\mathbf{p}}$  preserves compositions.

$$\begin{aligned} & \bar{\mathbf{p}}((B, [G]) \circ (C, [H])) \\ & := \bar{\mathbf{p}}(CB, [CGC^\top + H]) \\ & := \mathbf{p} \left( \text{Diagram: } C^\top, G, C, B, C \text{ on top; } H \text{ on bottom; } C \text{ on bottom.} \right) \\ & = \mathbf{p} \left( \text{Diagram: } G, C, B, C \text{ on top; } H \text{ on bottom; } C \text{ on bottom.} \right) \\ & = \mathbf{p} \left( \text{Diagram: } G, C, B, C \text{ on top; } H \text{ on bottom.} \right) \\ & = \mathbf{p} \left( \text{Diagram: } G, B \text{ on top; } H, C \text{ on bottom.} \right) \\ & = \mathbf{p} \left( \text{Diagram: } G, B \text{ on top} \right) \circ \mathbf{p} \left( \text{Diagram: } H, C \text{ on top} \right) \end{aligned}$$



$$=: \bar{\mathbf{p}}(B, [G]) \circledast \bar{\mathbf{p}}(C, [H])$$

The hypothesis that  $\mathbf{p}$  is a coequalising morphism implies that  $\mathbf{p}(s(b)) = \mathbf{p}(t(b))$  and that  $\bar{\mathbf{p}}$  preserves identities.

$$\begin{aligned} & \bar{\mathbf{p}}(1_n, [\mathbb{0}_n]) \\ & := \mathbf{p} \left( \text{Diagram with boxes } 0 \text{ and } 1_n \right) \\ & = \mathbf{p} \left( \text{Diagram with a black dot} \right) \\ & = \mathbf{p} \left( \text{Diagram with two inputs} \right) \\ & = \mathbf{p}(\text{---}_n) \\ & = 1_n \end{aligned}$$

Finally, we check that  $\bar{\mathbf{p}}$  preserves monoidal products.

$$\begin{aligned} & \bar{\mathbf{p}}((B, [G]) \otimes (B', [G'])) \\ & := \bar{\mathbf{p}}(B \oplus B', [G \oplus G']) \\ & := \mathbf{p} \left( \text{Diagram with boxes } B, B', G, G' \right) \\ & = \mathbf{p} \left( \text{Diagram with boxes } G, B, G', B' \right) \\ & = \mathbf{p} \left( \text{Diagram with boxes } G, B \right) \otimes \mathbf{p} \left( \text{Diagram with boxes } G', B' \right) \\ & =: \bar{\mathbf{p}}(B, [G]) \otimes \bar{\mathbf{p}}(B', [G']) \end{aligned}$$

□

The candidate coequaliser of Proposition 4.34 is, indeed, a coequaliser. This follows from checking that the candidate extension of a coequalising prop morphism  $\mathbf{p}$  in Proposition 4.38 is an extension of  $\mathbf{p}$  along  $\mathbf{q}$  and is unique. In particular, the prop morphism  $\phi$  defined in Equation (4.2) is the extension of the coequaliser map  $\text{Bialg} + \text{Cup} \rightarrow \text{Adj}$ . Then,  $\phi : \text{MAj} \rightarrow \text{Adj}$  is an isomorphism and gives a normal form for morphisms in  $\text{Adj}$ .

**Theorem 4.39.** *The prop  $\text{Adj}$  is isomorphic to the prop  $\text{MAdj}$  of adjacency matrices via the isomorphism  $\phi : \text{MAdj} \rightarrow \text{Adj}$  defined in Equation (4.2).*

*Proof.* Proposition 4.34 provides a candidate  $\mathbf{q} : \text{Bialg} + \text{Cup} \rightarrow \text{MAdj}$  for the coequaliser of  $\mathbf{s}$  and  $\mathbf{t}$ , and Proposition 4.38 defines a morphism  $\bar{\mathbf{p}} : \text{MAdj} \rightarrow \mathbf{P}$  for any coequalising morphism  $\mathbf{p} : \text{Bialg} + \text{Cup} \rightarrow \mathbf{P}$ . The prop morphism  $\mathbf{q}$  is the coequaliser if  $\bar{\mathbf{p}}$  is the unique prop morphism such that  $\mathbf{q} \circ \bar{\mathbf{p}} = \mathbf{p}$ . Since  $\mathbf{q}$  is a coproduct map, its composition with  $\bar{\mathbf{p}}$  is also a coproduct map

$$\mathbf{q} \circ \bar{\mathbf{p}} = [\mathbf{b} \circ \bar{\mathbf{p}}, \mathbf{c} \circ \bar{\mathbf{p}}]$$

and we can check that the desired equality holds by checking the components separately. We implicitly use that  $\mathbf{Mat}$  is an isomorphism.

$$\begin{aligned} \bar{\mathbf{p}}(\mathbf{b}(\mathbf{Mat}^{-1}(A))) & & \bar{\mathbf{p}}(\mathbf{c}(\bigcirc)) \\ = \bar{\mathbf{p}}(\mathbf{j}(A)) & & \\ := \bar{\mathbf{p}}(A, [0]) & & := \bar{\mathbf{p}}(\mathbf{Mat}(\begin{array}{c} \circ \\ \text{---} \\ \circ \end{array}), [\mathbf{Mat}(\begin{array}{c} \bullet \\ \text{---} \\ \circ \end{array})]) \\ := \mathbf{p}\left(\begin{array}{c} \text{---} \\ \boxed{A} \\ \text{---} \\ \circ \end{array}\right) & & := \mathbf{p}\left(\begin{array}{c} \bullet \\ \text{---} \\ \circ \end{array}\right) \\ = \mathbf{p}\left(\begin{array}{c} \text{---} \\ \boxed{A} \\ \text{---} \\ \circ \end{array}\right) & & = \mathbf{p}\left(\iota_2(\bigcirc)\right) \\ = \mathbf{p}(\iota_1(\mathbf{Mat}^{-1}(A))) & & \end{aligned}$$

Suppose that there is another morphism  $\mathbf{r} : \text{MAdj} \rightarrow \mathbf{P}$  such that  $\mathbf{q} \circ \mathbf{r} = \mathbf{p}$ . We check that  $\mathbf{r}$  must coincide with  $\bar{\mathbf{p}}$ .

$$\begin{aligned} \mathbf{r}(B, [G]) & \\ = \mathbf{r}\left(\left(\begin{array}{c} 1 \\ \circ \\ 0 \end{array}\right), \left[\begin{array}{ccc} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{array}\right]\right) \circ ((B \mid G \mid 1), [0]) & \\ = \mathbf{r}\left(\mathbf{c}\left(\begin{array}{c} \text{---} \\ \circ \end{array}\right) \circ \mathbf{b}\left(\begin{array}{c} \text{---} \\ \boxed{G} \quad \boxed{B} \\ \text{---} \\ \circ \end{array}\right)\right) & \\ = \mathbf{r}\left(\mathbf{q}\left(\begin{array}{c} \text{---} \\ \circ \end{array}\right) \circ \mathbf{q}\left(\begin{array}{c} \text{---} \\ \boxed{G} \quad \boxed{B} \\ \text{---} \\ \circ \end{array}\right)\right) & \\ = \mathbf{r}\left(\mathbf{q}\left(\begin{array}{c} \text{---} \\ \circ \end{array} \circ \begin{array}{c} \text{---} \\ \boxed{G} \quad \boxed{B} \\ \text{---} \\ \circ \end{array}\right)\right) & \\ = \mathbf{p}\left(\begin{array}{c} \text{---} \\ \boxed{G} \quad \boxed{B} \\ \text{---} \\ \circ \end{array}\right) & \\ =: \bar{\mathbf{p}}(B, [G]) & \end{aligned}$$

This shows that  $\text{MAdj}$  is the coequaliser of  $\mathbf{s}$  and  $\mathbf{t}$ , but so is  $\text{Adj}$  by its definition. Colimits are unique up to the unique isomorphism given by extensions, so the prop morphism  $\phi : \text{MAdj} \rightarrow \text{Adj}$  defined in Equation (4.2) is this isomorphism.  $\square$



$$\begin{aligned}
 & := \text{Diagram 1} \\
 & = \text{Diagram 2} \\
 & = \mathbf{H}(k + j, (P_\sigma \otimes \mathbb{1}_k) P_\tau) \\
 & = \mathbf{H}((k, P_\tau) \circledast (j, P_\sigma)).
 \end{aligned}$$

The functor  $\mathbf{H}$  is unique because any other prop morphism  $\mathbf{F} : \text{boundP} \rightarrow \mathbf{P}$  such that  $\mathbf{F}(1, \mathbb{1}_1) = v$  must, by functoriality of  $\mathbf{F}$ , coincide with  $\mathbf{H}$ .

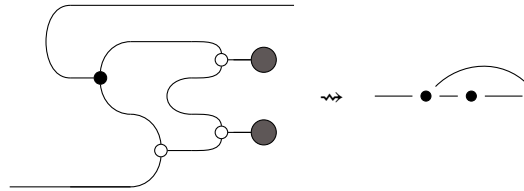
$$\begin{aligned}
 & \mathbf{F}(k, P_\tau) \\
 & = \mathbf{F}((0, P_\tau) \circledast ((0, \mathbb{1}_m) \otimes (k, \mathbb{1}_k))) \\
 & = \mathbf{F}(\tau \circledast (\mathbb{1}_m \otimes (1, \mathbb{1}_1)^k)) \\
 & = \mathbf{F}(\tau) \circledast (\mathbf{F}(\mathbb{1}_m) \otimes \mathbf{F}(1, \mathbb{1}_1)^k) \\
 & = \tau \circledast (\mathbb{1}_m \otimes v^k) \\
 & =: \mathbf{H}(k, P_\tau)
 \end{aligned}$$

□

The prop of graphs is the coproduct of that of adjacency matrices and that of vertices. Coproducts of props presented by generators and equations are presented by the disjoint union of the generators and of the equations of the components [Lac04] (see also [Zan15, Proposition 2.11]).

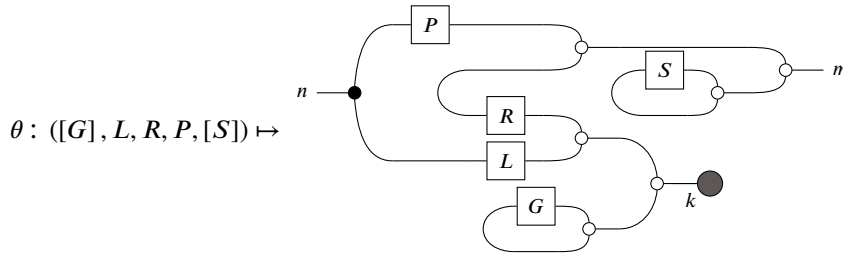
**Definition 4.42.** The prop of graphs  $\text{BGraph}$  is the coproduct of the props  $\text{Adj}$  and  $\text{Vert}$ ,  $\text{BGraph} := \text{Adj} + \text{Vert}$ . Its generators and equations are in Figures 4.3 to 4.5.

*Example 4.43.* The string diagram below on the left is a morphism  $1 \rightarrow 1$  in  $\text{BGraph}$  that represents a graph with two vertices connected by an edge. The vertices are also both connected to the right boundary, while only one of them is connected to the left boundary. This corresponds to the informal drawing of the graph below on the right.



As with the isomorphism between the props of adjacency matrices, the isomorphism  $\text{MGraph} \cong \text{BGraph}$  gives a normal form for morphisms in  $\text{BGraph}$ . This is the coproduct of the isomorphisms  $\phi : \text{MAj} \cong \text{Adj}$

and  $\psi : \text{boundP} \cong \text{Vert}$ .

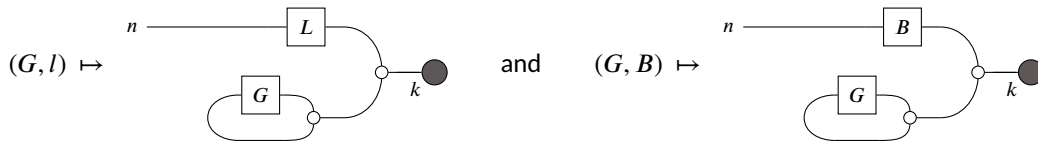


**Theorem 4.44.** *The prop of graphs BGraph is isomorphic to MGraph.*

*Proof.* By Proposition 4.30, the prop MGraph is the coproduct of MAdj and boundP. The prop MAdj is isomorphic to Adj by Theorem 4.39 and the prop boundP is isomorphic to Vert by Proposition 4.41. These imply that MGraph is isomorphic to the coproduct BGraph of Adj and Vert (Definition 4.42).  $\square$

### The operations for clique width and rank width

This section repeats the procedure of Section 4.1 for clique and rank widths. It takes the operations for clique width of Definition 2.35 introduced by Courcelle and Olariu [CO00] and the operations for rank width of Definition 2.56 introduced by Courcelle and Kanté [CK07], and examines them through a categorical lens. This time, the monoidal category BGraph specifies the categorical algebra and the operations for clique and rank widths derive from compositions and monoidal products in BGraph. This correspondence defines functions from graphs with labels and graphs with multiple labels to morphisms  $n \rightarrow 0$  in BGraph. An  $n$ -labelled graph  $(G, l)$  corresponds to the morphism  $([G], L, i, !, [()])$ , where the entry  $(i, j)$  of the matrix  $L$  is 1 if and only if  $l(i) = j$ . The matrix  $L$  is composed only of comonoid operations and symmetries. Similarly, a graph  $(G, B)$  with multiple  $n$ -labels corresponds to the morphism  $([G], B, i, !, [()])$ .

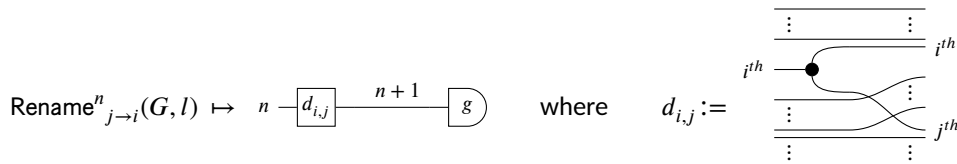


The various presentations of the operations for clique width [CER93; CO00; CV03] and rank width [CK07; CK09] define equivalent complexity measures. This becomes apparent when we express these operations as compositions and monoidal products in BGraph and its categorical structure becomes the canonical choice for the operations that define clique width and rank width. Chapter 6 proves this in detail.

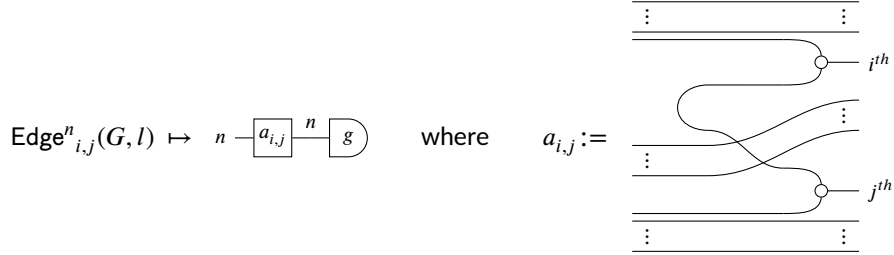
The generating graphs for clique width and rank width are the same morphisms in BGraph. The 1-labelled empty graph  $\emptyset_1$  is the discard map  $\bullet_1 : 1 \rightarrow 0$ , while the 1-labelled single vertex graph  $v_1$  is the vertex generator  $v_1 : 1 \rightarrow 0$ .

$$\emptyset_1 \mapsto 1 \text{ --- } \bullet \quad \text{and} \quad v_1 \mapsto 1 \text{ --- } \bullet$$

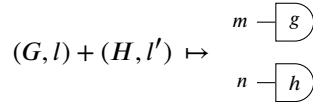
The operations for clique width derive from the categorical structure. The renaming  $\text{Rename}_{j \rightarrow i}^n(G, l)$  of label  $j$  to label  $i$  corresponds to precomposing the morphism  $g$  that corresponds to the graph  $(G, l)$  with a matrix  $d_{i,j} : n \rightarrow n + 1$  that joins the  $i^{\text{th}}$  and  $j^{\text{th}}$  outputs.



The creation of edges  $\text{Edge}_{i,j}^n(G, l)$  between the labels  $i$  and  $j$  is also a precomposition. We compose the morphism  $a_{i,j} : n \rightarrow n$ , which connects the  $i^{\text{th}}$  and  $j^{\text{th}}$  outputs through a cup, with the morphism  $g$  that corresponds to  $(G, l)$ .

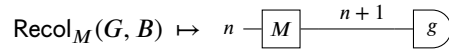


The disjoint union  $(G, l) + (H, l')$  of an  $m$ -labelled graph  $(G, l)$  and an  $n$ -labelled graph  $(H, l')$  is the monoidal product  $g \otimes h$  of the corresponding morphisms.

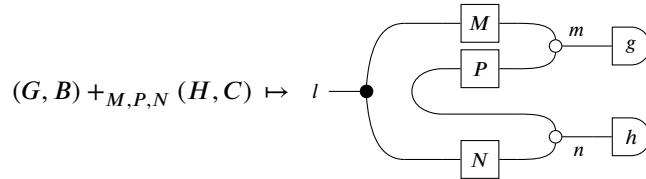


These operations together are as expressive as the operation of precomposition with a class of vertex-less morphisms in BGraph. In fact, these operations can construct all morphisms  $n \rightarrow 0$  where the connection to the left boundary is a matrix  $L$  only formed by the comonoid operations.

The operations for rank width are also derived from compositions and monoidal products in BGraph. The linear recolouring  $\text{Recol}_M(G, B)$  of the graph  $(G, B)$  with multiple labels by a matrix  $M$  corresponds to precomposing  $g$ , the morphism representing  $(G, B)$ , with the matrix  $M$ .



The bilinear product  $(G, B) +_{M,P,N} (H, C)$  of two graphs  $(G, B)$  and  $(H, C)$  with multiple labels is the composition that connects their corresponding morphisms,  $g$  and  $h$ , through  $P$  and precomposes  $M$  and  $N$  to the labels of  $g$  and  $h$ .



The operations of linear recolouring and bilinear product together define the operation of precomposition with a vertex-less morphism in BGraph. In fact, these operations can construct all morphisms  $n \rightarrow 0$  in BGraph.

## Chapter 5

# A Monoidal Algebra for Branch Width

Different categorical algebras for graphs determine different composition operations. Compositions in the category of cospans of hypergraphs join two hypergraphs by identifying some of their vertices. Section 4.1 derived the operations for tree width from compositions and monoidal products in this category. Similarly, in the category of bialgebra graphs, composing two graphs means connecting them along some dangling edges. Section 4.3 derived the operations for clique and rank widths from compositions and monoidal products in this category. What does monoidal width measure in these two cases?

Monoidal width in cospans of hypergraphs is equivalent to tree width. As recalled in Section 2.2, tree width [RS86] is based on the corresponding notion of tree decomposition, whose underlying compositional algebra is captured by cospan composition, and measures the structural complexity of graphs. The main results of this chapter and Chapter 6 validate the use of monoidal width as a measure of structural complexity.

Tree width and branch width are equivalent graph complexity measures. We leverage this fact to show equivalence between tree width and monoidal width in cospans of hypergraphs. Section 5.1 defines an inductive version of branch decompositions as an intermediate step towards the main result in Section 5.2, Theorem 5.16.

### 5.1 Inductive branch decompositions

Similarly to the Courcelle's graph expressions recalled in Section 2.3 ([BC87, Definition 3.4] and [Cou90, Definition 2.7]), monoidal decompositions in cospans of hypergraphs are also terms for hypergraphs, but where the operations are compositions and monoidal product in  $\text{Cospan}(\text{UHGraph})_*$ . This contrasts with the more combinatorial flavour of branch decompositions and makes translating between these two approaches technically involved. Following the intuitions behind Courcelle's proof of equivalence between tree width and width of graph expressions [Cou92a, Theorem 2.2], we introduce inductive branch decompositions as intermediate step between branch and monoidal decompositions. These add to branch decompositions the algebraic flavour of monoidal decompositions by relying on the inductive data structure of binary trees. In the same way that graph expressions define graphs with sources [BC87, Proposition 3.6], which appeared as rooted hypergraphs in Robertson and Seymour [RS90, Section 3], inductive decompositions define hypergraphs with sources. These are the unlabelled version of the relational structures with constants recalled in Definition 2.53. Since tree and branch decompositions of relational structures are tree and branch decompositions of their underlying hypergraph, we will work with the latter and consider the category  $\text{Cospan}(\text{UHGraph})_*$  of discrete cospans of hypergraphs instead of the category  $\text{sStruct}_\tau$  of discrete cospans of relational structures.

**Definition 5.1.** A *hypergraph with sources* is a pair  $\Gamma = (G, X)$  of a hypergraph  $G = (V, E)$  and a subset

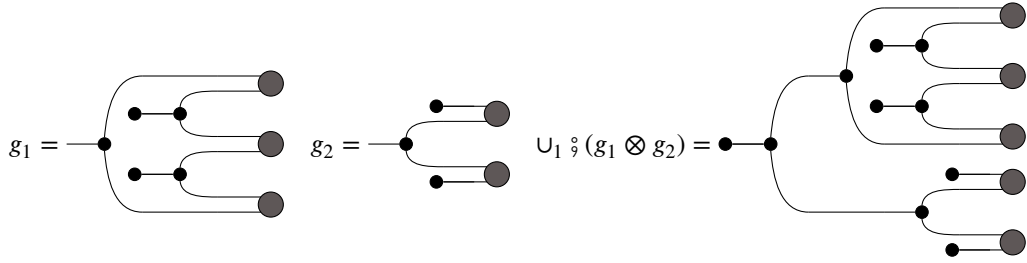
$X \subseteq V$  of its vertices, called the *sources*. Given two graphs with sources  $\Gamma = (G, X)$  and  $\Gamma' = (G', X')$ , we say that  $\Gamma'$  is a subgraph of  $\Gamma$  whenever  $G'$  is a subgraph of  $G$ .

Note that the sources of a subhypergraph  $\Gamma'$  of  $\Gamma$  need not to appear as sources of  $\Gamma$ , nor vice versa. In fact, if  $\Gamma$  is obtained by identifying all the sources of a hypergraph  $\Gamma_1$  with some of the sources of another hypergraph  $\Gamma_2$ , the sources of  $\Gamma$  and  $\Gamma_1$  will be disjoint. A hypergraph with sources  $\Gamma = (G, X)$  can be seen as a morphism  $g: X \rightarrow \emptyset$  in  $\text{Cospan}(\text{UHGraph})_*$ :  $g =: X \rightarrow G \leftarrow \emptyset$ , where the legs of the cospan are  $\iota: X \rightarrow V$  and  $\jmath: \emptyset \rightarrow V$ .

*Example 5.2.* Sources are marked vertices in the graph and are thought of as an interface that can be glued with that of another graph. Two graphs sharing the sources, as illustrated below, can be “glued together”:



These two graphs correspond to two morphisms  $g_1, g_2: 1 \rightarrow \emptyset$  in  $\text{Cospan}(\text{UHGraph})_*$  that can be composed to obtain the rightmost graph  $\cup_1 \circ (g_1 \otimes g_2)$ .



**Definition 5.3.** A *binary tree*  $T \in \mathcal{T}_\Gamma$  for a hypergraph  $\Gamma$  is defined inductively.

$$T ::= (\Gamma) \quad \text{if } |\text{edges}(\Gamma)| \leq 1 \\ | (T_1 - \Gamma - T_2) \quad \text{if } T_1 \in \mathcal{T}_{\Gamma_1}, T_2 \in \mathcal{T}_{\Gamma_2} \text{ and } \Gamma_1, \Gamma_2 \text{ are subgraphs of } \Gamma$$

An inductive branch decomposition of a hypergraph with sources  $\Gamma$  is a binary tree  $T \in \mathcal{T}_\Gamma$  satisfying some conditions such that, identifying the common sources in  $\Gamma_1$  and  $\Gamma_2$ , we obtain  $\Gamma$ .

**Definition 5.4.** An *inductive branch decomposition* of a hypergraph with sources  $\Gamma = ((V, E), X)$  is a binary tree  $T \in \mathcal{T}_\Gamma$  where either  $\Gamma$  has at most one edge and  $T = (\Gamma)$ , or  $T = (T_1 - \Gamma - T_2)$  and  $T_i \in \mathcal{T}_{\Gamma_i}$  are inductive branch decompositions of subhypergraphs  $\Gamma_i = ((V_i, E_i), X_i)$  of  $\Gamma$  such that:

- The edges are partitioned in two,  $E = E_1 \sqcup E_2$ , and  $V = V_1 \cup V_2$ ;
- The sources are those vertices shared with the original sources as well as those shared with the other subhypergraph,  $X_i = (V_1 \cap V_2) \cup (X \cap V_i)$ .

*Remark 5.5.* Note that  $\text{ends}(E_i) \subseteq V_i$  and that not all subtrees of a decomposition  $T$  are themselves decompositions: only those  $T'$  that contain all the nodes in  $T$  that are below the root of  $T'$ . We call these *full subtrees*,  $T' \leq T$ , and indicate with  $\lambda(T')$  the subhypergraph of  $\Gamma$  that  $T'$  is a decomposition of. We will sometimes write  $\Gamma_i = \lambda(T_i)$ ,  $V_i = \text{vertices}(\Gamma_i)$  and  $X_i = \text{sources}(\Gamma_i)$ . Then,

$$\text{sources}(\Gamma_i) = (\text{vertices}(\Gamma_1) \cap \text{vertices}(\Gamma_2)) \cup (\text{sources}(\Gamma) \cap \text{vertices}(\Gamma_i)).$$

At every step in a decomposition, two graphs with sources are composed along the common boundary identifying some sources of one graph with some sources of the other. The size of the biggest of these boundaries determines the width of the decomposition.



**Definition 5.6.** The *width* of an inductive branch decomposition  $T$  of a hypergraph with sources  $\Gamma = (G, X)$ , with sources  $X$ , is defined inductively:

$$\begin{aligned} \text{wd}(T) &:= |X| && \text{if } T = (\Gamma), \\ &| \max\{\text{wd}(T_1), \text{wd}(T_2), |X|\} && \text{if } T = (T_1 - \Gamma - T_2). \end{aligned}$$

Expanding this expression, we obtain

$$\text{wd}(T) = \max_{T' \text{ full subtree of } T} |\text{sources}(\lambda(T'))|.$$

### Equivalence with branch width

Inductive branch width coincides with branch width (Proposition 5.10). We show their equivalence by constructing, in Lemma 5.8, a branch decomposition from an inductive one and vice versa, in Lemma 5.9, preserving the width. For defining these mappings, we find an explicit expression for the set of sources of subgraphs  $\lambda(T_0)$  corresponding to full subtrees  $T_0$  of a decomposition  $T$ .

**Lemma 5.7.** *Let  $T$  be an inductive branch decomposition of a hypergraph with sources  $\Gamma$  and  $T_0$  be a full subtree of  $T$ . Then,*

$$\text{sources}(\lambda(T_0)) = \text{vertices}(\lambda(T_0)) \cap \left( X \cup \bigcup_{T' \not\leq T_0} \text{vertices}(\lambda(T')) \right),$$

where  $T' \not\leq T_0$  denotes a full subtree  $T'$  of  $T$  whose intersection with  $T_0$  is empty.

*Proof.* Proceed by induction on the decomposition tree  $T$ . If it is a leaf,  $T = (\Gamma)$ , then its subtree is also a leaf,  $T_0 = (\Gamma)$ , and we are done.

If  $T = (T_1 - \Gamma - T_2)$ , then either  $T_0$  is a full subtree of  $T_1$ , or it is a full subtree of  $T_2$  or it coincides with  $T$ . If  $T_0$  coincides with  $T$ , then their sources coincide and the statement holds because  $\text{sources}(\lambda(T_0)) = X = V \cap X$ . Suppose that  $T_0$  is a full subtree of  $T_1$ . Then, by applying the induction hypothesis, Remark 5.5, and using the fact that  $\lambda(T_0) \subseteq \lambda(T_1)$ , we compute its sources

$$\begin{aligned} &\text{sources}(\lambda(T_0)) \\ &= \text{vertices}(\lambda(T_0)) \cap \left( \text{sources}(\lambda(T_1)) \cup \bigcup_{T' \leq T_1, T' \not\leq T_0} \text{vertices}(\lambda(T')) \right) \\ &= \text{vertices}(\lambda(T_0)) \cap \left( (\text{vertices}(\lambda(T_1)) \cap (\text{vertices}(\lambda(T_2)) \cup X)) \cup \bigcup_{T'} \text{vertices}(\lambda(T')) \right) \\ &= \text{vertices}(\lambda(T_0)) \cap \left( \text{vertices}(\lambda(T_2)) \cup X \cup \bigcup_{T' \leq T_1, T' \not\leq T_0} \text{vertices}(\lambda(T')) \right) \\ &= \text{vertices}(\lambda(T_0)) \cap \left( X \cup \bigcup_{T' \leq T, T' \not\leq T_0} \text{vertices}(\lambda(T')) \right) \end{aligned}$$

A similar computation can be done if  $T_0$  is a full subtree of  $T_2$ . □

Given an inductive branch decomposition  $T$ , the branch decomposition  $\mathcal{I}^\dagger(T)$  is obtained by forgetting the labelling of its internal nodes and which node corresponds to the root.

**Lemma 5.8.** *Let  $T$  be an inductive branch decomposition of a hypergraph with sources  $\Gamma = (G, X)$ . Then, there is a branch decomposition  $\mathcal{I}^\dagger(T)$  of its underlying hypergraph  $G$  of bounded width:  $\text{wd}(\mathcal{I}^\dagger(T)) \leq \text{wd}(T)$ .*

*Proof.* A binary tree is, in particular, a subcubic tree. Then, we can define  $Y$  to be the unlabelled tree underlying  $T$ . If the label of a leaf  $l$  of  $T$  is a subhypergraph of  $\Gamma$  with one edge  $e_l$ , then we keep the leaf, otherwise, if the subhypergraph is discrete, we remove the leaf  $l$  from  $Y$ . Then, there is a bijection  $b: \text{leaves}(Y) \rightarrow \text{edges}(G)$  such that  $b(l) := e_l$ . Then,  $(Y, b)$  is a branch decomposition of  $G$  and we can define  $\mathcal{I}^\dagger(T) := (Y, b)$ .

By construction, if  $e \in \text{edges}(Y)$  then  $e \in \text{edges}(T)$ . Let  $\{v, w\} = \text{ends}(e)$  with  $v$  parent of  $w$  in  $T$  and let  $T_w$  the full subtree of  $T$  with root  $w$ . Let  $\{E_v, E_w\}$  be the (non-trivial) partition of  $E$  induced by  $e$ . Then, for the edges sets,  $E_w = \text{edges}(\lambda(T_w))$  and  $E_v = \bigcup_{T' \not\leq T_w} \text{edges}(\lambda(T'))$ , and, for the vertices sets,  $\text{ends}(E_w) \subseteq \text{vertices}(\lambda(T_w))$  and  $\text{ends}(E_v) \subseteq \bigcup_{T' \not\leq T_w} \text{vertices}(\lambda(T'))$ . Using these inclusions and applying Lemma 5.7,

$$\begin{aligned}
\text{ord}(e) & & \text{wd}(Y, b) \\
:= |\text{ends}(E_w) \cap \text{ends}(E_v)| & & := \max_{e \in \text{edges}(Y)} \text{ord}(e) \\
\leq |\text{vertices}(\lambda(T_w)) \cap \bigcup_{T' \not\leq T_w} \text{vertices}(\lambda(T'))| & & \leq \max_{T' < T} |\text{sources}(\lambda(T'))| \\
\leq |\text{vertices}(\lambda(T_w)) \cap (X \cup \bigcup_{T' \not\leq T_w} \text{vertices}(\lambda(T')))| & & \leq \max_{T' \leq T} |\text{sources}(\lambda(T'))| \\
= |\text{sources}(\lambda(T_w))| & & = \text{wd}(T)
\end{aligned}$$

□

Given a branch decomposition  $(Y, b)$  of a hypergraph  $G$ , we pick an edge of  $Y$  and subdivide it to add an extra vertex which will be the root. The labelling of the internal nodes comes as a consequence and define an inductive branch decomposition  $\mathcal{I}(Y, b)$  of the same width.

**Lemma 5.9.** *Let  $(Y, b)$  be a branch decomposition of a hypergraph  $G$  and let  $\Gamma = (G, X)$  be a hypergraph with sources  $X$  whose underlying hypergraph is  $G$ . Then, there is a branch decomposition  $\mathcal{I}(Y, b)$  of  $\Gamma$  of bounded width:  $\text{wd}(\mathcal{I}(Y, b)) \leq \text{wd}(Y, b) + |X|$ .*

*Proof.* Proceed by induction on  $|\text{edges}(Y)|$ . If  $Y$  has no edges, then either  $G$  has no edges and  $(Y, b) = ()$  or  $G$  has only one edge  $e_l$  and  $(Y, b) = (e_l)$ . In either case, define  $\mathcal{I}(Y, b) := (\Gamma)$  and  $\text{wd}(\mathcal{I}(Y, b)) := |X| \leq \text{wd}(Y, b) + |X|$ .

If  $Y$  has at least one edge  $e$ , then  $Y = Y_1 \overset{e}{-} Y_2$  with  $Y_i$  a subcubic tree. Let  $E_i = b(\text{leaves}(Y_i))$  be the sets of edges of  $G$  indicated by the leaves of  $Y_i$ . Then,  $E_1 \sqcup E_2 = E$ . By induction hypothesis, there are inductive branch decompositions  $T_i := \mathcal{I}(Y_i, b_i)$  of  $\Gamma_i = (G_i, X_i)$ , where  $V_1 := \text{ends}(E_1)$ ,  $V_2 := \text{ends}(E_2) \cup (V \setminus V_1)$ ,  $X_i := (V_1 \cap V_2) \cup (V_i \cap X)$  and  $G_i := (V_i, E_i)$ . Then, the tree  $\mathcal{I}(Y, b) := (T_1 - \Gamma - T_2)$  is an inductive branch decomposition of  $\Gamma$  and, applying Lemma 5.7,

$$\begin{aligned}
\text{wd}(\mathcal{I}(Y, b)) & \\
:= \max\{\text{wd}(T_1), |X|, \text{wd}(T_2)\} & \\
= \max_{T' \leq T} |\text{sources}(\lambda(T'))| &
\end{aligned}$$

$$\begin{aligned}
&\leq \max_{T' \leq T} |\text{vertices}(\lambda(T')) \cap \text{ends}(E \setminus \text{edges}(\lambda(T')))| + |X| \\
&= \max_{e \in \text{edges}(Y)} \text{ord}(e) + |X| \\
&=: \text{wd}(Y, b) + |X|
\end{aligned}$$

□

Combining Lemmas 5.8 and 5.9, we obtain the equivalence between branch width and inductive branch width.

**Proposition 5.10.** *For hypergraphs with no sources, branch width and inductive branch width coincide.*

## 5.2 Bounding branch width

Monoidal width in cospans of hypergraphs is equivalent to branch width (Theorem 5.16) and, as a consequence, it is also equivalent to tree width (Corollary 5.17). In particular, the monoidal width of a hypergraph is at most its branch width +1 and at least half of it. Proposition 5.13 shows the upper bound by mapping a branch decomposition to a monoidal decomposition of the same hypergraph with bounded width. Similarly, Proposition 5.15 defines a branch decomposition from a monoidal decomposition to show the lower bound.

The instantiation of monoidal width in cospans of hypergraphs needs an appropriate weight function. The width of a tree decomposition depends on the number of vertices contained in each bag, thus we define the weight function for  $\text{Cospan}(\text{UHGraph})_*$  to count the number of vertices of the apex graph in each cospan.

**Definition 5.11.** For a morphism  $g : X \rightarrow Y$  in  $\text{Cospan}(\text{UHGraph})_*$ , the weight function  $w$  is defined as  $w(g) := |V|$ , where  $V$  is the set of vertices of the apex of  $g$ , i.e.  $g = : X \rightarrow G \leftarrow Y :$  and  $G = (V, E)$ .

With this definition, the identity on  $X$  weights  $|X|$  and compositions along  $X$  cost  $|X|$ . This definition gives a weight function.

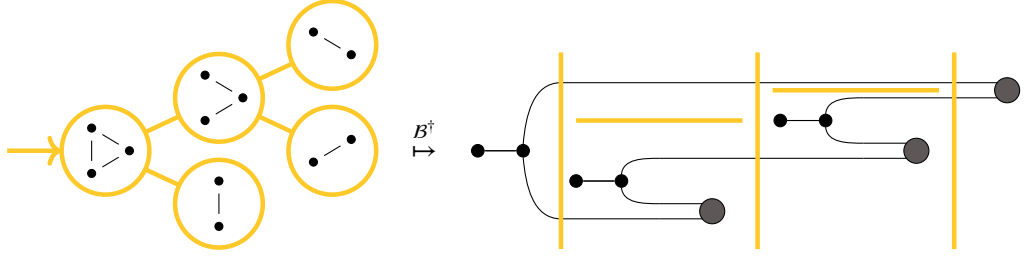
**Lemma 5.12.** *The function  $w$  in Definition 5.11 satisfies the conditions in Definition 3.3 for a weight function in the monoidal category  $\text{Cospan}(\text{UHGraph})_*$ .*

*Proof.* For  $f : X \rightarrow Y$ ,  $g : Y \rightarrow Z$  and  $f' : X' \rightarrow Y'$  with sets of vertices  $V$ ,  $W$  and  $V'$ , we can bound the weights of  $f \circ_Y g$  and  $f \otimes f'$ .

$$\begin{array}{ll}
w(f \circ_Y g) & w(f \otimes f') \\
:= |V +_Y W| & := |V + V'| \\
\leq |V| + |W| + |Y| & = |V| + |V'| \\
=: w(f) + w(g) + w(Y) & =: w(f) + w(f')
\end{array}$$

□

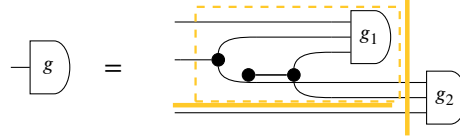
A branch decomposition divides a hypergraph into one-edge subhypergraphs. Given a branch decomposition of a hypergraph  $\Gamma$  with sources, the corresponding monoidal decomposition is defined by taking all the one-edge subhypergraphs and composing them according to the tree structure of the branch decomposition. For example, the monoidal decomposition shown below right corresponds to the inductive branch decomposition of 3-clique at its left: the three edge generators  $\text{---}\bullet$  are connected following the shape of the branch decomposition.



**Proposition 5.13.** Let  $T$  be an inductive branch decomposition of a hypergraph with sources  $\Gamma = (G, X)$ . Let  $g := \iota : X \rightarrow G \leftarrow \emptyset$  be the corresponding cospan and let  $\gamma(G)$  indicate the hyperedge size of  $G$ . Then, there is a monoidal decomposition  $\mathcal{B}^\dagger(T) \in D_g$  of bounded width:  $\text{wd}(\mathcal{B}^\dagger(T)) \leq \max\{\text{wd}(T) + 1, \gamma(G)\}$ .

*Proof.* Let  $G = (V, E)$  and proceed by induction on the decomposition tree  $T$ . If the tree  $T = (\Gamma)$  is composed of only one leaf, then the label  $\Gamma$  of this leaf must have at most one hyperedge with  $\gamma(G)$  endpoints and  $\text{wd}(T) := |X|$ . We define the corresponding monoidal decomposition to also consist of only a leaf,  $\mathcal{B}^\dagger(T) := (g)$ , and obtain the desired bound  $\text{wd}(\mathcal{B}^\dagger(T)) = \max\{|X|, \gamma(G)\} = \max\{\text{wd}(T), \gamma(G)\}$ .

If  $T = (T_1 - \Gamma - T_2)$ , then, by definition of inductive branch decomposition,  $T$  is composed of two subtrees  $T_1$  and  $T_2$  that give branch decompositions of  $\Gamma_1 = (G_1, X_1)$  and  $\Gamma_2 = (G_2, X_2)$ . There are three conditions imposed by the definition on these subgraphs  $G_i = (V_i, E_i)$ :  $E = E_1 \sqcup E_2$  with  $E_i \neq \emptyset$ ,  $V_1 \cup V_2 = V$ , and  $X_i = (V_1 \cap V_2) \cup (X \cap V_i)$ . Let  $g_i = \iota : X_i \rightarrow G_i \leftarrow \emptyset$  be the morphism in  $\text{Cospan}(\text{UHGraph})_*$  corresponding to  $\Gamma_i$ . Then, we decompose  $g$  in terms of identities, the structure of  $\text{Cospan}(\text{UHGraph})_*$ , and its subgraphs  $g_1$  and  $g_2$ , separating their boundaries into  $X_1 \setminus X_2$ ,  $(X_1 \cap X_2) \setminus X$ ,  $X_1 \cap X_2 \cap X$ , and  $X_2 \setminus X_1$ :



By induction hypothesis, there are monoidal decompositions  $\mathcal{B}^\dagger(T_i)$  of the morphisms  $g_i$  of bounded width:  $\text{wd}(\mathcal{B}^\dagger(T_i)) \leq \max\{\text{wd}(T_i) + 1, \gamma(G_i)\}$ . By Lemma 3.11, there is a monoidal decomposition  $\mathcal{C}(\mathcal{B}^\dagger(T_1))$  of the morphism in the above dashed box of bounded width:  $\text{wd}(\mathcal{C}(\mathcal{B}^\dagger(T_1))) \leq \max\{\text{wd}(\mathcal{B}^\dagger(T_1)), |X_1| + 1\}$ . Using this decomposition, we can define the monoidal decomposition given by the cuts in the figure above.

$$\mathcal{B}^\dagger(T) := ((\mathcal{C}(\mathcal{B}^\dagger(T_1))) - \otimes - \mathbb{1}_{X_2 \setminus X_1} - \circlearrowleft_{X_2} - \mathcal{B}^\dagger(T_2)).$$

We can bound its width by applying Lemma 3.11, the induction hypothesis and the relevant definitions of width ( $|X_i| \leq \text{wd}(T_i)$  by Definitions 5.6 and 5.11).

$$\begin{aligned} & \text{wd}(\mathcal{B}^\dagger(T)) \\ & := \max\{\text{wd}(\mathcal{C}(\mathcal{B}^\dagger(T_1))), \text{wd}(\mathcal{B}^\dagger(T_2)), |X_2|\} \\ & \leq \max\{\text{wd}(\mathcal{B}^\dagger(T_1)), \text{wd}(\mathcal{B}^\dagger(T_2)), |X_1| + 1, |X_2|\} \\ & \leq \max\{\text{wd}(T_1) + 1, \gamma(G_1), \text{wd}(T_2) + 1, \gamma(G_2), |X_1| + 1, |X_2|\} \\ & \leq \max\{\max\{\text{wd}(T_1), \text{wd}(T_2), |X_1|, |X_2|\} + 1, \gamma(G_1), \gamma(G_2)\} \\ & \leq \max\{\max\{\text{wd}(T_1), \text{wd}(T_2), |X|\} + 1, \gamma(G)\} \\ & =: \max\{\text{wd}(T) + 1, \gamma(G)\} \end{aligned}$$

□

The mapping from monoidal decompositions to inductive branch decompositions follows a similar idea to the previous one and also proceeds by induction on the decomposition tree. It requires some extra bureaucracy to handle the case of composition nodes, for which the following lemma is needed.

**Lemma 5.14.** *Consider a hypergraph with sources  $\Gamma = ((V, E), X)$ , a function  $\phi : V \rightarrow W$  and define the hypergraph with sources  $\phi(\Gamma) := ((\phi(V), E), \phi(X))$ . Suppose there is an inductive branch decomposition  $T$  of  $\Gamma$ . Then, there is an inductive branch decomposition  $\phi(T)$  of  $\phi(\Gamma)$  of bounded width:  $\text{wd}(\phi(T)) \leq \text{wd}(T)$ .*

*Proof.* Proceed by induction on the decomposition tree  $T$ . If  $T = (\Gamma)$  is just a leaf, then define  $\phi(T) := (\phi(\Gamma))$  to be a leaf as well. Its width is bounded by that of  $T$ :  $\text{wd}(\phi(T)) := |\phi(X)| \leq |X| =: \text{wd}(T)$ .

Otherwise,  $T = (T_1 - \Gamma - T_2)$  has two subtrees, where  $T_i$  is an inductive branch decomposition of  $\Gamma_i = ((V_i, E_i), X_i)$ . By the definition of inductive branch decomposition (Definition 5.4),  $E = E_1 \sqcup E_2$ ,  $V = V_1 \cup V_2$  and  $X_i = (V_1 \cap V_2) \cup (X \cap V_i)$ . Denote with  $\phi_1 : V_1 \rightarrow W$  and  $\phi_2 : V_2 \rightarrow W$  the compositions of  $\phi$  with the inclusions  $\iota_1 : V_1 \hookrightarrow V$  and  $\iota_2 : V_2 \hookrightarrow V$ . By induction hypothesis, there are inductive branch decompositions  $\phi_i(T_i)$  of  $\phi_i(\Gamma_i)$  of bounded width,  $\text{wd}(\phi_i(T_i)) \leq \text{wd}(T_i)$ . Define  $\phi(T) := (\phi_1(T_1) - \phi(\Gamma) - \phi_2(T_2))$  by combining the inductive branch decompositions of  $\phi_1(\Gamma_1)$  and  $\phi_2(\Gamma_2)$ . This is an inductive branch decomposition of  $\phi(\Gamma)$  because  $E = E_1 \sqcup E_2$ ,  $\phi(V) = \phi(V_1 \cup V_2) = \phi(\iota_1(V_1) \cup \iota_2(V_2)) = \phi_1(V_1) \cup \phi_2(V_2)$ , and  $\phi_i(X_i) = \phi_i((V_1 \cap V_2) \cup (X \cap V_i)) = \phi((V_1 \cap V_2) \cup (X \cap V_i)) = (\phi(V_1) \cap \phi(V_2)) \cup (\phi(X) \cap \phi(V_i))$ . The width of  $\phi(T)$  is bounded by that of  $T$ :

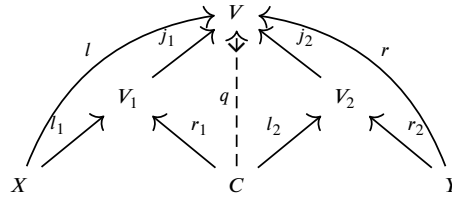
$$\begin{aligned} \text{wd}(\phi(T)) &:= \max\{\text{wd}(\phi_1(T_1)), \text{wd}(\phi_2(T_2)), |\phi(X)|\} \\ &\leq \max\{\text{wd}(T_1), \text{wd}(T_2), |X|\} \\ &=: \text{wd}(T) \end{aligned}$$

□

**Proposition 5.15.** *Let  $d \in D_g$  be a monoidal decomposition of a morphism  $g = l : X \rightarrow G \leftarrow Y : r$  in  $\text{Cospan}(\text{UHGraph})_*$ . Consider the hypergraph with sources  $\Gamma := (G, l(X) \cup r(Y))$  corresponding to  $g$ . Then, there is an inductive branch decomposition  $\mathcal{B}(d)$  of  $\Gamma$  of bounded width:  $\text{wd}(\mathcal{B}(d)) \leq 2 \cdot \max\{\text{wd}(d), |X|, |Y|\}$ .*

*Proof.* Proceed by induction on  $d$ . If  $d = (g)$  is just a leaf, then define  $\mathcal{B}(d)$  to be any inductive branch decomposition of  $\Gamma$ . The width of an inductive branch decomposition of  $\Gamma$  is bounded by the number of vertices of  $\Gamma$  and, as a consequence, by the width of  $d$ :  $\text{wd}(\mathcal{B}(d)) \leq |V| =: \text{wd}(d) \leq 2 \cdot \max\{\text{wd}(d), |X|, |Y|\}$ .

Suppose that  $d = (d_1 - \circ_C - d_2)$  starts with a composition node. Then,  $g = g_1 \circ g_2$  for two morphisms  $g_1 = l_1 : X \rightarrow G_1 \leftarrow C : r_1$  and  $g_2 = l_2 : C \rightarrow G_2 \leftarrow Y : r_2$ .



By induction hypothesis, there are inductive branch decompositions  $\mathcal{B}(d_1)$  and  $\mathcal{B}(d_2)$  of the hypergraphs with sources  $\Gamma_1 := (G_1, l_1(X) \cup r_1(C))$  and  $\Gamma_2 := (G_2, l_2(C) \cup r_2(Y))$  of bounded width:  $\text{wd}(\mathcal{B}(d_1)) \leq 2 \cdot \max\{\text{wd}(d_1), |X|, |C|\}$  and  $\text{wd}(\mathcal{B}(d_2)) \leq 2 \cdot \max\{\text{wd}(d_2), |Y|, |C|\}$ . We apply Lemma 5.14 to the decompositions  $\mathcal{B}(d_i)$  and functions  $j_i$  to obtain inductive branch decompositions  $j_i(\mathcal{B}(d_i))$  of  $j_i(\Gamma_i)$  bounded width:  $\text{wd}(j_i(\mathcal{B}(d_i))) \leq \text{wd}(\mathcal{B}(d_i))$ . These two decompositions combine into an inductive branch decomposition

$\mathcal{B}(d) := (j_1(\mathcal{B}(d_1)) - \Gamma - j_2(\mathcal{B}(d_2)))$ . This is, indeed, an inductive branch decomposition of  $\Gamma$  because it satisfies the condition on the edges and vertices,  $E = E_1 \sqcup E_2$  and  $V = j_1(V_1) \cup j_2(V_2)$ , and the conditions on the sources,

$$\begin{aligned}
j_1(l_1(X) \cup r_1(C)) & & j_2(l_2(C) \cup r_2(Y)) \\
= j_1(l_1(X)) \cup j_1(r_1(C)) & & = j_2(l_2(C)) \cup j_2(r_2(Y)) \\
= l(X) \cup q(C) & & = q(C) \cup r(Y) \\
= l(X) \cup (j_1(V_1) \cap j_2(V_2)) & & = (j_1(V_1) \cap j_2(V_2)) \cup r(Y) \\
= ((l(X) \cup r(Y)) \cap j_1(V_1)) \cup (j_1(V_1) \cap j_2(V_2)) & & = ((l(X) \cup r(Y)) \cap j_2(V_2)) \cup (j_1(V_1) \cap j_2(V_2))
\end{aligned}$$

in Definition 5.4. The width of  $\mathcal{B}(d)$  is bounded.

$$\begin{aligned}
\text{wd}(\mathcal{B}(d)) & \\
& := \max\{\text{wd}(j_1(\mathcal{B}(d_1))), |l(X) \cup r(Y)|, \text{wd}(j_2(\mathcal{B}(d_2)))\} \\
& \leq \max\{\text{wd}(\mathcal{B}(d_1)), |l(X)| + |r(Y)|, \text{wd}(\mathcal{B}(d_2))\} \\
& \leq \max\{2\text{wd}(d_1), 2|X|, 2|C|, |X| + |Y|, \text{wd}(d_2), 2|C|, 2|Y|\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), |C|, \text{wd}(d_2), |X|, |Y|\} \\
& =: 2 \cdot \max\{\text{wd}(d), |X|, |Y|\}
\end{aligned}$$

Suppose that  $d = (d_1 - \otimes - d_2)$  starts with a monoidal product node. Then,  $g = g_1 \otimes g_2$  for two morphisms  $g_1 = l_1 : X_1 \rightarrow G_1 \leftarrow Y_1 : r_1$  and  $g_2 = l_2 : X_2 \rightarrow G_2 \leftarrow Y_2 : r_2$ . By induction hypothesis, there are inductive branch decompositions  $\mathcal{B}(d_1)$  and  $\mathcal{B}(d_2)$  of the hypergraphs with sources  $\Gamma_1 := (G_1, l_1(X_1) \cup r_1(Y_1))$  and  $\Gamma_2 := (G_2, l_2(X_2) \cup r_2(Y_2))$  of bounded width:  $\text{wd}(\mathcal{B}(d_1)) \leq 2 \cdot \max\{\text{wd}(d_1), |X_1|, |Y_1|\}$  and  $\text{wd}(\mathcal{B}(d_2)) \leq 2 \cdot \max\{\text{wd}(d_2), |X_2|, |Y_2|\}$ . These decompositions combine into an inductive branch decomposition  $\mathcal{B}(d) := (\mathcal{B}(d_1) - \Gamma - \mathcal{B}(d_2))$ . This is, indeed, a decomposition of  $\Gamma$  because it satisfies the conditions of Definition 5.4:  $E = E_1 \sqcup E_2$ ,  $V = V_1 \cup V_2$  and  $l_i(X_i) \cup r_i(Y_i) = ((l(X) \cup r(Y)) \cap V_i) \cup (V_i \cap V_2)$ . The width of  $\mathcal{B}(d)$  is bounded.

$$\begin{aligned}
\text{wd}(\mathcal{B}(d)) & \\
& \leq \max\{\text{wd}(\mathcal{B}(d_1)), |l(X) \cup r(Y)|, \text{wd}(\mathcal{B}(d_2))\} \\
& \leq \max\{2\text{wd}(d_1), 2|X_1|, 2|Y_1|, |X| + |Y|, \text{wd}(d_2), 2|X_2|, 2|Y_2|\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), \text{wd}(d_2), |X|, |Y|\} \\
& =: 2 \cdot \max\{\text{wd}(d), |X|, |Y|\}
\end{aligned}$$

□

Theorem 5.16 summarises Propositions 5.10, 5.13 and 5.15.

**Theorem 5.16.** *Let  $G$  be a graph and  $g = : \emptyset \rightarrow G \leftarrow \emptyset$  : be the corresponding morphism of  $\text{Cospan}(\text{UHGraph})_*$ . Then,  $\frac{1}{2} \cdot \text{bwd}(G) \leq \text{mwd}(g) \leq \text{bwd}(G) + 1$ .*

With this result and Theorem 2.30, we obtain equivalence with tree width.

**Corollary 5.17.** *Tree width is equivalent to monoidal width in  $\text{Cospan}(\text{UHGraph})_*$ .*

## Chapter 6

# A Monoidal Algebra for Rank Width

Chapter 5 showed that composition in cospans of hypergraphs captures the operation that underlies tree decompositions. As a consequence, monoidal width in  $\text{Cospan}(\text{UHGraph})_*$  is equivalent to tree width. This chapter concerns rank width. As anticipated in Section 4.3, the operations for clique and rank widths derive from the categorical algebra of the prop  $\text{BGraph}$ . Here, we show that the prop  $\text{BGraph}$  captures the algebra of composition underlying rank width, making monoidal width in this category equivalent to rank width and, as a consequence, to clique width.

Rank width relies on the corresponding notion of rank decomposition, which we recalled in Section 2.2. Clique width and rank width are equivalent graph complexity measures. We leverage this fact to show equivalence between clique width and monoidal width in the category of bialgebra graphs. As an intermediate step towards the main result of this chapter, Theorem 6.19 in Section 6.2, we introduce inductive rank decompositions in Section 6.1.

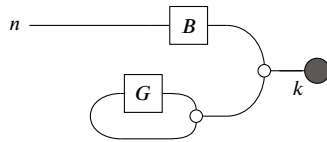
### 6.1 Inductive rank decompositions

As for branch decompositions, inductive rank decompositions are an intermediate step to add the inductive flavour of monoidal decompositions to rank decompositions. Inductive rank decompositions are binary trees and give expressions that define graphs whose interfaces are some marked “dangling edges”.

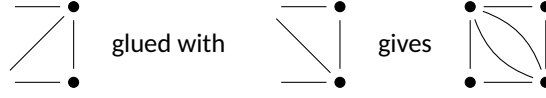
**Definition 6.1.** A *graph with dangling edges* is a pair  $\Gamma = ([G], B)$  of an adjacency matrix  $G \in \text{Mat}_{\mathbb{N}}(k, k)$  that records the connectivity of the graph and a matrix  $B \in \text{Mat}_{\mathbb{N}}(k, n)$  that records the dangling edges connected to  $n$  boundary ports. Two graphs with dangling edges  $\Gamma = ([G], B)$  and  $\Gamma' = ([G'], B')$  are equal if they encode the same graph with a different ordering on the vertices, i.e. there is a permutation matrix  $P \in \text{Mat}_{\mathbb{N}}(k, k)$  such that  $G = P \cdot G' \cdot P^{\top}$  and  $B = P \cdot B'$ .

We will sometimes write  $G \in \text{adjacency}(\Gamma)$  and  $B = \text{sources}(\Gamma)$ .

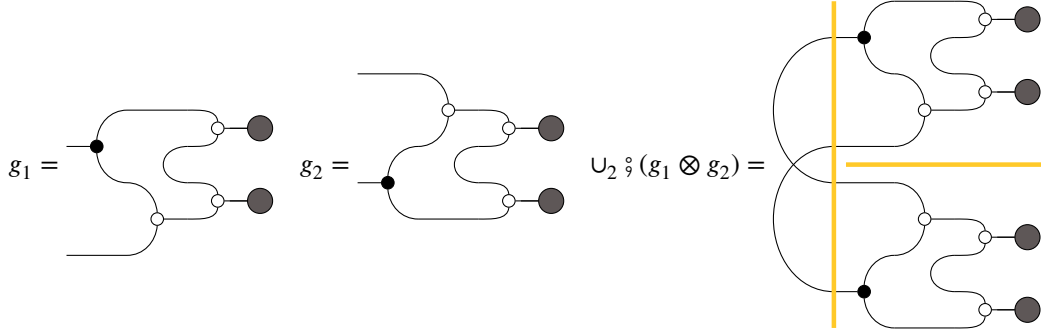
A graph with dangling edges  $\Gamma = ([G], B)$  can be seen as a morphism  $n \rightarrow 0$  in  $\text{BGraph}$ .



*Example 6.2.* Two graphs with the same ports, as illustrated below, can be “glued” together:



These two graphs correspond to two morphisms  $g_1, g_2 : 2 \rightarrow 0$  in BGraph that can be composed to obtain the rightmost graph  $\cup_2 \circ (g_1 \otimes g_2)$ .



An inductive rank decomposition of  $\Gamma$  is a binary tree satisfying some conditions that ensure that composing the dangling edges of  $\Gamma_1$  with those of  $\Gamma_2$  gives  $\Gamma$ .

**Definition 6.3.** A binary tree  $T \in \mathcal{T}_\Gamma$  for a graph  $\Gamma$  is defined inductively.

$$T ::= (\Gamma) \quad \text{if } |\text{vertices}(\Gamma)| \leq 1 \\ | (T_1 - \Gamma - T_2) \quad \text{if } T_1 \in \mathcal{T}_{\Gamma_1}, T_2 \in \mathcal{T}_{\Gamma_2} \text{ and } \Gamma_1, \Gamma_2 \text{ are subgraphs of } \Gamma$$

**Definition 6.4.** An inductive rank decomposition of a graph with dangling edges  $\Gamma = ([G], B)$  is a binary tree  $T \in \mathcal{T}_\Gamma$  where either:  $\Gamma$  has at most one vertex and  $T = (\Gamma)$ ; or  $T = (T_1 - \Gamma - T_2)$  and  $T_i \in \mathcal{T}_{\Gamma_i}$  are inductive rank decompositions of subgraphs  $\Gamma_i = ([G_i], B_i)$  of  $\Gamma$  such that:

- The vertices are partitioned in two,  $[G] = \left[ \begin{pmatrix} G_1 & C \\ 0 & G_2 \end{pmatrix} \right]$ ;
- The dangling edges are those to the original boundary and to the other subgraph,  $B_1 = (A_1 \mid C)$  and  $B_2 = (A_2 \mid C^\top)$ , where  $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ .

We will sometimes write  $\Gamma_i = \lambda(T_i)$ ,  $G_i = \text{adjacency}(\Gamma_i)$  and  $B_i = \text{sources}(\Gamma_i)$ .

*Remark 6.5.* Thanks to the equivalence relation on graphs with dangling edges, we can always assume that the rows of  $G$  and  $B$  are ordered like the leaves of  $T$  so that we can split  $B$  horizontally to get  $A_1$  and  $A_2$ .

At every step in a decomposition, two graphs with dangling edges are composed along a common boundary. The most complex of these boundaries determines the width of the decomposition.

**Definition 6.6.** The *width* of an inductive rank decomposition  $T$  of a graph with dangling edges  $\Gamma = ([G], B)$ , with boundary matrix  $B$ , is defined inductively:

$$\text{wd}(T) := \text{rk}(B) \quad \text{if } T = (\Gamma), \\ | \max\{\text{wd}(T_1), \text{wd}(T_2), \text{rk}(B)\} \quad \text{if } T = (T_1 - \Gamma - T_2).$$

Expanding this expression, we obtain

$$\text{wd}(T) = \max_{T' \text{ full subtree of } T} \text{rk}(\text{sources}(\lambda(T'))).$$



### Equivalence with rank width

Rank width coincides with inductive rank width as inductive rank decompositions can be transformed into rank decompositions while preserving their width (Lemma 6.8), and vice versa (Lemma 6.9). The width of an inductive rank decomposition of a graph  $\Gamma$  is defined inductively. The next lemma, which is needed for proving Lemma 6.8, shows that it can be computed “globally” by relating the boundaries and adjacency matrices of the subgraphs of  $\Gamma$  in the decomposition to the boundary and adjacency matrices of  $\Gamma$ .

**Lemma 6.7.** *Let  $T$  be an inductive rank decomposition of a graph with dangling edges  $\Gamma = ([G], B)$ . Consider a full subtree  $T'$  of  $T$  that identifies the subgraph  $\Gamma' := \lambda(T') = ([G'], B')$ . Then, the adjacency matrix of  $\Gamma$  can be written as  $[G] = \left[ \begin{pmatrix} G_L & C_L & C \\ 0 & G' & C_R \\ 0 & 0 & G_R \end{pmatrix} \right]$ , its boundary as  $B = \begin{pmatrix} A_L \\ A' \\ A_R \end{pmatrix}$  and we can compute the rank of the boundary of  $\Gamma'$ :  $\text{rk}(B') = \text{rk}(A' \mid C_L^\top \mid C_R)$ .*

*Proof.* Proceed by induction on the decomposition tree  $T$ . If it is just a leaf,  $T = (\Gamma)$ , then  $\Gamma$  has at most one vertex, and  $\Gamma' = \emptyset$  or  $\Gamma' = \Gamma$ . In both cases, the desired equality is true.

If  $T = (T_1 - \Gamma - T_2)$ , then, by Definition 6.4, we can write the adjacency and boundary matrices of  $\Gamma$  in terms of those of  $\Gamma_1 := \lambda(T_1) = ([G_1], B_1)$  and  $\Gamma_2 := \lambda(T_2) = ([G_2], B_2)$ :  $[G] = \left[ \begin{pmatrix} G_1 & C \\ 0 & G_2 \end{pmatrix} \right]$ ,  $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ ,  $B_1 = (A_1 \mid C)$  and  $B_2 = (A_2 \mid C^\top)$ . Suppose that  $T'$  is a full subtree of  $T_1$ . Then, we can write  $[G_1] = \left[ \begin{pmatrix} G_L & C_L & D' \\ 0 & G' & D_R \\ 0 & 0 & H_R \end{pmatrix} \right]$ ,  $A_1 = \begin{pmatrix} A_L \\ A' \\ F_R \end{pmatrix}$  and  $C = \begin{pmatrix} E_L \\ E' \\ E_R \end{pmatrix}$ . It follows that  $B_1 = \begin{pmatrix} A_L & E_L \\ A' & E' \\ F_R & E_R \end{pmatrix}$  and  $C_R = (D_R \mid E')$ . By induction hypothesis,  $\text{rk}(B') = \text{rk}(A' \mid E' \mid C_L^\top \mid D_R)$ . The rank is invariant to permuting the order of columns, thus  $\text{rk}(B') = \text{rk}(A' \mid C_L^\top \mid D_R \mid E') = \text{rk}(A' \mid C_L^\top \mid C_R)$ . We proceed analogously if  $T'$  is a full subtree of  $T_2$ .  $\square$

An inductive rank decomposition defines a rank decomposition by forgetting the labelling of the internal nodes and by forgetting the root node.

**Lemma 6.8.** *Let  $T$  be an inductive rank decomposition of a graph with dangling edges  $\Gamma$ . Then, there is a rank decomposition  $\mathcal{I}^\dagger(T)$  of  $G$  of bounded width:  $\text{wd}(\mathcal{I}^\dagger(T)) \leq \text{wd}(T)$ .*

*Proof.* A binary tree is, in particular, a subcubic tree. Then, we define the rank decomposition corresponding to an inductive rank decomposition  $T$  by its underlying unlabelled tree  $Y$  from which we remove the leaves of  $T$  with empty label. The corresponding bijection  $r: \text{leaves}(Y) \rightarrow \text{vertices}(G)$  between the leaves of  $Y$  and the vertices of  $G$  is defined by the labels of the leaves in  $T$ : if  $l$  is a leaf in  $Y$ , then it is a leaf in  $T$  with a non-empty label: the subgraph  $\Gamma_l$  of  $\Gamma$  with one vertex  $v_l$ . These subgraphs need to give  $\Gamma$  when composed together, then, the function  $r$  is a bijection with  $r(l) := v_l$ . Thus,  $(Y, r)$  is a branch decomposition of  $G$  and we can define  $\mathcal{I}^\dagger(T) := (Y, r)$ .

By construction, the edges of  $Y$  are edges of  $T$  so we can compute the order of the edges in  $Y$  from the labellings of the nodes in  $T$ . Consider an edge  $b$  in  $Y$  and consider its endpoints in  $T$ : let  $\{v, v_b\} = \text{ends}(b)$  with  $v$  parent of  $v_b$  in  $T$ . The order of  $b$  is related to the rank of the boundary of the subtree  $T_b$  of  $T$  with root in  $v_b$ . Let  $\lambda(T_b) = \Gamma_b = ([G_b], B_b)$  be the subgraph of  $\Gamma$  identified by  $T_b$ . We can express the adjacency and boundary matrices of  $\Gamma$  in terms of those of  $\Gamma_b$ :

$$[G] = \left[ \begin{pmatrix} G_L & C_L & C \\ 0 & G_b & C_R \\ 0 & 0 & G_R \end{pmatrix} \right] \quad \text{and} \quad B = \begin{pmatrix} A_L \\ A' \\ A_R \end{pmatrix}.$$

By Lemma 6.7, the boundary rank of  $\Gamma_b$  can be computed by  $\text{rk}(B_b) = \text{rk}(A' \mid C_L^\top \mid C_R)$ . By Definition 2.42, the order of the edge  $b$  is  $\text{ord}(b) := \text{rk}(C_L^\top \mid C_R)$ , and we can bound it with the boundary rank of  $\Gamma_b$ :  $\text{rk}(B_b) \geq$

$\text{ord}(b)$ . These observations allow us to bound the width of the rank decomposition  $Y$ .

$$\begin{aligned}
& \text{wd}(Y, r) \\
& := \max_{b \in \text{edges}(Y)} \text{ord}(b) \\
& \leq \max_{b \in \text{edges}(Y)} \text{rk}(B_b) \\
& \leq \max_{T' \leq T} \text{rk}(\text{sources}(\lambda(T'))) \\
& =: \text{wd}(T)
\end{aligned}$$

□

An inductive rank decomposition is almost the same as a rank decomposition but with a selected node, the root, that points to the first step in the decomposition. We assign a root to a rank decomposition by picking an edge in the decomposition tree and subdividing it. The extra vertex added in this operation becomes the root and determines the labelling of the internal nodes by proceeding bottom up from the leaves.

**Lemma 6.9.** *Let  $\Gamma = ([G], B)$  be a graph with dangling edges and  $(Y, r)$  be a rank decomposition of  $G$ . Then, there is an inductive rank decomposition  $\mathcal{I}(Y, r)$  of  $\Gamma$  of bounded width:  $\text{wd}(\mathcal{I}(Y, r)) \leq \text{wd}(Y, r) + \text{rk}(B)$ .*

*Proof.* Proceed by induction on the number of edges of the decomposition tree  $Y$  to construct an inductive decomposition tree  $T$  in which every non-trivial full subtree  $T'$  has a corresponding edge  $b'$  in the tree  $Y$ .

Suppose  $Y$  has no edges, then either  $G = \emptyset$  or  $G$  has one vertex. In either case, we define an inductive rank decomposition with just a leaf labelled with  $\Gamma$ ,  $\mathcal{I}(Y, r) := (\Gamma)$ . We compute its width by definition:  $\text{wd}(\mathcal{I}(Y, r)) := \text{rk}(B) \leq \text{wd}(Y, r) + \text{rk}(B)$ .

If the decomposition tree has at least an edge, then it is composed of two subcubic subtrees,  $Y = Y_1 \overset{b}{-} Y_2$ . Let  $V_i := r(\text{leaves}(Y_i))$  be the set of vertices associated to  $Y_i$  and  $G_i := G[V_i]$  be the subgraph of  $G$  induced by the set of vertices  $V_i$ . By induction hypothesis, there are inductive rank decompositions  $T_i$  of  $\Gamma_i = ([G_i], B_i)$  in which every full subtree  $T'$  has an associated edge  $b'$ . Associate the edge  $b$  to both  $T_1$  and  $T_2$  so that every subtree of  $T$  has an associated edge in  $Y$ . We can use these decompositions to define an inductive rank decomposition  $T = (T_1 - \Gamma - T_2)$  of  $\Gamma$ . Let  $T'$  be a full subtree of  $T$  corresponding to  $\Gamma' = ([G'], B')$ . By Lemma 6.7, we can compute the rank of its boundary matrix  $\text{rk}(B') = \text{rk}(A' \mid C_L^\top \mid C_R)$ , where  $A'$ ,  $C_L$  and  $C_R$  are as in the statement of Lemma 6.7. The matrix  $A'$  contains some of the rows of  $B$ , then its rank is bounded by the rank of  $B$  and we obtain  $\text{rk}(B') \leq \text{rk}(B) + \text{rk}(C_L^\top \mid C_R)$ . The matrix  $(C_L^\top \mid C_R)$  records the edges between the vertices in  $G'$  and the vertices in the rest of  $G$ , which, by Definition 2.42, are the edges that determine  $\text{ord}(b')$ . This means that the rank of this matrix is the order of the edge  $b'$ :  $\text{rk}(C_L^\top \mid C_R) = \text{ord}(b')$ . With these observations, we can compute the width of  $T$ .

$$\begin{aligned}
& \text{wd}(T) \\
& = \max_{T' \leq T} \text{rk}(B') \\
& = \max_{T' \leq T} \text{rk}(A' \mid C_L^\top \mid C_R) \\
& \leq \max_{T' \leq T} \text{rk}(C_L^\top \mid C_R) + \text{rk}(B) \\
& = \max_{b \in \text{edges}(Y)} \text{ord}(b) + \text{rk}(B) \\
& =: \text{wd}(Y, r) + \text{rk}(B)
\end{aligned}$$

□

By combining Lemmas 6.8 and 6.9 we obtain that rank decompositions and inductive ones give the same complexity measure.

**Proposition 6.10.** *For graphs with no dangling edges, rank width and inductive rank width coincide.*

## 6.2 Bounding rank width

Monoidal width in the prop BGraph of graphs is equivalent to rank width: it is at most twice and at least a half of rank width. Transforming an inductive rank decomposition into a monoidal decomposition gives the upper bound, while a mapping in the other direction yields the lower bound. As for cospans of graphs, the number of vertices in a graph gives its cost, so an appropriate weight function counts the number of vertices in each morphism.

**Definition 6.11.** For a morphism  $g : n \rightarrow m$  in MGraph, the weight function  $w$  is defined as  $w(g) := \text{rk}(G) + \text{rk}(L) + \text{rk}(R) + \text{rk}(P) + \text{rk}(F)$ , where  $g = ([G], L, R, P, [F])$ .

With this definition, the identity on  $n$  weights  $n$  because  $\text{rk}(\mathbb{1}_n) = n$ , and composing along  $n$  wires costs  $n$ . This defines a weight function.

**Lemma 6.12.** *The function  $w$  in Definition 6.11 satisfies the conditions for a weight function in Definition 3.3 in the monoidal category MGraph.*

*Proof.* For morphisms  $g : n \rightarrow m$ ,  $h : m \rightarrow l$  and  $g' : n' \rightarrow m'$ , in MGraph, given by  $g = ([G], L, R, P, [F])$ ,  $h = ([H], M, S, Q, [E])$  and  $g' = ([G'], L', R', P', [F'])$ , we recall the expressions for the composition  $g \circ h$  and the monoidal product  $g \otimes g'$ .

$$\begin{aligned} g \circ h &:= \left( \left[ \begin{pmatrix} G & RM^\top \\ 0 & H+MFM^\top \end{pmatrix} \right], \begin{pmatrix} L \\ MP \end{pmatrix}, \begin{pmatrix} RQ^\top \\ S+M(F+F^\top)Q^\top \end{pmatrix}, QP, [E + QFQ^\top] \right) \\ g \otimes g' &:= \left( [G \oplus G'], L \oplus L', R \oplus R', P \oplus P', [F \oplus F'] \right) \end{aligned}$$

We bound the ranks of these matrices individually.

$$\begin{aligned} \text{rk} \begin{pmatrix} G & RM^\top \\ 0 & H+MFM^\top \end{pmatrix} &\leq \text{rk}(G) + \text{rk}(H) + m & \text{rk}(G \oplus G') &\leq \text{rk}(G) + \text{rk}(G') \\ \text{rk} \begin{pmatrix} L \\ MP \end{pmatrix} &\leq \text{rk}(L) + \text{rk}(M) & \text{rk}(L \oplus L') &\leq \text{rk}(L) + \text{rk}(L') \\ \text{rk} \begin{pmatrix} RQ^\top \\ S+M(F+F^\top)Q^\top \end{pmatrix} &\leq \text{rk}(R) + \text{rk}(S) + \text{rk}(Q) & \text{rk}(R \oplus R') &\leq \text{rk}(R) + \text{rk}(R') \\ \text{rk}(QP) &\leq \text{rk}(P) & \text{rk}(P \oplus P') &\leq \text{rk}(P) + \text{rk}(P') \\ \text{rk}(E + QFQ^\top) &\leq \text{rk}(F) + \text{rk}(E) & \text{rk}(F \oplus F') &\leq \text{rk}(F) + \text{rk}(F') \end{aligned}$$

With these inequalities, we bound the weights of compositions and monoidal products.

$$w(g \circ_m h) \leq w(g) + w(h) + m \qquad w(g \otimes g') \leq w(g) + w(g')$$

□

Given the inductive nature of both kinds of decompositions, the monoidal decomposition corresponding to an inductive rank decomposition is constructed by induction. The inductive step relies on the factorisation of morphisms  $n \rightarrow 0$  as shown in Figure 6.1.

In order to show that such factorisation is always possible, Lemma 6.14 shows that any boundary matrix can be split along the ranks  $r_1$  and  $r_2$ .

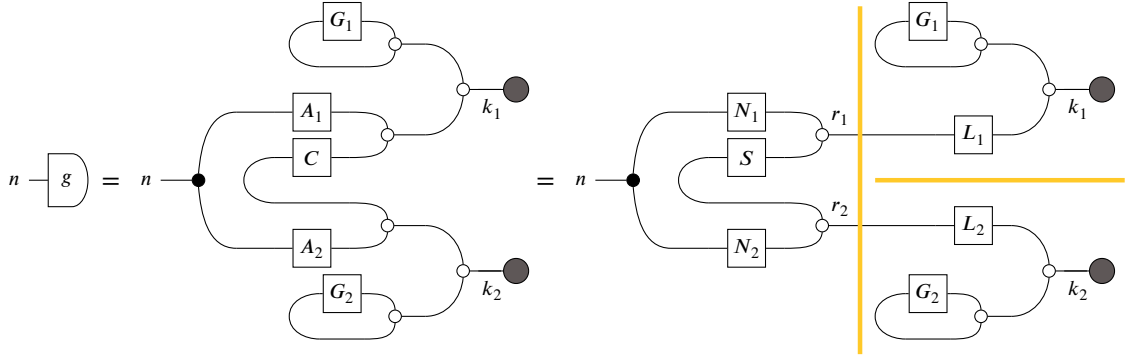


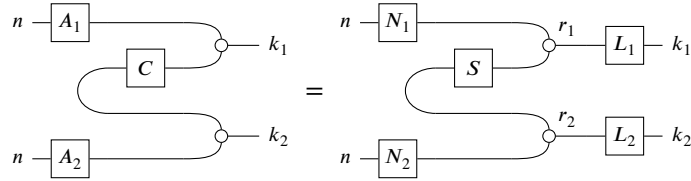
Figure 6.1: Splitting a graph with dangling edges optimally into subgraphs.

*Remark 6.13.* By Lemma 3.16, the rank of a composition of two matrices is bounded by their ranks:  $\text{rk}(A \cdot B) \leq \min\{\text{rk}(A), \text{rk}(B)\}$ . If, moreover,  $B$  has full rank, then  $\text{rk}(A \cdot B) = \text{rk}(A)$ .

**Lemma 6.14.** Let  $A_i \in \text{Mat}_{\mathbb{N}}(k_i, n)$ , for  $i = 1, 2$ , and  $C \in \text{Mat}_{\mathbb{N}}(k_1, k_2)$ . Then, there are rank decompositions of  $(A_1 \mid C)$  and  $(A_2 \mid C^\top)$  of the form

- $(A_1 \mid C) = L_1 \cdot (N_1 \mid S \cdot L_2^\top)$ , and
- $(A_2 \mid C^\top) = L_2 \cdot (N_2 \mid S^\top \cdot L_1^\top)$ .

This ensures that we can decompose the diagram below on the left-hand-side as the one on the right-hand-side, where  $r_1 = \text{rk}(A_1 \mid C)$  and  $r_2 = \text{rk}(A_2 \mid C^\top)$ .



*Proof.* Let  $r_1 = \text{rk}(A_1 \mid C)$  and  $r_2 = \text{rk}(A_2 \mid C^\top)$ . We start by factoring  $(A_1 \mid C)$  into  $L_1 \cdot (N_1 \mid K_1)$ ,

$$\begin{array}{c} n \\ \hline \boxed{A_1} \\ \hline k_2 \\ \hline \boxed{C} \end{array} \rightarrow k_1 = \begin{array}{c} n \\ \hline \boxed{N_1} \\ \hline k_2 \\ \hline \boxed{K_1} \end{array} \rightarrow r_1 \rightarrow \boxed{L_1} \rightarrow k_1$$

where  $L_1 \in \text{Mat}_{\mathbb{N}}(k_1, r_1)$ ,  $N_1 \in \text{Mat}_{\mathbb{N}}(r_1, n)$  and  $K_1 \in \text{Mat}_{\mathbb{N}}(r_1, k_2)$ . Then, we proceed with factoring  $(A_2 \mid K_1^\top)$  and we show that  $\text{rk}(A_2 \mid K_1^\top) = \text{rk}(A_2 \mid C^\top)$ . Let  $L_2 \cdot (N_2 \mid K_2)$  be a rank factorisation of  $(A_2 \mid K_1^\top)$ ,

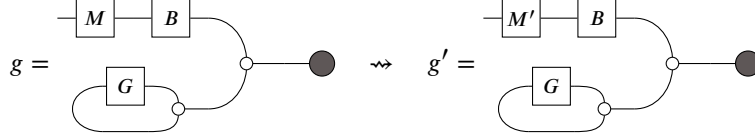
$$\begin{array}{c} r_1 \\ \hline \boxed{K_1^\top} \\ \hline n \\ \hline \boxed{A_2} \end{array} \rightarrow k_2 = \begin{array}{c} r_1 \\ \hline \boxed{K_2} \\ \hline n \\ \hline \boxed{N_2} \end{array} \rightarrow r' \rightarrow \boxed{L_2} \rightarrow k_2$$

with  $L_2 \in \text{Mat}_{\mathbb{N}}(k_2, r')$ ,  $N_2 \in \text{Mat}_{\mathbb{N}}(r', n)$  and  $K_2 \in \text{Mat}_{\mathbb{N}}(r', k_1)$ . We show that  $r' = r_2$ . By the first factorisation, we obtain that  $C = L_1 \cdot K_1$ , and

$$(A_2 \mid C^\top) = (A_2 \mid K_1^\top \cdot L_1^\top) = (A_2 \mid K_1^\top) \cdot \begin{pmatrix} 1 & 0 \\ 0 & L_1^\top \end{pmatrix}.$$

Then,  $r' = r_2$  because  $L_1$  and, consequently,  $\begin{pmatrix} 1 & 0 \\ 0 & L_1^\top \end{pmatrix}$  have full rank and we can apply Remark 6.13. By letting  $S = K_2^\top$ , we obtain the desired factorisation.  $\square$

Once the graph in Figure 6.1 has been split, the boundaries of its induced subgraphs have changed. This means that we cannot apply the inductive hypothesis right away, but we need to first transform the inductive rank decompositions of the old subgraphs into decompositions of the new ones, as shown in Lemma 6.15. More explicitly, when  $M$  has full rank, if there is an inductive rank decomposition of  $\Gamma = ([G], B \cdot M)$ , which corresponds to  $g$  below left, we can obtain one of  $\Gamma' = ([G], B')$ , which corresponds to  $g'$  below right, of the same width.



**Lemma 6.15.** *Let  $T$  be an inductive rank decomposition of  $\Gamma = ([G], B \cdot M)$ , with  $M$  that has full rank. Then, there is an inductive rank decomposition  $T'$  of  $\Gamma' = ([G], B \cdot M')$  such that  $\text{wd}(T) \leq \text{wd}(T')$  and such that  $T$  and  $T'$  have the same underlying tree structure. If, moreover,  $M'$  has full rank, then  $\text{wd}(T) = \text{wd}(T')$ .*

*Proof.* Proceed by induction on the decomposition tree  $T$ . If the tree  $T$  is just a leaf with label  $\Gamma$ , then we define the corresponding tree to be just a leaf with label  $\Gamma'$ :  $T' := (\Gamma')$ . Clearly,  $T$  and  $T'$  have the same underlying tree structure. By Remark 6.13 and the fact that  $M$  has full rank, we can relate their widths:  $\text{wd}(T') := \text{rk}(B \cdot M') \leq \text{rk}(B) = \text{rk}(B \cdot M) := \text{wd}(T)$ . If, moreover,  $M'$  has full rank, the inequality becomes an equality and  $\text{wd}(T') = \text{wd}(T)$ .

If  $T = (T_1 - \Gamma - T_2)$ , then the adjacency and boundary matrices of  $\Gamma$  can be expressed in terms of those of its subgraphs  $\Gamma_i := \lambda_i(T_i) = ([G_i], D_i)$ , by definition of inductive rank decomposition:  $G = \begin{pmatrix} G_1 & C \\ 0 & G_2 \end{pmatrix}$ ,  $B \cdot M = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \cdot M = \begin{pmatrix} A_1 \cdot M \\ A_2 \cdot M \end{pmatrix}$ , with  $D_1 = (A_1 \cdot M \mid C)$  and  $D_2 = (A_2 \cdot M \mid C^\top)$ . The boundary matrices  $D_i$  of the subgraphs  $\Gamma_i$  can also be expressed as a composition with a full-rank matrix:  $D_1 = (A_1 \cdot M \mid C) = (A_1 \mid C) \cdot \begin{pmatrix} M & 0 \\ 0 & \mathbb{1}_{k_2} \end{pmatrix}$  and  $D_2 = (A_2 \cdot M \mid C^\top) = (A_2 \mid C^\top) \cdot \begin{pmatrix} M & 0 \\ 0 & \mathbb{1}_{k_1} \end{pmatrix}$ . The matrices  $\begin{pmatrix} M & 0 \\ 0 & \mathbb{1}_{k_i} \end{pmatrix}$  have full rank because all their blocks do. Let  $B_1 = (A_1 \mid C)$  and  $B_2 = (A_2 \mid C^\top)$ . By induction hypothesis, there are inductive rank decompositions  $T'_1$  and  $T'_2$  of  $\Gamma'_1 = ([G_1], B_1 \cdot \begin{pmatrix} M & 0 \\ 0 & \mathbb{1}_{k_2} \end{pmatrix})$  and  $\Gamma'_2 = ([G_2], B_2 \cdot \begin{pmatrix} M & 0 \\ 0 & \mathbb{1}_{k_1} \end{pmatrix})$  with the same underlying tree structure as  $T_1$  and  $T_2$ , respectively. Moreover, their width is bounded,  $\text{wd}(T'_i) \leq \text{wd}(T_i)$ , and if, additionally,  $M'$  has full rank,  $\text{wd}(T'_i) = \text{wd}(T_i)$ . Then, we can use these decompositions to define an inductive rank decomposition  $T' := (T'_1 - \Gamma' - T'_2)$  of  $\Gamma'$  because its adjacency and boundary matrices can be expressed in terms of those of  $\Gamma'_i$  as in the definition of inductive rank decomposition:  $G = \begin{pmatrix} G_1 & C \\ 0 & G_2 \end{pmatrix}$ ,  $B_1 \cdot \begin{pmatrix} M' & 0 \\ 0 & \mathbb{1}_{k_2} \end{pmatrix} = (A_1 \cdot M' \mid C)$  and  $B_2 \cdot \begin{pmatrix} M' & 0 \\ 0 & \mathbb{1}_{k_1} \end{pmatrix} = (A_2 \cdot M' \mid C^\top)$ . Applying the induction hypothesis and Remark 6.13, we compute the width of this decomposition.

$$\begin{aligned} \text{wd}(T') & \\ & := \max\{\text{rk}(B \cdot M'), \text{wd}(T'_1), \text{wd}(T'_2)\} \\ & \leq \max\{\text{rk}(B), \text{wd}(T_1), \text{wd}(T_2)\} \\ & = \max\{\text{rk}(B \cdot M), \text{wd}(T_1), \text{wd}(T_2)\} \\ & =: \text{wd}(T) \end{aligned}$$

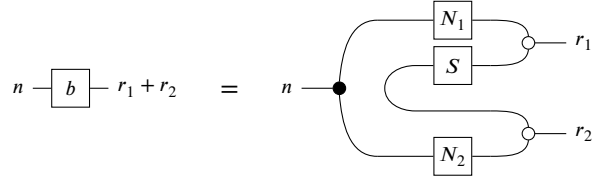
If, moreover,  $M'$  has full rank, the inequality becomes an equality and  $\text{wd}(T') = \text{wd}(T)$ .  $\square$

With the above results, we construct a monoidal decomposition from an inductive rank decomposition and show the upper bound on monoidal width.

**Proposition 6.16.** *Let  $\Gamma = ([G], B)$  be a graph with dangling edges and  $g : n \rightarrow 0$  be the morphism in BGraph corresponding to  $\Gamma$ . Let  $T$  be a inductive rank decomposition of  $\Gamma$ . Then, there is a monoidal decomposition  $\mathcal{R}^\dagger(T)$  of  $g$  of bounded width  $\text{wd}(\mathcal{R}^\dagger(T)) \leq 2 \cdot \text{wd}(T)$ .*

*Proof.* Proceed by induction on the decomposition tree  $T$ . If the decomposition tree consists of just one leaf with label  $\Gamma$ , then  $\Gamma$  must have at most one vertex, we can define  $\mathcal{R}^\dagger(T) := (g)$  to also be just a leaf, and bound its width  $\text{wd}(T) := \text{rk}(G) = \text{wd}(\mathcal{R}^\dagger(T))$ .

If  $T = (T_1 - \Gamma - T_2)$ , then we can relate the adjacency and boundary matrices of  $\Gamma$  to those of  $\Gamma_i := \lambda(T_i) = ([G_i], B_i)$ , by definition of inductive rank decomposition:  $G = \begin{pmatrix} G_1 & C \\ 0 & G_2 \end{pmatrix}$ ,  $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$ ,  $B_1 = (A_1 \mid C)$  and  $B_2 = (A_2 \mid C^\top)$ . By Lemma 6.14, there are rank decompositions of  $(A_1 \mid C)$  and  $(A_2 \mid C^\top)$  of the form:  $(A_1 \mid C) = L_1 \cdot (N_1 \mid S \cdot L_2^\top)$ ; and  $(A_2 \mid C^\top) = L_2 \cdot (N_2 \mid S^\top \cdot L_1^\top)$ . This means that we can write  $g$  as in Figure 6.1, with  $r_i = \text{rk}(B_i)$ . Then,  $B_i = L_i \cdot M_i$  with  $M_i$  that has full rank  $r_i$ . By Lemma 6.15, there is an inductive rank decomposition  $T'_i$  of  $\Gamma'_i = ([G_i], L_i)$ , with the same underlying binary tree as  $T_i$ , such that  $\text{wd}(T_i) = \text{wd}(T'_i)$ . Let  $g_i : r_i \rightarrow 0$  be the morphisms in BGraph corresponding to  $\Gamma'_i$  and let  $b : n \rightarrow r_1 + r_2$  be defined as



By induction hypothesis, there are monoidal decompositions  $\mathcal{R}^\dagger(T'_i)$  of the morphisms  $g_i$  of bounded width:  $\text{wd}(\mathcal{R}^\dagger(T'_i)) \leq 2 \cdot \text{wd}(T'_i) = 2 \cdot \text{wd}(T_i)$ . Then,  $g = b \circ_{r_1+r_2} (g_1 \otimes g_2)$  and  $\mathcal{R}^\dagger(T) := (b - \circ_{r_1+r_2} - (\mathcal{R}^\dagger(T'_1) - \otimes - \mathcal{R}^\dagger(T'_2)))$  is a monoidal decomposition of  $g$ . Its width can be computed.

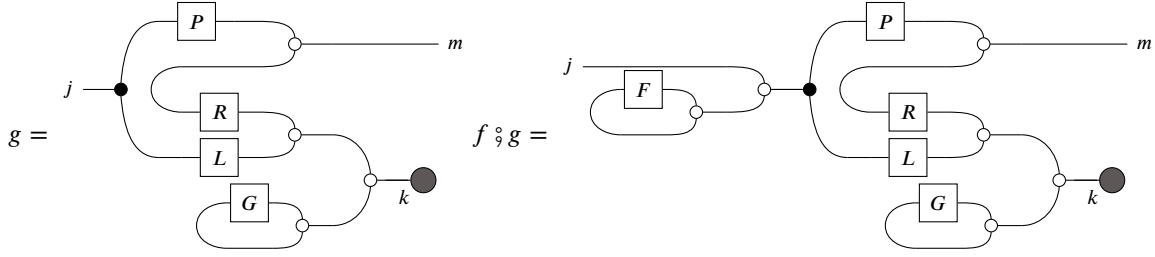
$$\begin{aligned}
 \text{wd}(\mathcal{R}^\dagger(T)) & \\
 &:= \max\{\text{w}(b), \text{w}(r_1 + r_2), \text{wd}(\mathcal{R}^\dagger(T'_1)), \text{wd}(\mathcal{R}^\dagger(T'_2))\} \\
 &\leq \max\{\text{w}(b), \text{w}(r_1 + r_2), 2 \cdot \text{wd}(T'_1), 2 \cdot \text{wd}(T'_2)\} \\
 &= \max\{\text{w}(b), r_1 + r_2, 2 \cdot \text{wd}(T_1), 2 \cdot \text{wd}(T_2)\} \\
 &\leq 2 \cdot \max\{r_1, r_2, \text{wd}(T_1), \text{wd}(T_2)\} \\
 &=: 2 \cdot \text{wd}(T)
 \end{aligned}$$

□

Each node in a monoidal decomposition of a graph  $g$  determines a cut in  $g$ . This correspondence maps monoidal decompositions to inductive rank decompositions. However, bounding their widths requires some care because the splitting determined by a monoidal decomposition may be not the canonical one needed to define an inductive rank decomposition of the same graph. Lemma 6.7 shows that this does not matter as, from the induced inductive rank decompositions, we can construct ones of the correct subgraphs by adding some connections between the vertices as long as the complexity of these connections is bounded by the boundary.

Given an inductive rank decomposition of  $\Gamma = ([G], (L \mid R))$ , associated to  $g$  below left, we construct one of  $\Gamma' := ([G + L \cdot F \cdot L^\top], (L \mid R + L \cdot (F + F^\top) \cdot P^\top))$ , which corresponds to  $f \circ g$  below right, of at

most the same width.



**Lemma 6.17.** Let  $T$  be an inductive rank decomposition of  $\Gamma = ([G], (L \mid R))$ , with  $G \in \text{Mat}_{\mathbb{N}}(k, k)$ ,  $L \in \text{Mat}_{\mathbb{N}}(k, j)$  and  $R \in \text{Mat}_{\mathbb{N}}(k, m)$ . Let  $F \in \text{Mat}_{\mathbb{N}}(j, j)$ ,  $P \in \text{Mat}_{\mathbb{N}}(m, j)$  and define the graph  $\Gamma'$  by precomposing with the adjacency matrix  $[F]$ ,  $\Gamma' := ([G + L \cdot F \cdot L^\top], (L \mid R + L \cdot (F + F^\top) \cdot P^\top))$ . Then, there is an inductive rank decomposition  $T'$  of  $\Gamma'$  such that  $\text{wd}(T') \leq \text{wd}(T)$ .

*Proof.* Note that we can factor the boundary matrix of  $\Gamma'$  as  $(L \mid R + L \cdot (F + F^\top) \cdot P^\top) = (L \mid R) \cdot \begin{pmatrix} 1_j & (F + F^\top) \cdot P^\top \\ 0 & 1_m \end{pmatrix}$ . Then, we can bound its rank,  $\text{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top) \leq \text{rk}(L \mid R)$ .

Proceed by induction on the decomposition tree  $T$ .

If it is just a leaf with label  $\Gamma$ , then  $\Gamma$  has one vertex and we can define a decomposition for  $\Gamma'$  to be also just a leaf:  $T' := (\Gamma')$ . We can bound its width with the width of  $T$ :  $\text{wd}(T') := \text{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top) \leq \text{rk}(L \mid R) =: \text{wd}(T)$ .

If  $T = (T_1 - \Gamma - T_2)$ , then there are two subgraphs  $\Gamma_1 = ([G_1], (L_1 \mid R_1 \mid C))$  and  $\Gamma_2 = ([G_2], (L_2 \mid R_2 \mid C))$  such that  $T_i$  is an inductive rank decomposition of  $\Gamma_i$ , and we can relate the adjacency and boundary matrices of  $\Gamma$  to those of  $\Gamma_1$  and  $\Gamma_2$ , by definition of inductive rank decomposition:  $[G] = \begin{bmatrix} G_1 & C \\ 0 & G_2 \end{bmatrix}$  and  $(L \mid R) = \begin{pmatrix} L_1 & R_1 \\ L_2 & R_2 \end{pmatrix}$ . Similarly, we express the adjacency and boundary matrices of  $\Gamma'$  in terms of the same components:  $[G + L \cdot F \cdot L^\top] = \begin{bmatrix} G_1 + L_1 \cdot F \cdot L_1^\top & C + L_1 \cdot (F + F^\top) \cdot L_2^\top \\ 0 & G_2 + L_2 \cdot F \cdot L_2^\top \end{bmatrix}$  and  $(L \mid R + L \cdot (F + F^\top) \cdot P^\top) = \begin{pmatrix} L_1 & R_1 + L_1 \cdot (F + F^\top) \cdot P^\top \\ L_2 & R_2 + L_2 \cdot (F + F^\top) \cdot P^\top \end{pmatrix}$ . We use these decompositions to define two subgraphs of  $\Gamma'$  and apply the induction hypothesis to them.

$$\begin{aligned} \Gamma'_1 &:= ([G_1 + L_1 \cdot F \cdot L_1^\top], (L_1 \mid R_1 + L_1 \cdot (F + F^\top) \cdot P^\top \mid C + L_1 \cdot (F + F^\top) \cdot L_2^\top)) \\ &= ([G_1 + L_1 \cdot F \cdot L_1^\top], (L_1 \mid (R_1 \mid C) + L_1 \cdot (F + F^\top) \cdot (P^\top \mid L_2^\top))) \end{aligned}$$

and

$$\begin{aligned} \Gamma'_2 &:= ([G_2 + L_2 \cdot F \cdot L_2^\top], (L_2 \mid R_2 + L_2 \cdot (F + F^\top) \cdot P^\top \mid C^\top + L_2 \cdot (F + F^\top) \cdot L_1^\top)) \\ &= ([G_2 + L_2 \cdot F \cdot L_2^\top], (L_2 \mid (R_2 \mid C^\top) + L_2 \cdot (F + F^\top) \cdot (P^\top \mid L_1^\top))) \end{aligned}$$

By induction, we have inductive rank decompositions  $T'_i$  of  $\Gamma'_i$  such that  $\text{wd}(T'_i) \leq \text{wd}(T_i)$ . We defined  $\Gamma'_i$  so that  $T' := (T'_1 - \Gamma' - T'_2)$  would be an inductive rank decomposition of  $\Gamma'$ . We can bound its width as desired.

$$\begin{aligned} &\text{wd}(T') \\ &:= \max\{\text{wd}(T'_1), \text{wd}(T'_2), \text{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top)\} \\ &\leq \max\{\text{wd}(T_1), \text{wd}(T_2), \text{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top)\} \\ &\leq \max\{\text{wd}(T_1), \text{wd}(T_2), \text{rk}(L \mid R)\} \\ &=: \text{wd}(T) \end{aligned}$$

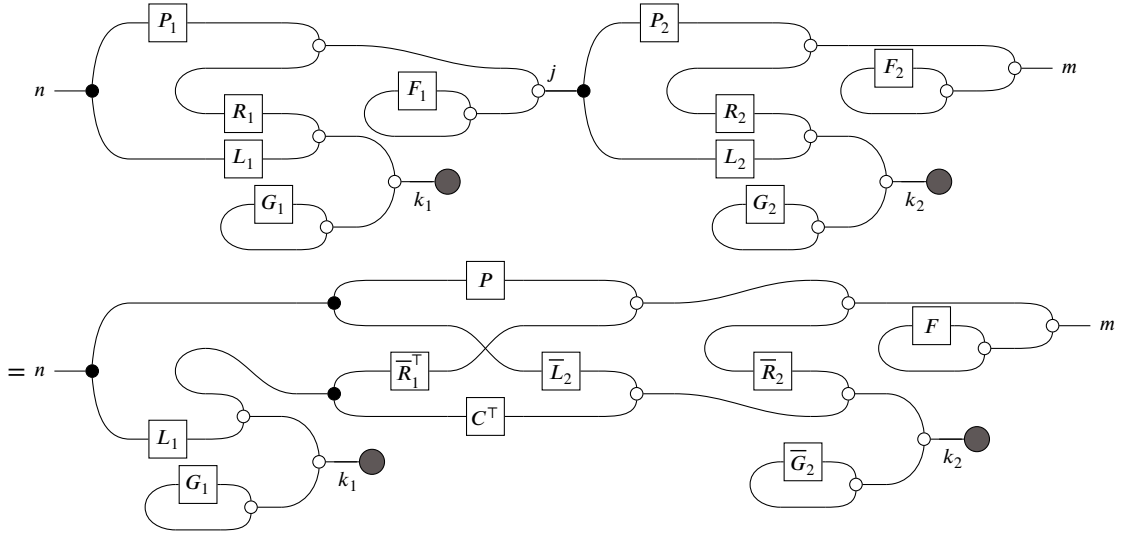
□

A monoidal decomposition defines by induction an inductive rank decomposition. The inductive step relies on Lemma 6.15 and Lemma 6.7 to obtain, from an inductive rank decomposition of a graph  $g$ , one of a graph constructed from  $g$  by adding additional connections to the boundary or between the vertices in a controlled manner.

**Proposition 6.18.** *Let  $d \in D_g$  be a monoidal decomposition of a morphism  $g : n \rightarrow m$  in BGraph given by  $g = ([G], L, R, P, [F])$ , and let  $\Gamma = ([G], (L | R))$  be its corresponding graph with dangling edges. Then, there exist an inductive rank decomposition  $\mathcal{R}(d)$  of  $\Gamma$  of bounded width:  $\text{wd}(\mathcal{R}(d)) \leq 2 \cdot \max\{\text{wd}(d), \text{rk}(L), \text{rk}(R)\}$ .*

*Proof.* Proceed by induction on the decomposition tree  $d$ . If it is just a leaf with label  $g$ , then its width is defined to be the number  $k$  of vertices of  $g$ ,  $\text{wd}(d) := k$ . Pick any inductive rank decomposition of  $\Gamma$  and define  $\mathcal{R}(d) := T$ . Surely,  $\text{wd}(T) \leq k =: \text{wd}(d)$

If  $d = (d_1 - \circ_j - d_2)$  starts with a composition node, then  $g$  is the composition of two morphisms:  $g = g_1 \circ g_2$ , with  $g_i = ([G_i], L_i, R_i, P_i, [F_i])$ . Given the partition of the vertices determined by  $g_1$  and  $g_2$ , we can decompose  $g$  in another way, by writing  $[G] = \begin{bmatrix} \bar{G}_1 & C \\ 0 & \bar{G}_2 \end{bmatrix}$  and  $B = (L | R) = \begin{pmatrix} \bar{L}_1 & \bar{R}_1 \\ \bar{L}_2 & \bar{R}_2 \end{pmatrix}$ . Then, we have that  $\bar{G}_1 = G_1$ ,  $\bar{L}_1 = L_1$ ,  $P = P_2 \cdot P_1$ ,  $C = R_1 \cdot L_2^\top$ ,  $\bar{R}_1 = R_1 \cdot P_2^\top$ ,  $\bar{L}_2 = L_2 \cdot P_1$ ,  $\bar{R}_2 = R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top$ ,  $\bar{G}_2 = G_2 + L_2 \cdot F_1 \cdot L_2^\top$ , and  $F = F_2 + P_2 \cdot F_1 \cdot P_2^\top$ . This corresponds to the following diagrammatic rewriting using the equations of BGraph.



We define  $\bar{B}_1 := (\bar{L}_1 | \bar{R}_1 | C)$  and  $\bar{B}_2 := (\bar{L}_2 | \bar{R}_2 | C^\top)$ . In order to build an inductive rank decomposition of  $\Gamma$ , we need rank decompositions of  $\bar{\Gamma}_i = ([\bar{G}_i], \bar{B}_i)$ . We obtain these in three steps. Firstly, we apply induction to obtain inductive rank decompositions  $\mathcal{R}(d_i)$  of  $\Gamma_i = ([G_i], (L_i | R_i))$  such that  $\text{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\text{wd}(d_i), \text{rk}(L_i), \text{rk}(R_i)\}$ . Secondly, we apply Lemma 6.17 to obtain an inductive rank decomposition  $T'_2$  of  $\Gamma'_2 = ([\bar{G}_2 + L_2 \cdot F_1 \cdot L_2^\top], (L_2 | R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top))$  such that  $\text{wd}(T'_2) \leq \text{wd}(\mathcal{R}(d_2))$ . Lastly, we observe that  $(\bar{R}_1 | C) = R_1 \cdot (P_2^\top | L_2^\top)$  and  $(\bar{L}_2 | C^\top) = L_2 \cdot (P_1 | R_1^\top)$ . Then we obtain that  $\bar{B}_1 = (L_1 | R_1) \cdot \begin{pmatrix} 1_n & 0 & 0 \\ 0 & P_2^\top & L_2^\top \end{pmatrix}$  and  $\bar{B}_2 = (L_2 | R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top) \cdot \begin{pmatrix} P_1 & 0 & R_1^\top \\ 0 & 1_m & 0 \end{pmatrix}$ , and we can apply Lemma 6.15 to get inductive rank decompositions  $T_i$  of  $\bar{\Gamma}_i$  such that  $\text{wd}(T_1) \leq \text{wd}(\mathcal{R}(d_1))$  and  $\text{wd}(T_2) \leq \text{wd}(T'_2) \leq \text{wd}(\mathcal{R}(d_2))$ .



If  $k_1, k_2 > 0$ , then we define  $\mathcal{R}(d) := (T_1 - \Gamma - T_2)$ , which is an inductive rank decomposition of  $\Gamma$  because  $\bar{\Gamma}_i$  satisfy the conditions in Definition 6.4. If  $k_1 = 0$ , then  $\Gamma = \bar{\Gamma}_2$  and we can define  $\mathcal{R}(d) := T_2$ . Similarly, if  $k_2 = 0$ , then  $\Gamma = \bar{\Gamma}_1$  and we can define  $\mathcal{R}(d) := T_1$ . In any case, we can compute the width of  $\mathcal{R}(d)$  (if  $k_i = 0$  then  $T_i = ()$  and  $\text{wd}(T_i) = 0$ ) using the inductive hypothesis, Lemma 6.17, Lemma 6.15, the fact that  $\text{rk}(L) \geq \text{rk}(L_1)$ ,  $\text{rk}(R) \geq \text{rk}(R_2)$  and  $j \geq \text{rk}(R_1)$ ,  $\text{rk}(L_2)$  because  $R_1 : j \rightarrow k_1$  and  $L_2 : j \rightarrow k_2$ .

$$\begin{aligned}
& \text{wd}(T) \\
& := \max\{\text{wd}(T_1), \text{wd}(T_2), \text{rk}(L \mid R)\} \\
& \leq \max\{\text{wd}(\mathcal{R}(d_1)), \text{wd}(T_2'), \text{rk}(L \mid R)\} \\
& \leq \max\{\text{wd}(\mathcal{R}(d_1)), \text{wd}(\mathcal{R}(d_2)), \text{rk}(L \mid R)\} \\
& \leq \max\{\text{wd}(\mathcal{R}(d_1)), \text{wd}(\mathcal{R}(d_2)), \text{rk}(L) + \text{rk}(R)\} \\
& \leq \max\{2 \cdot \text{wd}(d_1), 2 \cdot \text{rk}(L_1), 2 \cdot \text{rk}(R_1), 2 \cdot \text{wd}(d_2), 2 \cdot \text{rk}(L_2), 2 \cdot \text{rk}(R_2), \text{rk}(L) + \text{rk}(R)\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), \text{rk}(L_1), \text{rk}(R_1), \text{wd}(d_2), \text{rk}(L_2), \text{rk}(R_2), \text{rk}(L), \text{rk}(R)\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), \text{wd}(d_2), j, \text{rk}(L), \text{rk}(R)\} \\
& =: 2 \cdot \max\{\text{wd}(d), \text{rk}(L), \text{rk}(R)\}
\end{aligned}$$

If  $d = (d_1 - \otimes - d_2)$  starts with monoidal product node, then  $g$  is the monoidal product of two morphisms:  $g = g_1 \otimes g_2$ , with  $g_i = ([G_i], L_i, R_i, P_i, [F_i]) : n_i \rightarrow m_i$ . By explicitly computing the monoidal product, we obtain that  $[G] = \begin{bmatrix} G_1 & 0 \\ 0 & G_2 \end{bmatrix}$ ,  $L = \begin{pmatrix} L_1 & 0 \\ 0 & L_2 \end{pmatrix}$ ,  $R = \begin{pmatrix} R_1 & 0 \\ 0 & R_2 \end{pmatrix}$ ,  $P = \begin{pmatrix} P_1 & 0 \\ 0 & P_2 \end{pmatrix}$  and  $F = \begin{pmatrix} F_1 & 0 \\ 0 & F_2 \end{pmatrix}$ . By induction, we have inductive rank decompositions  $\mathcal{R}(d_i)$  of  $\Gamma_i := ([G_i], B_i)$ , where  $B_i = (L_i \mid R_i)$ , of bounded width:  $\text{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\text{wd}(d_i), \text{rk}(L_i), \text{rk}(R_i)\}$ . Let  $\bar{B}_1 := (L_1 \mid \mathbb{0}_{n_2} \mid R_1 \mid \mathbb{0}_{m_2} \mid \mathbb{0}_{k_2}) = B_1 \cdot \begin{pmatrix} \mathbb{1}_{n_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbb{1}_{m_1} & 0 & 0 \end{pmatrix}$  and  $\bar{B}_2 := (\mathbb{0}_{n_1} \mid L_2 \mid \mathbb{0}_{m_1} \mid R_2 \mid \mathbb{0}_{k_1}) = B_2 \cdot \begin{pmatrix} 0 & \mathbb{1}_{n_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbb{1}_{m_2} & 0 \end{pmatrix}$ . By Lemma 6.15, we can obtain inductive rank decompositions  $T_i$  of  $\bar{\Gamma}_i := ([G_i], \bar{B}_i)$  such that  $\text{wd}(T_i) \leq \text{wd}(\mathcal{R}(d_i))$ . If  $k_1, k_2 > 0$ , then we define  $\mathcal{R}(d) := (T_1 - \Gamma - T_2)$ , which is an inductive rank decomposition of  $\Gamma$  because  $\bar{\Gamma}_i$  satisfy the conditions in Definition 6.4. If  $k_1 = 0$ , then  $\Gamma = \bar{\Gamma}_2$  and we can define  $\mathcal{R}(d) := T_2$ . Similarly, if  $k_2 = 0$ , then  $\Gamma = \bar{\Gamma}_1$  and we can define  $\mathcal{R}(d) := T_1$ . In any case, we can compute the width of  $\mathcal{R}(d)$  (if  $k_i = 0$  then  $T_i = ()$  and  $\text{wd}(T_i) = 0$ ) using the inductive hypothesis and Lemma 6.15.

$$\begin{aligned}
& \text{wd}(T) \\
& := \max\{\text{wd}(T_1), \text{wd}(T_2), \text{rk}(L \mid R)\} \\
& \leq \max\{\text{wd}(\mathcal{R}(d_1)), \text{wd}(\mathcal{R}(d_2)), \text{rk}(L \mid R)\} \\
& \leq \max\{\text{wd}(\mathcal{R}(d_1)), \text{wd}(\mathcal{R}(d_2)), \text{rk}(L) + \text{rk}(R)\} \\
& \leq \max\{2 \cdot \text{wd}(d_1), 2 \cdot \text{rk}(L_1), 2 \cdot \text{rk}(R_1), 2 \cdot \text{wd}(d_2), 2 \cdot \text{rk}(L_2), 2 \cdot \text{rk}(R_2), \text{rk}(L) + \text{rk}(R)\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), \text{rk}(L_1), \text{rk}(R_1), \text{wd}(d_2), \text{rk}(L_2), \text{rk}(R_2), \text{rk}(L), \text{rk}(R)\} \\
& \leq 2 \cdot \max\{\text{wd}(d_1), \text{wd}(d_2), \text{rk}(L), \text{rk}(R)\} \\
& =: 2 \cdot \max\{\text{wd}(d), \text{rk}(L), \text{rk}(R)\}
\end{aligned}$$

□

Propositions 6.10, 6.16 and 6.18 combine to the equivalence of monoidal width and rank width.

**Theorem 6.19.** *Let  $G$  be a graph and let  $g = ([G], i, i, (), [()])$  be the corresponding morphism in BGraph. Then,  $\frac{1}{2} \cdot \text{rwd}(G) \leq \text{mwd}(g) \leq 2 \cdot \text{rwd}(G)$ .*

With this result and Theorem 2.39, we obtain equivalence with clique width.

**Corollary 6.20.** *Clique width is equivalent to monoidal width in BGraph.*

## Chapter 7

# A Monoidal Courcelle-Makowsky Theorem

This chapter unifies the results of previous chapters to obtain a general strategy for proving fixed-parameter tractability for problems on monoidal categories. We aim to bring the technique exposed in Section 2.3 for checking formulae on relational structures to the categorical setting. As outlined in Section 2.3, the fixed-parameter tractability result for relational structures relies on the two fundamental steps below.

1. Identifying generators and operations to express relational structures and graphs. The operations have a cost that determines the *width* of structures.
2. Showing a preservation theorem. In fact, the preservation theorems recalled in Section 2.3 are composed of a structural and a computational part.
  - (a) Showing that partial solutions can be combined into solutions for compound structures.
  - (b) Showing that combining partial solutions takes time that is constant in the size of the compound structure but depends on its width.

Classical examples of this procedure are Courcelle's theorems for tree width [Cou90] and clique width [CO00], which we recalled in Sections 2.2 and 2.3.

Chapter 3 gives the first step of this procedure for monoidal categories. Depending on the choice of operations and their cost, algebraic decompositions of relational structures give their algebraic width. In the same way, depending on the choice of monoidal category and its weight function, monoidal decompositions of morphisms give their monoidal width. Once the monoidal category is fixed, the categorical structure gives a canonical choice for the operations: compositions indexed by the objects and monoidal product. Chapter 4 identifies the appropriate categories of relational structures and graphs to derive the operations for tree and clique widths.

This chapter identifies the assumptions that correspond to preservation theorems for showing fixed-parameter tractability for problems on monoidal categories. We exemplify this technique for computing colimits compositionally.

### 7.1 Fixed-parameter tractability in monoidal categories

This section shows that compositional algorithms can solve functorial problems efficiently on inputs of bounded monoidal width (Theorem 7.6). As for the analogous result for checking formulae on relational structures (Theorem 2.52), this result is a relatively straightforward consequence of its assumptions. In fact, the difficult part of showing fixed-parameter tractability lies in showing a preservation theorem, which Theorem 2.52 assumes as hypothesis, and we make a similar assumption for Theorem 7.6. Nonetheless, this result is still informative as it provides a general strategy for proving fixed-parameter tractability of problems on monoidal categories.

The class of problems covered by this result is wider than computing the theory of relational structures. The possible inputs are the morphisms of a fixed monoidal category  $\mathbf{C}$  and, for a morphism  $f : A \rightarrow B$ , we seek to compute  $\mathbf{S}(f)$ . We always assume that the input morphism  $f$  is provided with a monoidal decomposition  $d \in D_f$ . A divide-and-conquer algorithm requires that the mapping  $\mathbf{S}$  from inputs to solutions respects the structure of the monoidal category  $\mathbf{C}$ , i.e. it is a monoidal functor. For the divide-and-conquer algorithm to be efficient, combining solutions must also respect the categorical structure. These assumptions recast the strategy outlined in Section 2.3, and recalled above, in the categorical setting. The two steps below expand on this strategy to prove fixed-parameter tractability for problems on monoidal categories.

1. Find a monoidal category whose morphisms are the inputs to the problem we seek to solve. Although we do not assume that the set of generators is finite, both the examples of structures with sources and graphs with boundaries are finitely presented props.
2. Show that the problem  $\mathbf{S}$  is both structurally and computationally compositional.
  - (a) Show that the mapping  $\mathbf{S}$  from inputs to solutions defines a strong monoidal functor  $\mathbf{S} : \mathbf{C} \rightarrow \mathbf{D}$ , for some monoidal category  $\mathbf{D}$ .
  - (b) Show that combining solutions  $\mathbf{S}(f_1)$  and  $\mathbf{S}(f_2)$  with the operations of the monoidal category  $\mathbf{D}$  depends linearly on the sizes of  $f_1$  and  $f_2$ , but may depend arbitrarily on the cost of the operation used to combine them.

With these assumptions, there is a divide-and-conquer algorithm similar to Algorithm 1 in Section 2.3 that computes solutions compositionally. It runs through the monoidal decomposition given as input starting from the leaves and proceeding bottom-up: it computes the solutions on the leaves by brute-force and combines them according to the operations that appear in the decomposition. Assumption 2b ensures that the running time of this algorithm is linear in the size of the monoidal decomposition given as input, but arbitrarily large on its monoidal width.

**Definition 7.1.** A problem on morphisms of a monoidal category  $\mathbf{C}$  is *functorial* if the mapping from morphisms to solutions is a monoidal functor  $\mathbf{S} : \mathbf{C} \rightarrow \mathbf{D}$ , for some monoidal category  $\mathbf{D}$ .

The structural part of the preservation theorems recalled in Section 2.3 ensures that the mapping from structures and graphs to their theories is functorial.

**Lemma 7.2.** Let  $\sim_{A,B}$  be a class of equivalence relations on the sets  $\mathbf{C}(A, B)$  of morphisms of a monoidal category  $\mathbf{C}$  that respects the categorical structure: if  $f \sim_{A,B} f'$  and  $g \sim_{B,C} g'$ , then  $f \circ g \sim_{A,C} f' \circ g'$ ; and, if  $f \sim_{A,B} f'$  and  $g \sim_{C,D} g'$ , then  $f \otimes g \sim_{A \otimes C, B \otimes D} f' \otimes g'$ . Then, quotienting the sets of morphisms of  $\mathbf{C}$  by these equivalence relations gives a monoidal category  $\mathbf{C}/\sim$  and a functor  $\mathbf{Q} : \mathbf{C} \rightarrow \mathbf{C}/\sim$ .

*Proof.* This is a standard result. See, for example [Mac78, Section II.8]. □

**Example 7.3.** Recall from Section 4.1 that relational structures with  $n$  constants can be seen as morphisms  $n \rightarrow 0$  in the category of cospans of relational structures  $\mathbf{sStruct}_\tau$ . This monoidal category is equivalent to the finitely presented prop  $\mathbf{sFrob}_\tau$ . In Section 4.3, morphisms  $n \rightarrow 0$  in the category  $\mathbf{MGraph}$  are interpreted as graphs with  $n$  labels. The monoidal category  $\mathbf{MGraph}$  is equivalent to the finitely presented prop  $\mathbf{BGraph}$ .

With these interpretations for morphisms in  $\mathbf{sFrob}_\tau$  and  $\mathbf{BGraph}$  in mind, we define logical equivalence for morphisms in these two categories. Two morphisms  $g = c : m \rightarrow G \leftarrow n : d$  and  $g' = c' : m \rightarrow G' \leftarrow n : d'$  in  $\mathbf{Cospan}(\mathbf{UHGraph})_*$  are MSO logically equivalent when the corresponding structures with  $m+n$  constants,  $(G, [c, d])$  and  $(G', [c', d'])$ , are MSO logically equivalent. Similarly, two morphisms  $([G], L, R, P, [S]) : m \rightarrow n$  and  $([G'], L', R', P, [S]) : m \rightarrow n$  in  $\mathbf{MGraph}$  are MSO logically equivalent when their corresponding  $m+n$ -labelled graphs,  $(G, (L \mid R))$  and  $(G', (L' \mid R'))$ , are MSO logically equivalent.

We can now apply the preservation theorems recalled in Section 2.3 to obtain that the operations in the monoidal categories  $\mathbf{sFrob}_\tau$  and  $\mathbf{BGraph}$  preserve logical equivalence. By the Feferman-Vaught-Mostowski (Theorem 2.59) and the Courcelle-Kanté (Theorem 2.60) preservation theorems, MSO logical equivalence

respects compositions and monoidal product in the monoidal categories  $\text{sFrob}_\tau$  and  $\text{BGraph}$ . More in detail, preservation by the disjoint union of relational structures corresponds to preservation by the monoidal product in  $\text{sFrob}_\tau$ , while preservation by the disjoint union and fuse operations together gives preservation by compositions in  $\text{sFrob}_\tau$ . Similarly, preservation by disjoint union of labelled graphs gives preservation by monoidal product in  $\text{BGraph}$ , while preservation by disjoint union, edge creation and bilinear product gives preservation by compositions in  $\text{BGraph}$ .

These considerations show that logical equivalence respects the structure of both monoidal categories  $\text{sFrob}_\tau$  and  $\text{BGraph}$ , and we can apply Lemma 7.2 to obtain that MSO logical equivalence defines quotient categories  $\text{sFrob}_\tau / \equiv_{\text{MSO}}$  and  $\text{BGraph} / \equiv_{\text{MSO}}$ , and monoidal functors  $\mathbf{T} : \text{sFrob}_\tau \rightarrow \text{sFrob}_\tau / \equiv_{\text{MSO}}$  and  $\mathbf{R} : \text{BGraph} \rightarrow \text{BGraph} / \equiv_{\text{MSO}}$ .

As mentioned above, functorial problems can be solved by divide-and-conquer algorithms that go through the monoidal decomposition given as input, starting from the leaves. For a problem to be functorial it is

---

**Algorithm 2:** MonoidalSolve
 

---

**Data:** a monoidal decomposition  $d$  for a morphism  $f$

**Result:** the value of  $\mathbf{S}(f)$

**if**  $d = (G)$  **then**

    compute  $s := \mathbf{S}(f)$  by brute force

**else if**  $d = (d_1 \overset{\circ}{\circlearrowleft} d_2)$  **then**

    compute  $s_1 := \text{MonoidalSolve}(d_1)$

    compute  $s_2 := \text{MonoidalSolve}(d_2)$

    compute  $s := s_1 \overset{\circ}{\circlearrowleft} s_2$

**else if**  $d = (d_1 \otimes d_2)$  **then**

    compute  $s_1 := \text{MonoidalSolve}(d_1)$

    compute  $s_2 := \text{MonoidalSolve}(d_2)$

    compute  $s := s_1 \otimes s_2$

**return**  $s$

---

not necessary that the generators of  $C$  are finite. However, in the case of computing theories of relational structures, finiteness is a necessary assumption. Algorithm 1 relies on precomputing all the solutions on the generators and a table to combine them. This is possible if the generators and the reduction sets of the formulae are finite. We use a slightly different strategy: Algorithm 2 computes the solutions on the generators as needed.

The preservation theorems, Theorems 2.59 and 2.60, in Section 2.3 have a second computational part. They show that the theories of structures and graphs can be composed in time that is constant in the size of the input. The dependency on the cost of the operation can be arbitrarily large because, in the class of inputs of bounded monoidal width, this cost is also bounded.

**Definition 7.4.** An algorithm that computes the solution  $\mathbf{S}(f)$  of a functorial problem on a monoidal category  $C$  with weight function  $w : \mathcal{A} \rightarrow \mathbb{N}$  is *compositional* if there is some function  $c : \mathbb{N} \rightarrow \mathbb{N}$  such that:

1. computing  $\mathbf{S}(f)$  takes  $\mathcal{O}(c(w(f)))$ ;
2. for  $f : A \rightarrow C$  and  $g : C \rightarrow B$  in  $C$ , and given  $s = \mathbf{S}(f)$  and  $t = \mathbf{S}(g)$ , computing the composition  $s \overset{\circ}{\circlearrowleft} t$  along the object  $\mathbf{S}(C)$  in  $D$  takes  $\mathcal{O}(c(w(C)))$ ;
3. for  $f$  and  $g$  in  $C$ , and given  $s = \mathbf{S}(f)$  and  $t = \mathbf{S}(g)$ , computing the monoidal product  $s \otimes t$  in  $D$  takes  $\mathcal{O}(c(0))$ .

For the problem of checking formulae on structures and graphs, having effectively smooth operations implies having a compositional algorithm.

*Example 7.5.* Computing the logical equivalence classes of graphs and relational structures is equivalent to computing their theories. When the operations are effectively smooth, the theories can be combined efficiently with a look-up table (Definition 2.51). The look-up table is precomputed in finite time that is constant in the size of the input, and its size also does not depend on the input, so it can be accessed in constant time. Constant time is less than linear in the input size and the conditions in Definition 7.4 are satisfied.

Denote with  $C_k(A, B)$  the set of morphisms  $A \rightarrow B$  in  $C$  of monoidal width at most  $k$  together with a witness decomposition  $d \in D_f$  of width at most  $k$ .

$$C_k(A, B) := \{(f, d) : f \in C(A, B) \text{ and } d \in D_f \text{ and } \text{wd}(d) \leq k\}$$

On this set, when Algorithm 2 is compositional and the input is provided with a monoidal decomposition, the algorithm runs in time that is linear in the size of the input.

**Theorem 7.6.** *Computing a functorial problem  $S$  on  $C_k(A, B)$  with a compositional algorithm is linear in  $\text{size}(d)$ . Explicitly, given an optimal monoidal decomposition of  $f$ , computing  $S(f)$  takes  $\mathcal{O}(c(k) \cdot \text{size}(d))$ , for some  $c : \mathbb{N} \rightarrow \mathbb{N}$ .*

*Proof.* Let  $d \in D_f$  be a monoidal decomposition of a morphism  $f : A \rightarrow B$  with  $\text{wd}(d) \leq k$ . We show by induction on  $d$  that running Algorithm 2 takes  $\mathcal{O}(c(k) \cdot \text{size}(d))$ .

Suppose that the decomposition is a leaf,  $d = (f)$ . Then, the weight of  $f$  is bounded by  $k$ , and the size of the decomposition is 1. By hypothesis,  $w(f) := \text{wd}(d) \leq k$ , and computing  $S(f)$  takes  $\mathcal{O}(c(w(f))) = \mathcal{O}(c(k) \cdot 1)$  by Assumption 1.

Suppose that the first node is a composition,  $d = (d_1 - \circ_C - d_2)$ . Then, the widths of  $d_1$  and  $d_2$ , and the weight of  $C$  are bounded by  $k$  because the width of  $d$  is:  $\text{wd}(d) := \max\{\text{wd}(d_1), w(C), \text{wd}(d_2)\} \leq k$  by hypothesis. We apply Assumption 2 and the induction hypothesis to bound the time complexity of computing  $S(f)$  as the composition  $S(f_1) \circ_C S(f_2)$  in  $D$ .

$$\begin{aligned} & \mathcal{O}(c(w(C))) + \mathcal{O}(c(k) \cdot \text{size}(d_1)) + \mathcal{O}(c(k) \cdot \text{size}(d_2)) \\ &= \mathcal{O}(c(k) \cdot (\text{size}(d_1) + 1 + \text{size}(d_2))) \\ &= \mathcal{O}(c(k) \cdot \text{size}(d)) \end{aligned}$$

Suppose that the first node is a monoidal product,  $d = (d_1 - \otimes - d_2)$ . Then, the widths of  $d_1$  and  $d_2$  are bounded by  $k$  because the width of  $d$  is:  $\text{wd}(d) := \max\{\text{wd}(d_1), \text{wd}(d_2)\} \leq k$  by hypothesis. We apply Assumption 3 and the induction hypothesis to calculate the time complexity of computing  $S(F)$  as the monoidal product  $S(f_1) \otimes S(f_2)$ .

$$\begin{aligned} & \mathcal{O}(c(0)) + \mathcal{O}(c(k) \cdot \text{size}(d_1)) + \mathcal{O}(c(k) \cdot \text{size}(d_2)) \\ &= \mathcal{O}(c(k) \cdot (\text{size}(d_1) + 1 + \text{size}(d_2))) \\ &= \mathcal{O}(c(k) \cdot \text{size}(d)) \end{aligned}$$

□

## 7.2 Computing colimits compositionally

This section considers the problem of computing finite colimits in a category  $E$  that admits them. This is a functorial problem [RSW08] and we show that it satisfies the assumptions of Theorem 7.6. Diagrams, seen as graph morphisms to the graph underlying  $E$ , are the objects of a category  $\text{Diag}(E)$  with colimits. There is a functor that takes a diagram as inputs and returns an object of  $E$ , its colimit. However, to make this problem compositional, we need to lift this functor to discrete cospans.

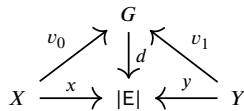
The graph  $|E|$  underlying the category  $E$  is an object of the category  $\text{Graph}_\infty$  of possibly infinite graphs and their homomorphisms. We consider the slice category  $\text{Graph}_\infty/|E|$ , where objects are diagrams  $d : G \rightarrow |E|$  and morphisms are commutative triangles. We restrict to finite diagrams, diagrams  $d : G \rightarrow |E|$  where the graph  $G$  is finite.

**Definition 7.7.** The category  $\text{Diag}(E)$  of diagrams in  $E$  is the full subcategory of  $\text{Graph}_\infty/|E|$  on finite diagrams.

There is a functor  $\text{colim} : \text{Diag}(E) \rightarrow E$  that assigns to each diagram  $d$  an object in  $E$  that is its colimit<sup>1</sup>. This functor is unique up to isomorphism. In order to decompose diagrams, we consider discrete cospans of them. A diagram  $d : G \rightarrow |E|$  is discrete if the graph  $G$  is discrete.

**Definition 7.8.** The category  $\text{CDiag}(E)$  is the full subcategory of  $\text{Cospan}(\text{Diag}(E))$  on discrete cospans of diagrams in  $E$ .

Explicitly, objects are graph morphisms  $X \rightarrow |E|$ , i.e. functions  $X \rightarrow \text{Obj}(E)$  and morphisms are commutative diagrams of graph homomorphisms.



Composition is given by pushout and monoidal product by the coproduct.

**Proposition 7.9** ([RSW05; RSW08]). *The category  $\text{CDiag}(E)$  is equivalent to free strict symmetric monoidal category on the monoidal signature composed of the generators of a Frobenius monoid (Figure 4.1) for every vertex of  $|E|$  and all the edges of  $|E|$ , quotiented by the axioms of Frobenius monoids. These generators and equations are in Figure 7.1.*

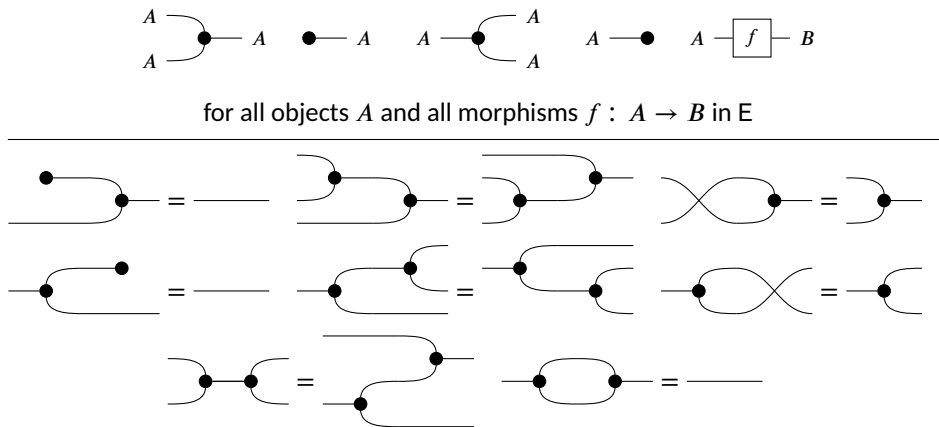


Figure 7.1

<sup>1</sup>Note that we are using the axiom of choice in this definition.

**Theorem 7.10** ([RSW08]). *There is a monoidal functor  $\mathbf{Colim} : \mathbf{CDiag} \rightarrow \mathbf{Cospan}(\mathbf{E})$  from discrete cospans of diagrams to cospans in  $\mathbf{E}$  that extends the colimit functor  $\mathbf{colim} : \mathbf{Diag}(\mathbf{E}) \rightarrow \mathbf{E}$ .*

The functor  $\mathbf{Colim}$  makes the problem of computing colimits functorial. For some choices of the category  $\mathbf{E}$ , we can show that there is a compositional algorithm for computing colimits.

**Colimits in  $\mathbf{FinSet}$ .** Suppose that we are working with a representation of finite sets that allows computations of disjoint unions in constant time  $c_0$ . Computing the colimit of a finite diagram  $d : G \rightarrow |\mathbf{FinSet}|$  by brute force means to take the disjoint union of the sets  $d(v)$  for  $v \in \text{vertices}(G)$  and then quotienting by the equivalence relation given by the edges of  $G$ .

$$\mathbf{colim} d = \left( \bigsqcup_{v \in \text{vertices}(G)} d(v) \right) / \sim$$

The equivalence relation  $\sim$  is the transitive closure of a relation  $\sim_0$ . Two elements in this union,  $a, b \in \bigsqcup_{v \in \text{vertices}(G)} d(v)$ , are related,  $a \sim_0 b$ , if and only if there are edges  $e_1 = (u, v_1)$  and  $e_2 = (u, v_2)$  of  $G$  and an element  $y \in d(u)$  that maps to  $a$  and  $b$ :  $d(e_1)(y) = a$  and  $d(e_2)(y) = b$ . We can encode these relations as square boolean matrices  $E_0$  and  $E$  whose dimension is the sum of the cardinalities of the images of the functions in the diagram:  $\sum_{e \in \text{edges}(G)} |\text{im}(d(e))|$ . As we do not have further information on the shape of the colimit, we can bound this size with  $n = \sum_{v \in \text{vertices}(G)} |d(v)|$ . The matrix  $E_0$  can be computed in  $\mathcal{O}(n^2)$  time, it is symmetric and has all the diagonal elements equal to 1. This means that it represents a symmetric and reflexive relation. The matrix  $E$  needs to be computed from  $E_0$  by transitive closure. A square boolean matrix  $A$  represents a transitive relation if and only if  $A = A \cdot A$ . Then, computing  $E$  from  $E_0$  means computing  $E_{k+1} = E_k \cdot E_k$  until convergence:  $E_{k+1} = E_k$ . This procedure terminates in at most  $n^2$  steps and each step takes  $\mathcal{O}(n^3)$  for matrix multiplication, which gives a running time of  $\mathcal{O}(n^5)$ .

Cospans compose by pushout. To show Assumption 2, we need to bound the computational cost of computing pushouts in  $\mathbf{Set}$ . The pushout  $U +_Y V$  of  $u : Y \rightarrow U$  and  $v : Y \rightarrow V$  in  $\mathbf{Set}$  is their disjoint union  $U + V$  quotiented by the equivalence relation generated by  $u$  and  $v$ . As for generic colimits,  $a \sim_0 b$  if there is a  $y \in Y$  such that  $u(y) = a$  and  $v(y) = b$ . The relation  $\sim_0$  can be easily made symmetric and reflexive, while computing its transitive closure is a bit more computationally involved. We record the relation  $\sim_0$  in a square boolean matrix  $E_0$ , but, this time, its size can be bound by  $2 \cdot |Y|$  because  $|Y|$  bounds the number of elements in the images of both  $u$  and  $v$ . By the same reasoning as above, we can compute its transitive closure in  $\mathcal{O}(|Y|^5)$ . With the computation of the disjoint union, this makes  $\mathcal{O}(|Y|^5 + c_0) = \mathcal{O}(|Y|^5)$  and we have shown Assumption 2.

As just mentioned, computing disjoint unions takes  $\mathcal{O}(c_0)$ , which satisfies Assumption 3 about computing monoidal products.

We have shown that colimits in  $\mathbf{FinSet}$  can be computed compositionally and Theorem 7.6 applies to this problem.

**Colimits in presheaves.** Computing the colimit of a finite diagram in a (co)presheaf category  $[\mathbf{C}, \mathbf{FinSet}]$  means computing the same colimit in  $\mathbf{FinSet}$  for each object  $A$  in  $\mathbf{C}$  and computing the corresponding unique morphism for every morphism  $f : A \rightarrow B$  in  $\mathbf{C}$ . When the category  $\mathbf{C}$  is finite, this can be done in finite time. We assume that this is the case and let  $s$  be the maximum between the number of objects and the number of morphisms.

Consider a diagram  $d : G \rightarrow [[\mathbf{C}, \mathbf{FinSet}]]$  in the presheaf category  $[\mathbf{C}, \mathbf{FinSet}]$ . This diagram determines functors  $\mathbf{D}_v := d(v)$ , for every vertex  $v$  of  $G$ , and natural transformations  $\gamma_e := d(e)$ , for every edge  $e$  in  $G$ . For every object  $A$  in  $\mathbf{C}$ , the diagram  $d$  in  $[\mathbf{C}, \mathbf{FinSet}]$  determines a diagram  $d_A : G \rightarrow |\mathbf{FinSet}|$  of the same



shape in  $\text{FinSet}$ , defined on vertices by  $d_A(v) := \mathbf{D}_v(A)$  and on edges by  $d_A(e) := \gamma_e(A)$ . For a functor  $\mathbf{F} : \mathbf{C} \rightarrow \text{FinSet}$ , we let its cardinality to be the maximum cardinality of the sets in its image,  $|\mathbf{F}| := \max_{A \in \text{Obj}(\mathbf{C})} |\mathbf{F}(A)|$ .

The colimit of  $d$  is computed component-wise: for every object  $A$  of  $\mathbf{C}$ , we compute the colimit in  $\text{FinSet}$  of the diagram  $d_A$ , and, for every morphism  $f : A \rightarrow B$  in  $\mathbf{C}$ , we compute the unique colimit function  $\text{colim } f : \text{colim } d_A \rightarrow \text{colim } d_B$ . The time complexity of computing  $\text{colim } d_A$  for each object  $A$  is  $\mathcal{O}(n_A^5)$ , where  $n_A := \sum_{v \in \text{vertices } G} |d_A(v)|$ . As a consequence, computing all these colimits takes  $\mathcal{O}(s \cdot n^5)$ , where  $n := \sum_{v \in \text{vertices } G} |\mathbf{D}_v|$ . The computation of the corresponding morphisms is irrelevant as it is linear in  $s \cdot n$ . When computing  $\text{colim } d$ , we recorded all the injections  $i_v^A : d_A(v) \rightarrow \text{colim } d_A$ . Then, for each object  $A$  of  $\mathbf{C}$  and each vertex  $v$  of  $G$ , we define the colimit of  $f$  thanks to the universal property:  $\text{colim } f(i_v^A(x)) := i_v^B(\mathbf{D}_v f(x))$ . The image on some elements in  $\text{colim } d_A$  is computed more than once, but, thanks to the universal property, all these values coincide and we have computed  $\text{colim } f$  going through at most  $s \cdot n$  elements.

For compositions in  $\text{Cospan}([\mathbf{C}, \text{FinSet}])$ , we need to compute pushouts in  $[\mathbf{C}, \text{FinSet}]$ . As explained above, we can reuse the complexity bounds for  $\text{FinSet}$  and deduce that the time complexity of computing the pushout  $\mathbf{U} +_{\mathbf{Y}} \mathbf{V}$  of  $\mathbf{U}$  and  $\mathbf{V}$  along  $\mathbf{Y}$  is  $\mathcal{O}(s \cdot |\mathbf{Y}|^5)$ . Similarly, monoidal products in  $\text{Cospan}([\mathbf{C}, \text{FinSet}])$  correspond to coproducts in  $[\mathbf{C}, \text{FinSet}]$ , and computing  $\mathbf{U} + \mathbf{V}$  takes  $\mathcal{O}(s \cdot c_0)$ .



## Chapter 8

# Conclusions

This thesis has defined monoidal width, a structural complexity measure of morphisms in monoidal categories based on the corresponding notion of monoidal decomposition. This interpretation is validated by the results that show that monoidal width, in the monoidal category of graphs with vertex interfaces, is equivalent to tree width, and, in the monoidal category of graphs with edge interfaces, is equivalent to clique width. We have concluded with a fixed-parameter tractability result. Functorial problems that admit a compositional algorithm can be computed in linear time on morphisms of bounded monoidal width. An example of such a problem is computing colimits in presheaf categories.

**Future work** Monoidal categories often represent process theories or semantic universes for programming languages. Applications of monoidal width to such monoidal categories remain to be explored. There may be problems on these monoidal categories that satisfy the assumptions for the monoidal fixed-parameter tractability result and, for these problems, we would obtain that they are tractable on morphisms of bounded monoidal width.

This work does not deal with the problem of finding efficient decompositions in general, which is, indeed, an important problem. We do not expect to find a general purpose tractable algorithm for finding efficient monoidal decompositions, as that would particularise to one for clique decompositions and it is still an open problem whether graphs of bounded clique width can be recognised in polynomial time [Oum08]. However, this problem could be studied in some finitely presented props. The results about categories with biproducts in Section 3.3 are a first step in this direction as they construct, given unique  $\otimes$ -decompositions of the objects, minimal monoidal decompositions of morphisms.

Monoidal width can capture tree width and clique width by changing the categorical algebra that describes graphs. Twin width [Bon+21] is a recently defined graph width measure which is similar in flavour to clique width but stronger, in the sense that bounded twin width graphs must have bounded clique width but vice versa does not hold. Future work could look for a categorical algebra to capture twin width.

Game comonads [ADW17] capture decompositions with coalgebras. On the other hand, produoidal categories give the algebra for decompositions in monoidal categories [EHR23]. These lines of work suggest that there might be some categorical structure that captures monoidal decompositions as well.



# Bibliography

- [AC09] Samson Abramsky and Bob Coecke. “Categorical Quantum Mechanics”. In: *Handbook of Quantum Logic and Quantum Structures*. Ed. by Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann. Amsterdam: Elsevier, 2009, pp. 261–323. ISBN: 978-0-444-52869-8. DOI: 10.1016/B978-0-444-52869-8.50010-4.
- [ADW17] Samson Abramsky, Anuj Dawar, and Pengming Wang. “The pebbling comonad in finite model theory”. In: *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2017, pp. 1–12. DOI: 10.1109/LICS.2017.8005129.
- [AJP22] Samson Abramsky, Tomáš Jakl, and Thomas Paine. “Discrete density comonads and graph parameters”. In: *Coalgebraic Methods in Computer Science*. Ed. by Helle Hvid Hansen and Fabio Zanasi. Springer International Publishing, 2022, pp. 23–44. ISBN: 978-3-031-10736-8. DOI: 10.1007/978-3-031-10736-8\_2.
- [AM21] Samson Abramsky and Dan Marsden. “Comonadic semantics for guarded fragments”. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470594.
- [AM99] Samson Abramsky and Guy McCusker. “Game Semantics”. In: *Computational Logic*. Ed. by Ulrich Berger and Helmut Schwichtenberg. NATO ASI Series. Berlin, Heidelberg: Springer, 1999, pp. 1–55. DOI: 10.1007/978-3-642-58622-4\_1.
- [AR23] Samson Abramsky and Luca Reggio. “Arboreal categories: An axiomatic theory of resources”. In: *Logical Methods in Computer Science* 19.3 (Aug. 2023). DOI: 10.46298/lmcs-19(3:14)2023.
- [AS21] Samson Abramsky and Nihil Shah. “Relating structure and power: Comonadic semantics for computational resources”. In: *Journal of Logic and Computation* 31.6 (Aug. 2021), pp. 1390–1428. ISSN: 0955-792X. DOI: 10.1093/logcom/exab048.
- [BFP16] John C Baez, Brendan Fong, and Blake S Pollard. “A compositional framework for Markov processes”. In: *Journal of Mathematical Physics* 57.3 (Mar. 2016), p. 033301. ISSN: 0022-2488. DOI: 10.1063/1.4941578.
- [BM20] John C Baez and Jade Master. “Open petri nets”. In: *Mathematical Structures in Computer Science* 30.3 (2020), pp. 314–341. DOI: 10.1017/S0960129520000043.
- [BP17] John C Baez and Blake S Pollard. “A compositional framework for reaction networks”. In: *Reviews in Mathematical Physics* 29.09 (2017), p. 1750028. DOI: 10.1142/S0129055X17500283.
- [BC87] Michel Bauderon and Bruno Courcelle. “Graph expressions and graph rewritings”. In: *Mathematical Systems Theory* 20.1 (1987), pp. 83–127. DOI: 10.1007/BF01692060.
- [Bén67] Jean Bénabou. “Introduction to bicategories”. In: *Reports of the Midwest Category Seminar*. Springer, 1967, pp. 1–77.

- [BB73] Umberto Bertele and Francesco Brioschi. “On non-serial dynamic programming”. In: *Journal of Combinatorial Theory, Series A* 14.2 (1973), pp. 137–148.
- [Blu+11] Christoph Blume, HJ Sander Bruggink, Martin Friedrich, and Barbara König. “Treewidth, path-width and cospan decompositions”. In: *Electronic Communications of the EASST* 41 (2011). ISSN: 1863-2122. DOI: 10.14279/tuj.eceasst.41.643.
- [Bod93a] Hans L Bodlaender. “A linear time algorithm for finding tree-decompositions of small treewidth”. In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. Ed. by S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal. 1993, pp. 226–234. DOI: 10.1145/167088.167161.
- [Bod93b] Hans L Bodlaender. “A tourist guide through treewidth”. In: *Acta Cybernetica* 11.1-2 (1993), pp. 1–21.
- [Bon+19a] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. “Diagrammatic algebra: from linear to concurrent systems”. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), 25:1–25:28. DOI: 10.1145/3290338.
- [Bon+19b] Filippo Bonchi, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. “Graphical Affine Algebra”. In: *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785877.
- [BSS18] Filippo Bonchi, Jens Seeber, and Paweł Sobociński. “Graphical Conjunctive Queries”. In: *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*. Ed. by Dan Ghica and Achim Jung. Vol. 119. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 13:1–13:23. ISBN: 978-3-95977-088-0. DOI: 10.4230/LIPIcs.CSL.2018.13.
- [BSZ14] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. “A categorical semantics of signal flow graphs”. In: *International Conference on Concurrency Theory*. Ed. by Paolo Baldan and Daniele Gorla. Springer Berlin Heidelberg, 2014, pp. 435–450. ISBN: 978-3-662-44584-6. DOI: 10.1007/978-3-662-44584-6\_30.
- [BSZ15] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. “Full abstraction for signal flow graphs”. In: *ACM SIGPLAN Notices*. POPL ’15 50.1 (2015), pp. 515–526. ISSN: 0362-1340. DOI: 10.1145/2775051.2676993.
- [BSZ17] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. “Interacting Hopf algebras”. In: *Journal of Pure and Applied Algebra* 221.1 (2017), pp. 144–184. ISSN: 0022-4049. DOI: 10.1016/j.jpaa.2016.06.002.
- [Bon+21] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. “Twin-width I: tractable FO model checking”. In: *ACM Journal of the ACM* 69.1 (2021), pp. 1–46. DOI: 10.1145/3486655.
- [Bum21] Benjamin Merlin Bumpus. “Generalizing graph decompositions”. PhD thesis. University of Glasgow, 2021.
- [BKM23] Benjamin Merlin Bumpus, Zoltan Kocsis, and Jade Edenstar Master. *Structured Decompositions: Structural and Algorithmic Compositionality*. 2023. arXiv: 2207.06091 [math.CT].
- [BK21] Benjamin Merlin Bumpus and Zoltan A Kocsis. *Spined categories: generalizing tree-width beyond graphs*. 2021. arXiv: 2104.01841 [math.CO].
- [Car87] Aurelio Carboni. “Bicategories of partial maps”. In: *Cahiers de topologie et géométrie différentielle catégoriques* 28.2 (1987), pp. 111–126.

- [CVP21] Titouan Carette, Marc de Visme, and Simon Perdrix. "Graphical Language with Delayed Trace: Picturing Quantum Computing with Finite Memory". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470553.
- [CS15] Apiwat Chantawibul and Paweł Sobociński. "Towards compositional graph theory". In: *Electronic Notes in Theoretical Computer Science* 319 (2015), pp. 121–136. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2015.12.009.
- [CHP17] Florence Clerc, Harrison Humphrey, and Prakash Panangaden. "Bicategories of Markov processes". In: *Models, Algorithms, Logics and Tools: Essays Dedicated to Kim Guldstrand Larsen on the Occasion of His 60th Birthday* (2017). Ed. by Luca Aceto, Giorgio Bacci, Giovanni Bacci, Anna Ingólfssdóttir, Axel Legay, and Radu Mardare, pp. 112–124. DOI: 10.1007/978-3-319-63121-9\_6.
- [CL07] Robin Cockett and Stephen Lack. "Restriction categories III: colimits, partial limits and extensivity". In: *Mathematical Structures in Computer Science* 17.4 (2007), pp. 775–817. DOI: 10.1017/S0960129507006056.
- [CFS16] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. "A mathematical theory of resources". In: *Information and Computation* 250 (2016), pp. 59–86. DOI: 10.1016/j.ic.2016.02.008.
- [CS12] Bob Coecke and Robert W Spekkens. "Picturing classical and quantum Bayesian inference". In: *Synthese* 186 (2012), pp. 651–696. DOI: 10.1007/s11229-011-9917-5.
- [Cou90] Bruno Courcelle. "The monadic second-order logic of graphs. I. Recognizable sets of finite graphs". In: *Information and computation* 85.1 (1990), pp. 12–75. ISSN: 0890-5401. DOI: 10.1016/0890-5401(90)90043-H.
- [Cou92a] Bruno Courcelle. "The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues". In: *RAIRO-Theoretical Informatics and Applications* 26.3 (1992), pp. 257–286.
- [Cou92b] Bruno Courcelle. "The monadic second-order logic of graphs VII: Graphs as relational structures". In: *Theoretical Computer Science* 101.1 (1992), pp. 3–33. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)90148-9.
- [CER93] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. "Handle-rewriting hypergraph grammars". In: *Journal of computer and system sciences* 46.2 (1993), pp. 218–270. ISSN: 0022-0000. DOI: 10.1016/0022-0000(93)90004-G.
- [CK07] Bruno Courcelle and Mamadou Moustapha Kanté. "Graph operations characterizing rank-width and balanced graph expressions". In: *Graph-Theoretic Concepts in Computer Science: 33rd International Workshop, WG 2007, Dornburg, Germany, June 21-23, 2007. Revised Papers* 33. Ed. by Andreas Brandstädt, Dieter Kratsch, and Haiko Müller. Springer Berlin Heidelberg, 2007, pp. 66–75. ISBN: 978-3-540-74839-7. DOI: 10.1007/978-3-540-74839-7\_7.
- [CK09] Bruno Courcelle and Mamadou Moustapha Kanté. "Graph operations characterizing rank-width". In: *Discrete Applied Mathematics* 157.4 (2009), pp. 627–640. ISSN: 0166-218X. DOI: 10.1016/j.dam.2008.08.026.
- [CMR00] Bruno Courcelle, Johann A Makowsky, and Udi Rotics. "Linear time solvable optimization problems on graphs of bounded clique-width". In: *Theory of Computing Systems* 33.2 (2000), pp. 125–150. ISSN: 1433-0490. DOI: 10.1007/s002249910009.
- [CM02] Bruno Courcelle and Johann A. Makowsky. "Fusion in relational structures and the verification of monadic second-order properties". In: *Mathematical Structures in Computer Science* 12.2 (2002), pp. 203–235. DOI: 10.1017/S0960129501003565.

- [CO00] Bruno Courcelle and Stephan Olariu. “Upper bounds to the clique width of graphs”. In: *Discrete Applied Mathematics* 101.1 (2000), pp. 77–114. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(99)00184-5.
- [CV03] Bruno Courcelle and Rémi Vanicat. “Query efficient implementation of graphs of bounded clique-width”. In: *Discrete Applied Mathematics* 131.1 (2003), pp. 129–150. ISSN: 0166-218X. DOI: 10.1016/S0166-218X(02)00421-3.
- [CO89] P-L Curien and A Obtulowicz. “Partiality, cartesian closedness, and toposes”. In: *Information and Computation* 80.1 (1989), pp. 50–95. ISSN: 0890-5401. DOI: 10.1016/0890-5401(89)90023-0.
- [DJR21] Anuj Dawar, Tomáš Jakl, and Luca Reggio. “Lovász-type theorems and game comonads”. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2021, pp. 1–13. DOI: 10.1109/LICS52264.2021.9470609.
- [DFR22] Elena Di Lavore, Giovanni de Felice, and Mario Román. “Monoidal Streams for Dataflow Programming”. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. 2022, pp. 1–14. DOI: 10.1145/3531130.3533365. arXiv: 2202.02061 [cs.LG].
- [Di+21a] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. “A Canonical Algebra of Open Transition Systems”. In: *Formal Aspects of Component Software*. Ed. by Gwen Salaün and Anton Wijs. Vol. 13077. Cham: Springer International Publishing, 2021, pp. 63–81. ISBN: 978-3-030-90636-8. DOI: 10.1007/978-3-030-90636-8\_4. arXiv: 2010.10069v1 [math.CT].
- [Di+23] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. “Span(Graph): a Canonical Feedback Algebra of Open Transition Systems”. In: *Software and Systems Modeling* 22 (2023), pp. 495–520. DOI: 10.1007/s10270-023-01092-7. arXiv: 2010.10069 [math.CT].
- [DHS21] Elena Di Lavore, Jules Hedges, and Paweł Sobociński. “Compositional Modelling of Network Games”. In: *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Ed. by Christel Baier and Jean Goubault-Larrecq. Vol. 183. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021, 30:1–30:24. ISBN: 978-3-95977-175-7. DOI: 10.4230/LIPIcs.CSL.2021.30. arXiv: 2006.03493 [cs.GT].
- [DR23] Elena Di Lavore and Mario Román. “Evidential Decision Theory via Partial Markov Categories”. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2023, pp. 1–14. DOI: 10.1109/LICS56636.2023.10175776.
- [DS22] Elena Di Lavore and Paweł Sobociński. “Monoidal Width: Capturing Rank Width”. In: *Proceedings Fifth International Conference on Applied Category Theory*, Glasgow, United Kingdom, 18–22 July 2022. Ed. by Jade Master and Martha Lewis. Vol. 380. Electronic Proceedings in Theoretical Computer Science. Open Publishing Association, 2022, pp. 268–283. DOI: 10.4204/EPTCS.380.16.
- [DS23] Elena Di Lavore and Paweł Sobociński. “Monoidal Width”. In: *Logical Methods in Computer Science* 19 (3 Sept. 2023). DOI: 10.46298/lmcs-19(3:15)2023.
- [Di+21b] Ivan Di Liberti, Fosco Loregian, Chad Nester, and Paweł Sobociński. “Functorial semantics for partial theories”. In: *Proceedings of the ACM on Programming Languages* 5.POPL (2021), pp. 1–28. DOI: 10.1145/3434338.
- [EHR23] Matt Earnshaw, James Hefford, and Mario Román. *The Produoidal Algebra of Process Decomposition*. 2023. arXiv: 2301.11867 [cs.LG].



- [Ehr57] Andrzej Ehrenfeucht. "Application of games to some problems of mathematical logic". In: *Bulletin of the Polish Academy of Sciences Cl. III* 5 (1957), pp. 35–37.
- [Ehr61] Andrzej Ehrenfeucht. "An application of games to the completeness problem for formalized theories". In: *Fundamenta Mathematicae* 49.2 (1961), pp. 129–141. ISSN: 0016-2736.
- [Fef57] Solomon Feferman. "Some recent work of Ehrenfeucht and Fraïssé". In: *Proceedings of the Summer Institute of Symbolic Logic, Ithaca (1957)*, pp. 201–209.
- [FV59] Solomon Feferman and Robert L Vaught. "The first order properties of products of algebraic systems". In: *Fundamenta Mathematicae* 47 (1959), pp. 57–103. ISSN: 0016-2736.
- [FS07] José Luiz Fiadeiro and Vincent Schmitt. "Structured co-spans: an algebra of interaction protocols". In: *International Conference on Algebra and Coalgebra in Computer Science*. Ed. by Till Mossakowski, Ugo Montanari, and Magne Haveraaen. Springer. 2007, pp. 194–208. ISBN: 978-3-540-73859-6. DOI: 10.1007/978-3-540-73859-6\_14.
- [Fon15] Brendan Fong. "Decorated Cospans". In: *Theory and Applications of Categories* 30.33 (2015), pp. 1096–1120.
- [Fox76] Thomas Fox. "Coalgebras and cartesian categories". In: *Communications in Algebra* 4.7 (1976), pp. 665–667. DOI: 10.1080/00927877608822127.
- [Fra55] Roland Fraïssé. "Sur quelques classifications des relations, basées sur les isomorphismes restraints, I: Études générale". In: *Publications Scientifiques de l'Université d'Alger, Série A 2* (1955), pp. 15–60.
- [Fra57] Roland Fraïssé. "Sur quelques classifications des relations, basées sur des isomorphismes restraints, II: application aux relations d'ordre, et construction d'exemples montrant que ces classifications sont distinctes". In: *Publications Scientifiques de l'Université d'Alger, Série A 2* (1957), pp. 273–295.
- [Fri20] Tobias Fritz. "A Synthetic Approach to Markov Kernels, Conditional Independence and Theorems on Sufficient Statistics". In: *Advances in Mathematics* 370 (2020), p. 107239. ISSN: 0001-8708. DOI: 10.1016/j.aim.2020.107239.
- [GH97] Fabio Gadducci and Reiko Heckel. "An inductive view of graph transformation". In: *International Workshop on Algebraic Development Techniques*. Springer. 1997, pp. 223–237. DOI: 10.1007/3-540-64299-4\_36.
- [GHL99] Fabio Gadducci, Reiko Heckel, and Merce Llabrés. "A bi-categorical axiomatisation of concurrent graph rewriting". In: *Electronic Notes in Theoretical Computer Science* 29 (1999), pp. 80–100. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80309-3.
- [Gar23] Richard Garner. "Stream processors and comodels". In: *Logical Methods in Computer Science* 19.1 (2023). DOI: 10.46298/lmcs-19(1:2)2023.
- [GRO0] Martin Charles Golumbic and Udi Rotics. "On the clique-width of some perfect graph classes". In: *International Journal of Foundations of Computer Science* 11.03 (2000), pp. 423–443. DOI: 10.1142/S0129054100000260.
- [Gui80] René Guitart. "Tenseurs et machines". In: *Cahiers de topologie et géométrie différentielle* 21.1 (1980), pp. 5–62.
- [Hal76] Rudolf Halin. "S-functions for graphs". In: *Journal of geometry* 8 (1976), pp. 171–186. DOI: 10.1007/BF01917434.
- [HV19] Chris Heunen and Jamie Vicary. *Categories for Quantum Theory: An Introduction*. Oxford University Press, Nov. 2019. ISBN: 9780198739623. DOI: 10.1093/oso/9780198739623.001.0001.

- [Hli+08] Petr Hliněný, Sang-il Oum, Detlef Seese, and Georg Gottlob. “Width parameters beyond tree-width and their applications”. In: *The computer journal* 51.3 (2008), pp. 326–362. DOI: 10.1093/comjnl/bxm052.
- [JMS23] Tomáš Jakl, Dan Marsden, and Nihil Shah. “A categorical account of composition methods in logic”. In: *2023 38th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2023, pp. 1–14. DOI: 10.1109/LICS56636.2023.10175751.
- [Jel10] Vít Jelínek. “The rank-width of the square grid”. In: *Discrete Applied Mathematics* 158.7 (2010), pp. 841–850. ISSN: 0166-218X. DOI: 10.1016/j.dam.2009.02.007.
- [JH90] He Jifeng and C. A. R. Hoare. “Categorical Semantics for Programming Languages”. In: *Mathematical Foundations of Programming Semantics*. Ed. by M. Main, A. Melton, M. Mislove, and D. Schmidt. Lecture Notes in Computer Science. New York, NY: Springer, 1990, pp. 402–417. DOI: 10.1007/BFb0040271.
- [JS91] André Joyal and Ross Street. “The geometry of tensor calculus, I”. In: *Advances in mathematics* 88.1 (1991), pp. 55–112.
- [Kat+00] Piergiulio Katis, Robert Rosebrugh, Nicoletta Sabadini, and Robert FC Walters. “An automata model of distributed systems”. In: *Proceedings of the Workshop on Trace Theory and Code Parallelization, Università degli Studi di Milano*. Vol. 125144. 2000.
- [KSW99] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. “On the algebra of feedback and systems with boundary”. In: *Rendiconti del Seminario Matematico di Palermo*. 1999.
- [KSW02] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. “Feedback, trace and fixed-point semantics”. In: *RAIRO-Theor. Informatics Appl.* 36.2 (2002), pp. 181–194. DOI: 10.1051/ita:2002009.
- [KSW97a] Piergiulio Katis, Nicoletta Sabadini, and Robert FC Walters. “Bicategories of processes”. In: *Journal of Pure and Applied Algebra* 115.2 (1997), pp. 141–178. ISSN: 0022-4049. DOI: 10.1016/S0022-4049(96)00012-6.
- [KSW97b] Piergiulio Katis, Nicoletta Sabadini, and Robert FC Walters. “Span (Graph): A categorical algebra of transition systems”. In: *Algebraic Methodology and Software Technology: 6th International Conference, AMAST’97 Sydney, Australia, December 13–17, 1997 Proceedings 6*. Springer. 1997, pp. 307–321. DOI: 10.1007/BFb0000479.
- [KSW04] Piergiulio Katis, Nicoletta Sabadini, and Robert FC Walters. “Compositional Minimization in Span (Graph): Some Examples”. In: *Electronic Notes in Theoretical Computer Science* 104 (2004), pp. 181–197. DOI: 10.1016/j.entcs.2004.08.025.
- [Lac04] Stephen Lack. “Composing props”. In: *Theory and Applications of Categories* 13.9 (2004), pp. 147–163.
- [Lam86] Joachim Lambek. “Cartesian Closed Categories and Typed  $\lambda$ -Calculi”. In: *Combinators and Functional Programming Languages*. Ed. by Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1986, pp. 136–175. ISBN: 978-3-540-47253-7. DOI: 10.1007/3-540-17184-3\_44.
- [Law63] F William Lawvere. “Functorial semantics of algebraic theories”. In: *Proceedings of the National Academy of Sciences* 50.5 (1963), pp. 869–872. DOI: 10.1073/pnas.50.5.869.
- [Mac65] Saunders Mac Lane. “Categorical algebra”. In: *Bulletin of the American Mathematical Society* 71 (1965), pp. 40–106. DOI: 10.1090/S0002-9904-1965-11234-4.
- [Mac78] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978, nil. DOI: 10.1007/978-1-4757-4721-8.

- [Mac63] Saunders MacLane. "Natural associativity and commutativity". In: *Rice Institute Pamphlet-Rice University Studies* 49.4 (1963).
- [Mak04] Johann A. Makowsky. "Algorithmic uses of the Feferman-Vaught theorem". In: *Annals of Pure and Applied Logic* 126.1-3 (2004), pp. 159–213. ISSN: 0168-0072. DOI: 10.1016/j.apal.2003.11.002.
- [MP96] Johann A. Makowsky and Yachin B Pnueli. "Ariety and alternation in second-order logic". In: *Annals of Pure and Applied Logic* 78.1-3 (1996), pp. 189–202. ISSN: 0168-0072. DOI: 10.1016/0168-0072(95)00013-5.
- [Mog91] Eugenio Moggi. "Notions of Computation and Monads". In: *Information and Computation* 93.1 (1991), pp. 55–92. DOI: 10.1016/0890-5401(91)90052-4.
- [MS22] Yoav Montacute and Nihil Shah. "The pebble-relation comonad in finite model theory". In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*. Association for Computing Machinery, 2022, pp. 1–11. ISBN: 9781450393515. DOI: 10.1145/3531130.3533335.
- [Mos52] Andrzej Mostowski. "On direct products of theories". In: *The Journal of Symbolic Logic* 17.1 (1952), pp. 1–31. DOI: 10.2307/2267454.
- [MD12] Abbe Mowshowitz and Matthias Dehmer. "Entropy and the complexity of graphs revisited". In: *Entropy* 14.3 (2012), pp. 559–570. DOI: 10.3390/e14030559.
- [MHH16] Koko Muroya, Naohiko Hoshino, and Ichiro Hasuo. "Memoryful geometry of interaction II: recursion and adequacy". In: *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*. Ed. by Rastislav Bodík and Rupak Majumdar. ACM, 2016, pp. 748–760. DOI: 10.1145/2837614.2837672.
- [ÓD21] Adam Ó Conghaile and Anuj Dawar. "Game Comonads & Generalised Quantifiers". In: *29th EACSL Annual Conference on Computer Science Logic*. Ed. by Christel Baier and Jean Goubault-Larrecq. Vol. 183. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2021, 16:1–16:17. ISBN: 978-3-95977-175-7. DOI: 10.4230/LIPIcs.CSL.2021.16.
- [Oli84] José Nuno Oliveira. "The formal semantics of deterministic dataflow programs". PhD thesis. University of Manchester, UK, 1984.
- [Oum05] Sang-il Oum. "Graphs of bounded rank-width". PhD thesis. Princeton University, 2005.
- [Oum08] Sang-il Oum. "Approximating rank-width and clique-width quickly". In: *ACM Transactions on Algorithms (TALG)* 5.1 (2008), pp. 1–20. ISSN: 1549-6325. DOI: 10.1145/1435375.1435385.
- [OS06] Sang-il Oum and Paul D. Seymour. "Approximating clique-width and branch-width". In: *Journal of Combinatorial Theory, Series B* 96.4 (2006), pp. 514–528. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2005.10.006.
- [Pai20] Thomas Paine. "A pebbling comonad for finite rank and variable logic, and an application to the equirank-variable homomorphism preservation theorem". In: *Electronic Notes in Theoretical Computer Science* 352 (2020), pp. 191–209. ISSN: 1571-0661. DOI: 10.1016/j.entcs.2020.09.010.
- [Pan99] Prakash Panangaden. "The Category of Markov Kernels". In: *Electronic Notes in Theoretical Computer Science*. PROBMIV'98, First International Workshop on Probabilistic Methods in Verification 22 (Jan. 1999), pp. 171–187. ISSN: 1571-0661. DOI: 10.1016/S1571-0661(05)80602-4.

- [PO99] R Piziak and PL Odell. "Full rank factorization of matrices". In: *Mathematics magazine* 72.3 (1999), pp. 193–201. DOI: 10.2307/2690882.
- [PRS88] Pavel Pudlák, Vojtěch Rödl, and Petr Savický. "Graph complexity". In: *Acta Informatica* 25.5 (1988), pp. 515–535. DOI: 10.1007/BF00279952.
- [RS83] Neil Robertson and Paul D Seymour. "Graph minors. I. Excluding a forest". In: *Journal of Combinatorial Theory, Series B* 35.1 (1983), pp. 39–61. ISSN: 0095-8956. DOI: 10.1016/0095-8956(83)90079-5.
- [RS86] Neil Robertson and Paul D Seymour. "Graph minors. II. Algorithmic aspects of tree-width". In: *Journal of algorithms* 7.3 (1986), pp. 309–322. ISSN: 0196-6774. DOI: 10.1016/0196-6774(86)90023-4.
- [RS90] Neil Robertson and Paul D Seymour. "Graph minors. IV. Tree-width and well-quasi-ordering". In: *Journal of Combinatorial Theory, Series B* 48.2 (1990), pp. 227–254. ISSN: 0095-8956. DOI: 10.1016/0095-8956(90)90120-0.
- [RS91] Neil Robertson and Paul D Seymour. "Graph minors. X. Obstructions to tree-decomposition". In: *Journal of Combinatorial Theory, Series B* 52.2 (1991), pp. 153–190. ISSN: 0095-8956. DOI: 10.1016/0095-8956(91)90061-N.
- [RSO4] Neil Robertson and Paul D Seymour. "Graph minors. XX. Wagner's conjecture". In: *Journal of Combinatorial Theory, Series B* 92.2 (2004), pp. 325–357. ISSN: 0095-8956. DOI: 10.1016/j.jctb.2004.08.001.
- [RR88] Edmund Robinson and Giuseppe Rosolini. "Categories of partial maps". In: *Information and computation* 79.2 (1988), pp. 95–130. ISSN: 0890-5401. DOI: 10.1016/0890-5401(88)90034-X.
- [Rom23] Mario Román. "Promonads and String Diagrams for Effectful Categories". In: *Proceedings Fifth International Conference on Applied Category Theory*, Glasgow, United Kingdom, 18-22 July 2022. Ed. by Jade Master and Martha Lewis. Vol. 380. *Electronic Proceedings in Theoretical Computer Science*. Open Publishing Association, 2023, pp. 344–361. DOI: 10.4204/EPTCS.380.20.
- [RSW04] Robert Rosebrugh, Nicoletta Sabadini, and Robert FC Walters. "Minimisation and minimal realisation in Span (Graph)". In: *Mathematical Structures in Computer Science* 14.5 (2004), pp. 685–714. DOI: 10.1017/S096012950400430X.
- [RSW05] Robert Rosebrugh, Nicoletta Sabadini, and Robert FC Walters. "Generic commutative separable algebras and cospans of graphs". In: *Theory and applications of categories* 15.6 (2005), pp. 164–177.
- [RSW08] Robert Rosebrugh, Nicoletta Sabadini, and Robert FC Walters. "Calculating colimits compositionally". In: *Concurrency, Graphs and Models: Essays Dedicated to Ugo Montanari on the Occasion of His 65th Birthday*. Ed. by Pierpaolo Degano, Rocco De Nicola, and José Meseguer. Vol. 5065. Springer, 2008, pp. 581–592. ISBN: 978-3-540-68679-8. DOI: 10.1007/978-3-540-68679-8\_36.
- [RB88] David E Rydeheard and Rod M Burstall. *Computational category theory*. Vol. 152. Prentice Hall Englewood Cliffs, 1988.
- [Sel11] Peter Selinger. "A survey of graphical languages for monoidal categories". In: *New structures for physics*. Springer, 2011, pp. 289–355. ISBN: 978-3-642-12821-9. DOI: 10.1007/978-3-642-12821-9\_4.

- [SK19] David Sprunger and Shin-ya Katsumata. “Differentiable Causal Computations via Delayed Trace”. In: *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*. IEEE, 2019, pp. 1–12. DOI: 10.1109/LICS.2019.8785670.
- [Sta17] Sam Staton. “Commutative semantics for probabilistic programming”. In: *European Symposium on Programming*. Springer Berlin Heidelberg, 2017, pp. 855–879. ISBN: 978-3-662-54434-1. DOI: 10.1007/978-3-662-54434-1\_32.
- [Ște86a] Gheorghe Ștefănescu. “An algebraic theory of flowchart schemes”. In: *CAAP '86*. Ed. by Paul Franchi-Zannettacci. Springer Berlin Heidelberg, 1986, pp. 60–73. ISBN: 978-3-540-39783-0. DOI: 10.1007/BFb0022659.
- [Ște86b] Gheorghe Ștefănescu. *Feedback Theories (a Calculus for Isomorphism Classes of Flowchart Schemes)*. 24. Institutul de Matematica, 1986.
- [Ste21] Dario Maximilian Stein. “Structural Foundations for Probabilistic Programming Languages”. In: *University of Oxford* (2021).
- [UV08] Tarmo Uustalu 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 Jiří Adámek and Clemens Kupke. Vol. 203. Electronic Notes in Theoretical Computer Science. Elsevier, 2008, pp. 263–284. DOI: 10.1016/j.entcs.2008.05.029.
- [Zan15] Fabio Zanasi. “Interacting Hopf Algebras - The Theory of Linear Systems”. PhD thesis. École Normale Supérieure de Lyon, 2015.