DOCTORAL THESIS

Monoidal Width

Elena Di Lavore

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2023

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
55/2023

Monoidal Width

ELENA DI LAVORE

TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
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 Information Technologies,
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

European Union
European Regional
Development Fund

Investing
in your future

ISSN 2585-6898 (publication)
ISBN 978-9916-80-075-1 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-076-8 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
55/2023

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; mudeleid 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 struktuurset keerukust ning kui graafi laius on väike, on osalahenduste kombineerimine praktiliselt arvutatav. Siinne töö üritab tuua parametriseeritud keerukuses kasutatavad võtted monoidilisse kategooriateooriasse.

Käesolev doktoritöö toob sisse monoidilise laiuse mõiste, et mõõta morfismide struktuurset keerukust monoidilistes kategooriates, ning uurib mõningaid selle omadusi. Valides sobiva kategoorsed algebrad, 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 funktoriaalsed 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

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-Mostowsky theorem [Fef57; FV59] that shows, via Ehrenfeucht-Fraissé 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 that 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.

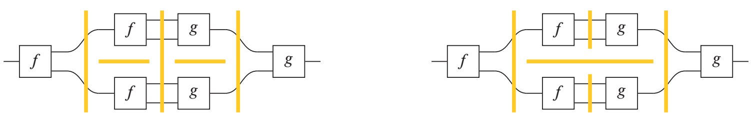


Figure 1.1: An inefficient (left) and an efficient (right) monoidal decompositions.

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-Mostowsky preservation theorem [Fef57; FV59] relies on Ehrenfeucht-Fraissé 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 though 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 $\mathbf{C}_k(G) \to H$ 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 $G \to \mathbf{C}_k(G)$. The pebbling comonad defines tree width [ADW17], the Ehrenfeucht-Fraissé 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-Mostowsky 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.

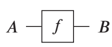
Chapter 2

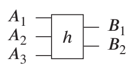
Background

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.

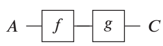
2.1 Monoidal categories

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; CL07; Di +21b], or quantum processes [AC09; 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.

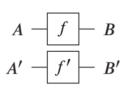




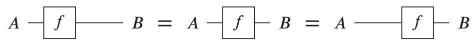
The categorical structure allows processes to be composed sequentially: for two morphisms $f : A \to B$ and $g : B \to C$, there is a composite morphism $f \mathbin{;} g : A \to C$ that, usually, represent the process of executing f first and then g.



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

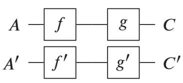


Both these composition operations have units. The identity morphisms $\mathbb{1}_A$ are the units for sequential composition: they represent the process that "does nothing" to a resource, so composing sequentially with the identity morphism should not change the process.



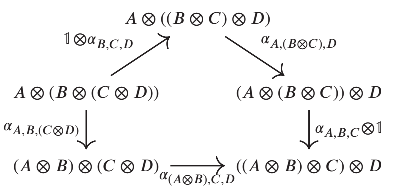
The monoidal unit I is the unit for parallel composition: it represents "absence of resources", so a process that produces as outputs, or requires as inputs, a resource A and the monoidal unit I, it is essentially the same as the same process only producing, or requiring, A. This reflects in the algebra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

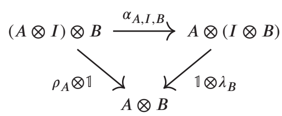
The string diagrammatic syntax is convenient because it hides the bureaucracy isomorphisms that ensure associativity and unitality of the monoidal structure, and equations like functoriality of the monoidal product, $(f \otimes f') ⨾ (g \otimes g') = (f ⨾ g) \otimes (f' ⨾ g')$ also become trivial in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



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, α, and unitality, λ and ρ, of the monoidal structure.

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

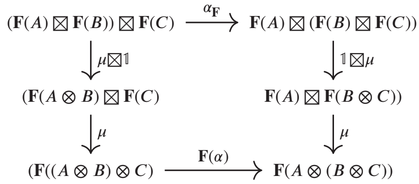




A monoidal category is *strict* if the coherence isomorphisms are identities.

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

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



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

Locally small monoidal categories and monoidal functors form a monoidal category MonCat with the Cartesian product and the one-object category.

Example 2.3 (The monoidal category of hypergraphs). A *hypergraph* $G = (V, E, \mathsf{ends})$ is a set of vertices V, a set of edges E and a function $\mathsf{ends} : E \to \wp(V)$[1]. A morphism $h : G \to H$ of graphs is a pair of functions $h_V : V_G \to V_H$ and $h_E : E_G \to E_H$ that preserve the adjacency relation: $h_E \,;\, \mathsf{ends}_H = \mathsf{ends}_G \,;\, \wp(h_V)$. Hypergraphs and their homomorphisms form a monoidal category 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 α, λ, ρ 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 C be a monoidal category, S be its strictification and let $\mathbf{H} : \mathsf{C} \to \mathsf{S}$ and $\mathbf{A} : \mathsf{S} \to \mathsf{C}$ 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 $f : A \to B$ in C, its image $\mathbf{A}(\mathbf{H}(f))$ does not necessarily coincide with f, but $f = \phi_A \,;\, \mathbf{A}(\mathbf{H}(f)) \,;\, \phi_B^{-1}$. This means that, every time we show an equality $u = v$ between morphisms $u, v : X \to Y$ in the strictification S, we can deduce that $f = g$, for all objects A and B and morphisms $f, g : A \to B$ in C such that $\mathbf{H}(f) = u$ and $\mathbf{H}(g) = v$, because $f = \phi_A \,;\, \mathbf{A}(u) \,;\, \phi_B^{-1} = \phi_A \,;\, \mathbf{A}(v) \,;\, \phi_B^{-1} = g$. In particular, a syntax for strict monoidal categories gives a syntax for monoidal categories.

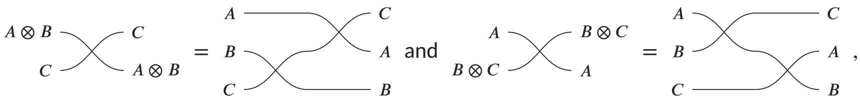
Example 2.6 (The monoidal category of monoidal signatures). A *monoidal signature* $\Sigma = E \rightrightarrows V^*$ is a set of types V, a set of generators E, and source and target functions $\mathsf{s}, \mathsf{t} : E \to V^*$ that associate to each generator the types of its inputs and outputs. A monoidal signature is *one-sorted* if V contains only one element. A morphism $h : \Sigma \to \Sigma'$ of monoidal signatures is a pair of functions $h_V : V \to V'$ and $h_E : E \to E'$ that preserve the inputs and outputs: $h_E \,;\, s' = s \,;\, h_V^*$ and $h_E \,;\, t' = t \,;\, h_V^*$. Monoidal signatures and their morphisms form a monoidal category 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 1MonSig of MonSig.

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

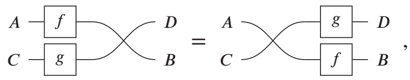
Theorem 2.7 ([JS91])**.** *String diagrams on a monoidal signature Σ form a strict monoidal category and, in fact, the free strict monoidal category on Σ.*

[1] We indicate with $\wp$ the covariant powerset functor.

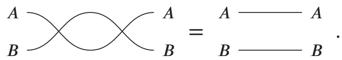
Symmetric monoidal categories are monoidal categories equipped with processes , the symmetries, that permute the order of resources. This family of processes is compatible with the monoidal structure,



it defines a natural transformation,



and it is an isomorphism,



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

$$\begin{array}{ccc} A \otimes (B \otimes C) & \xrightarrow{\alpha} & (A \otimes B) \otimes C \\ {\scriptstyle \mathbb{1} \otimes \sigma_{B,C}} \downarrow & & \downarrow {\scriptstyle \sigma_{A \otimes B, C}} \\ A \otimes (C \otimes B) & & C \otimes (A \otimes B) \\ {\scriptstyle \alpha} \downarrow & & \downarrow {\scriptstyle \alpha} \\ (A \otimes C) \otimes B & \xrightarrow[\sigma_{A,C} \otimes \mathbb{1}]{} & (C \otimes A) \otimes B \end{array} \qquad \begin{array}{ccc} (A \otimes B) \otimes C & \xrightarrow{\alpha^{-1}} & A \otimes (B \otimes C) \\ {\scriptstyle \sigma_{A,B} \otimes \mathbb{1}} \downarrow & & \downarrow {\scriptstyle \sigma_{A, B \otimes C}} \\ (B \otimes A) \otimes C & & (B \otimes C) \otimes A \\ {\scriptstyle \alpha^{-1}} \downarrow & & \downarrow {\scriptstyle \alpha^{-1}} \\ B \otimes (A \otimes C) & \xrightarrow[\mathbb{1} \otimes \sigma_{A,C}]{} & B \otimes (C \otimes A) \end{array}$$

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* $\mathbf{F} : \mathsf{C} \to \mathsf{D}$ between braided monoidal categories $(\mathsf{C}, \otimes, I)$ and $(\mathsf{D}, \boxtimes, J)$ is a monoidal functor that respects the braiding.

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

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 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 α, λ, ρ, σ

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

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

Given a monoidal signature Σ, a string diagram with symmetries over that signature is a string diagram over Σ 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 MonSig of monoidal signatures and the category SymMonCat of symmetric monoidal categories, where the free symmetric monoidal category on a monoidal signature Σ is given by string diagrams with symmetries over Σ [JS91, Theorem 2.3].

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

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 $n \to m$ 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 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 Prop that is a subcategory of 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 1MonSig and the category 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 $: 2 \to 1$ and the unit $: 0 \to 1$, 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 $\{\;,\;\}$, with E the free prop on three generators $\{u : 1 \to 1, a : 3 \to 1, c : 2 \to 1\}$, and with $\mathbf{l}, \mathbf{r} : \mathsf{E} \to \mathsf{M}_0$ 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:

$$\mathsf{E} \overset{\mathbf{l},\mathbf{r}}{\rightrightarrows} \mathsf{M}_0 \overset{\mathbf{q}}{\to} \mathsf{M}\ .$$

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

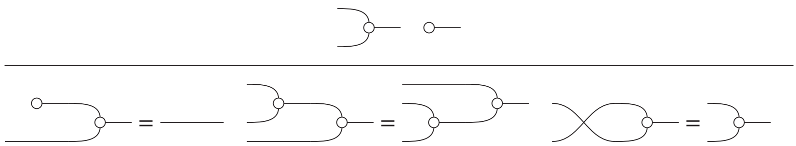


Figure 2.1: Generators and equations for a monoid.

Any prop S contains the initial prop P_0 of permutations as subprop. This determines two prop morphisms $\mathbf{l} :\ \mathsf{P}_0 \otimes \mathsf{S} \to \mathsf{S}$ and $\mathbf{r} :\ \mathsf{S} \otimes \mathsf{P}_0 \to \mathsf{S}$ by pre- and post-composition because $\mathsf{P}_0 \cong \mathsf{P}_0{}^{\mathrm{op}}$. The composition $\mathsf{S} \otimes_{\mathsf{P}_0} \mathsf{T}$ of two props S and T is the coequaliser of $\mathbb{1}_{\mathsf{S}} \otimes \mathbf{l}$ and $\mathbf{r} \otimes \mathbb{1}_{\mathsf{T}}$.

$$\mathsf{S} \otimes \mathsf{P}_0 \otimes \mathsf{T} \overset{\mathbb{1}\otimes\mathbf{l},\mathbf{r}\otimes\mathbb{1}}{\rightrightarrows} \mathsf{S} \otimes \mathsf{T} \to \mathsf{S} \otimes_{\mathsf{P}_0} \mathsf{T}$$

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* $\iota_{\mathsf{S}} :\ \mathsf{S} \to \mathsf{R}$ *and* $\iota_{\mathsf{T}} :\ \mathsf{T} \to \mathsf{R}$. *Suppose that any morphism* $r :\ m \to n$ *in* R *can be written as a composition* $r = \iota_{\mathsf{S}}(s) \mathbin{⨟} \iota_{\mathsf{T}}(t)$ *for some* $s :\ m \to p$ *in* S *and some* $t :\ p \to n$ *in* T, *uniquely up to permutations* $\sigma :\ p \to p$. *Then,* R *is the composite of* S *and* T *via a distributive law* $\lambda :\ \mathsf{T} \otimes_{\mathsf{P}_0} \mathsf{S} \to \mathsf{S} \otimes_{\mathsf{P}_0} \mathsf{T}$ *that associates to a pair* $(t \mid s)$ *the pair* $(\hat{s} \mid \hat{t})$, *where* $\iota_{\mathsf{S}}(\hat{s}) \mathbin{⨟} \iota_{\mathsf{T}}(\hat{t})$ *is the unique factorisation of* $\iota_{\mathsf{T}}(t) \mathbin{⨟} \iota_{\mathsf{S}}(s)$.

As explained in detail in Zanasi's PhD thesis [Zan15, Proposition 2.27], when composing finitely presented props, the distributive law λ gives the additional equations that determine the composite theory: for each pair $(t \mid s)$ in $\mathsf{T} \otimes_{\mathsf{P}_0} \mathsf{S}$, we add the equation $\iota_{\mathsf{T}}(t) \mathbin{⨟} \iota_{\mathsf{S}}(s) = \iota_{\mathsf{S}}(\hat{s}) \mathbin{⨟} \iota_{\mathsf{T}}(\hat{t})$. In other words, the composed prop $\mathsf{S} \otimes_{\mathsf{P}_0} \mathsf{T}$ is the coequaliser

$$\mathsf{T} \otimes_{\mathsf{P}_0} \mathsf{S} \overset{\mathbf{l},\mathbf{r}}{\rightrightarrows} \mathsf{S} + \mathsf{T} \dashrightarrow \mathsf{S} \otimes_{\mathsf{P}_0} \mathsf{T}$$

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

$$\mathbf{l}(t \mid s) := \iota_{\mathsf{T}}(t) \mathbin{⨟} \iota_{\mathsf{S}}(s) \qquad \text{and} \qquad \mathbf{r}(t \mid s) := \iota_{\mathsf{S}}(\hat{s}) \mathbin{⨟} \iota_{\mathsf{T}}(\hat{t})\ .$$

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* (Σ_1, E_1) *and* (Σ_2, E_2). *Then, their coproduct* $\mathsf{P}_1 + \mathsf{P}_2$ *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 ϕ is true in a graph or, more generally, in a relational structure G.

$$G \vDash \phi$$

This problem is hard in general, even when the formula ϕ is fixed. For example, for monadic second order logic and every level Σ_i^P of the polynomial hierarchy, there are formulae and classes of structures that make the model checking problem complete for Σ_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* $G = (V, E, \mathsf{ends})$ is determined by a function $\mathsf{ends} : E \to \wp(V)$ that assigns to each edge $e \in E$ a set of vertices $\mathsf{ends}(e) \subseteq V$, 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 τ of relational symbols with a specified arity $\alpha : \tau \to \mathbb{N}$.

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

Example 2.22. Graphs and hypergraphs can be encoded as relational structures for the signatures τ_{gr} and τ_{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 $e \in E$, $v, w \in \mathsf{ends}(e)$. 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 $v_i, v_{i+1} \in \mathsf{ends}(e_i)$ 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.

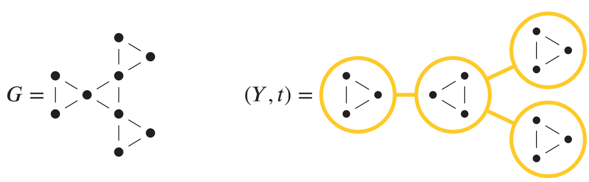
Tree width. Tree width, introduced by Robertson and Seymour [RS86, Section 1], measures the structural complexity of relational structures by comparing their structure to trees. In fact, forests have tree width 2, while the family of cliques has unbounded tree width. Tree width is based on tree decompositions, 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 $t : \mathsf{vertices}(Y) \to \wp(V)$ such that:

1. Every vertex v is in at least one of the bags $t(i)$, $\bigcup_{i \in \mathsf{vertices}(Y)} t(i) = V$.
2. For every edge $e \in E$ there is a node $i \in \mathsf{vertices}(Y)$ whose bag $t(i)$ contains all the endpoints $\mathsf{ends}(e)$ 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 $j \in \mathsf{vertices}(Y)$ that are on the path between i and k on the tree Y.

A tree decomposition of a relational τ-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 $i \in \mathsf{vertices}(Y)$ 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, $\mathsf{wd}(Y, t) := \max_{i \in \mathsf{vertices}(Y)} |t(i)|$. The *tree width* of G is given by the min-max formula:

$$\mathsf{twd}(G) := \min_{(Y,t)} \mathsf{wd}(Y, t).$$

Note that Robertson and Seymour subtract 1 from $\mathsf{twd}(G)$ 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 τ-structure G is the maximum arity of the relations with non-empty interpretation: $\gamma(G) := \max_{R^G \neq \emptyset} \alpha_R$. The *hyperedge size* of a relational signature τ is the maximum arity of its symbols: $\gamma(\tau) := \max_{R \in \tau} \alpha_R$.

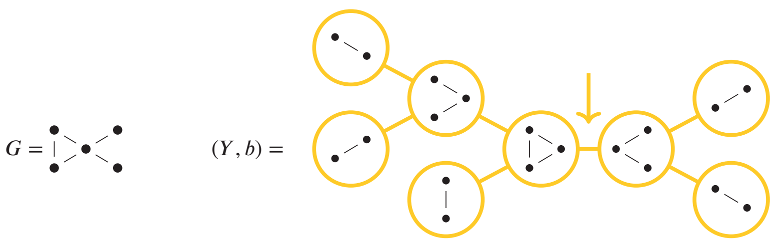
Theorem 2.30 ([RS91, Theorem 5.1])**.** *Branch width is equivalent to tree width. More precisely, for a hypergraph G,*

$$\max\{\mathsf{bwd}(G), \gamma(G)\} \leq \mathsf{twd}(G) \leq \max\{\frac{3}{2}\mathsf{bwd}(G), \gamma(G), 1\}.$$

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 $b : \mathsf{leaves}(Y) \cong E$ between the leaves of Y and the edges of G. A branch decomposition of a relational τ-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: $v \in \mathsf{ends}(A_e) \cap \mathsf{ends}(B_e)$.

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: $\mathsf{ord}(e) := |\mathsf{ends}(A_e) \cap \mathsf{ends}(B_e)|$.

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, $\mathsf{wd}(Y, b) := \max_{e \in \mathsf{edges}(Y)} \mathsf{ord}(e)$. The *branch width* of a hypergraph G is given by the min-max formula:

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

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 $l : V \to \{1, \ldots, n\}$.

- 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 $\mathsf{Rename}^n{}_{i \to j}$ 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* $\mathsf{Edge}^n{}_{i,j}$ 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.

$$
\begin{array}{rcll}
t & ::= & (G) & \text{if } G = \emptyset_1 \text{ or } G = \mathsf{v}_1 \\
& \mid & \mathsf{Rename}^n{}_{i\to j}(t') & \text{if } G = \mathsf{Rename}^n{}_{i\to j}(G') \text{ and } t' \in T_{G'} \\
& \mid & \mathsf{Edge}^n{}_{i,j}(t') & \text{if } G = \mathsf{Edge}^n{}_{i,j}(G') \text{ and } t' \in T_{G'} \\
& \mid & t_1 + t_2 & \text{if } G = G_1 + G_2 \text{ and } t_i \in T_{G_i}
\end{array}
$$

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

$$\mathsf{Rename}^2{}_{2\to 1}\mathsf{Edge}^2{}_{1,2}(\mathsf{Rename}^3{}_{3\to 2}\mathsf{Edge}^3{}_{2,3}(\mathsf{Rename}^4{}_{4\to 3}\mathsf{Edge}^4{}_{3,4}(\mathsf{v}_1 + \mathsf{v}_1 + \mathsf{v}_1 + \mathsf{v}_1))) \,,$$

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

$$\mathsf{Rename}^2{}_{2\to 1}\mathsf{Edge}^2{}_{1,2}(\mathsf{v}_1 + \mathsf{Rename}^2{}_{2\to 1}\mathsf{Edge}^2{}_{1,2}(\mathsf{v}_1 + \mathsf{Rename}^2{}_{2\to 1}\mathsf{Edge}^2{}_{1,2}(\mathsf{v}_1 + \mathsf{v}_1))) \,,$$

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, $\mathsf{w}(\mathsf{Rename}^n{}_{i\to j}) := n$, $\mathsf{w}(\mathsf{Edge}^n{}_{i,j}) := n$ and $\mathsf{w}(+) := 0$. The *width* of a clique decomposition t of G is the maximum cost of its operations.

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

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

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

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 $n + 1$ [GR00].

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.

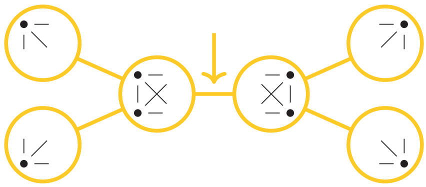
Theorem 2.39 ([OS06, Proposition 6.3])**.** *Rank width is equivalent to clique width. More precisely, for a graph G,*

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

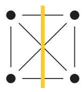
Rank decompositions are similar in spirit to branch decompositions, but, instead of partitioning the edges of a graph, they partition their vertices. A rank decomposition of a graph $G = (V, E)$ is a tree where the leaves are in bijection with the vertices V of the graph. If this tree had a root, a rank 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 $r\colon$ leaves$(Y) \to$ 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.



$$\mathrm{rk}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = 1$$

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

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

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

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 τ-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 $b : \mathbb{2}^{m \cdot n} \to \mathbb{2}$.

For all τ-structures $G_1, \dots, G_n$, the values of the formulae ψ_i^j on $G_1, \dots, G_n$ and the function b need to determine the value of ϕ 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 ϕ and the operation o need to be explicit, we will denote the list of formulae ψ_i^j by RedSeq(ϕ, o) and the boolean function b by RedBool(ϕ, 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 $\mathrm{Th}_{\mathcal{L}(\tau)}(o(G_1, \dots, G_n))$ of the τ-structure $o(G_1, \dots, G_n)$ in $\mathcal{L}(\tau)$ depends only on $\mathrm{Th}_{\mathcal{L}(\tau)}(G_1), \dots, \mathrm{Th}_{\mathcal{L}(\tau)}(G_n)$, for all τ-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 RedSeq(ϕ, o) and its associated formula RedBool(ϕ, 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 τ-structures is *$\mathcal{L}$-inductive* if there are

- a finite *generating* set $\mathcal{K}_0 \subseteq \mathcal{K}$ of τ-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 $\mathcal{K}_{n+1} := \{G\ \tau\text{-structure} : \exists G_1, \dots, G_k \in \mathcal{K}_n\ \exists o \in \mathcal{O}\ G = o(G_1, \dots, G_k)\}$ is the set of all the τ-structures A that are obtained by applying an operation $o \in \mathcal{O}$ to some τ-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 τ-structures, an *algebraic decomposition* of a τ-structure $G \in \mathcal{K}$ is a term $t \in T_G$ constructed from applications of operations $o \in \mathcal{O}$ to τ-structures G_0 in the generating set $\mathcal{K}_0$:

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

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, $\max_{i,j}\{\mathsf{w}(o_i), |V_j|\}$, where we fixed costs $\mathsf{w}(o_i)$ for operations o_i.

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

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

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

$$\begin{aligned} \mathsf{size}(t) \quad &:= \quad 1 && \text{if } t = (G)\,, \\ &\quad\;\; \mid \quad \mathsf{size}(t_1) + \cdots + \mathsf{size}(t_k) && \text{if } t = o(t_1, \ldots, t_k)\,. \end{aligned}$$

The *algebraic width* of a τ-structure G is the width of a best decomposition:

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

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* $\mathsf{Red}(\phi, \mathcal{O})$ 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 ϕ and
- is closed under taking o-reduction sequences $\mathsf{RedSeq}(-, o)$, 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 ϕ and $\mathcal{O}$ is a list

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

When the look-up table $\mathsf{Look}(\phi, \mathcal{O})$ 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 τ-structures with respect to a finite set $\mathcal{O}$ of effectively $\mathcal{L}$-smooth operations. Let $G \in \mathcal{K}$ be a τ-structure with a parse term d. Then, whether the formula ϕ holds in G, $G \models \phi$, can be decided in time linear in* $\mathsf{size}(d)$.

Proof. We precompute the look-up table $\mathsf{Look}(\phi, \mathcal{O})$ 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 $\mathsf{Check}(d, \phi)$ (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 $\mathsf{Look}(\phi, \mathcal{O})$. Looking up the information in $\mathsf{Look}(\phi, \mathcal{O})$ 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 $c : \mathbb{N} \to \mathbb{N}$. If the size of the decomposition is n and the maximum size of the substructures G_i is k, the computation takes $\mathcal{O}(c(k) \cdot n)$. □

Algorithm 1: Check(d, ϕ)

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

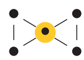
Definition 2.53. A relational τ-structure *with n constants*, for a natural number $n \in \mathbb{N}$, is a pair (G, c) of a structure G together with a function $c : \{1, \dots, n\} \to V$.

- The generating structures are the empty structure with no constants, $\emptyset$, and, for every relational symbol $R \in \tau$, the structure e_R with α_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* $\mathsf{Relab}^n{}_f$ of a relational structure (G, c) with n constants with a function $f : \{1, \dots, m\} \to \{1, \dots, n\}$, is the relational strcuture (G, fc) with m constants.
- The *fusion of constants i* and j, with $0 < i < j \le n+1$, on a relational structure (G, c) with $n+1$ constants gives a relational structure $\mathsf{Fuse}^n{}_{i,j}(G, c)$ 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 $R^{\mathsf{Fuse}^n{}_{i,j}(G)}$ 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 $\mathsf{Fuse}^n{}_{i,j}c$ is the reindexing of the function c as $\mathsf{Fuse}^n{}_{i,j}c(k) = c(k)$ if $k < j$ and $\mathsf{Fuse}^n{}_{i,j}c(k) = c(k+1)$ if $k \geq j$.
- The *addition of constant* i, $\mathsf{Vert}^n{}_i$, on a relational structure (G, c) with $n \geq i - 1$ constants is the relational structure $(G + \{v\}, c')$ with $n + 1$ constants $c' : n + 1 \to V + \{v\}$ 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, $\mathsf{w}(+) := 0$, $\mathsf{w}(\mathsf{Relab}^n{}_f) := n$, $\mathsf{w}(\mathsf{Fuse}^n{}_{i,j}) := n$ and $\mathsf{w}(\mathsf{Vert}^n{}_i) := n$, 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 $\mathsf{Fuse}^2{}_{1,2}(t + t)$, where t is a term for the 3-clique with one constant:

$$t = \mathsf{Fuse}^2{}_{1,2}\mathsf{Relab}^3{}_\iota\mathsf{Fuse}^4{}_{2,3}(\mathsf{e} + \mathsf{Relab}^3{}_\iota\mathsf{Fuse}^4{}_{2,3}(\mathsf{e} + \mathsf{e}))\ ,$$

that creates an edge at a time and joins its endpoints with the existing edges. The function $\iota : \{1,2\} \to \{1,2,3\}$ 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 τ-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:*

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

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 $B \in \mathsf{Mat}_{\mathbb{Z}_2}(k, n)$ 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* Recol_M of an n-labelled graph (G, B) by an n by m matrix $M \in \mathsf{Mat}_{\mathbb{Z}_2}(n, m)$ 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 $M \in \mathsf{Mat}_{\mathbb{Z}_2}(m, l)$, $N \in \mathsf{Mat}_{\mathbb{Z}_2}(n, l)$ and $P \in \mathsf{Mat}_{\mathbb{Z}_2}(m, n)$, is the l-labelled graph $(G +_P H, \left(\begin{smallmatrix} B \cdot M \\ C \cdot N \end{smallmatrix}\right))$, 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, $\mathsf{w}(\mathsf{Recol}_M) := n$ and $\mathsf{w}(+_{M,P,N}) := \max\{m, n\}$, and obtain a corresponding notion of width for decompositions.

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

$$\mathsf{Recol}_{\bullet}(\mathsf{v}_1 +_{1,1,1} \mathsf{Recol}_{\bullet}(\mathsf{v}_1 +_{1,1,1} \mathsf{Recol}_{\bullet}(\mathsf{v}_1 +_{1,1,1} \mathsf{v}_1)))\ ,$$

that creates one vertex at a time and connects it to all existing vertices. The recolouring by the matrix  $:= \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ 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:*

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

All these operations preserve monadic second order formulae. More precisely, the preservation theorem for disjoint union is known as Feferman-Vaugh-Mostowksi 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-Fraıssé 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 τ-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 ϕ in the monadic second order logic of relational τ-structures, the problem of checking ϕ 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 ϕ in the monadic second order logic of graphs, the problem of checking ϕ 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 $f : A \to B$ in a monoidal category C is a syntax tree that uses the composition $⨟$ and the monoidal product $\otimes$ in C as operations.

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

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 $f : 1 \to 2$ and $g : 2 \to 1$ be morphisms in a prop. A monoidal decomposition of $f ⨟ (f \otimes f) ⨟ (g \otimes g) ⨟ g$ 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.

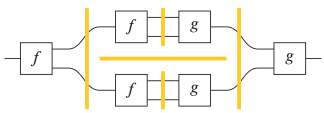


Figure 3.1: A monoidal decomposition represented with cuts in a string diagram.

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.

$$(f - ⨟_2 - (((f - ⨟_2 - g) - \otimes - (f - ⨟_2 - g)) - ⨟_2 - g)),$$

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* $\mathsf{w} : \mathrm{Mor}(\mathsf{C}) \to \mathbb{N}$[1] for a monoidal category C is a function that assigns a natural number to each morphism of C such that

1. $\mathsf{w}(f ⨟_B g) \leq \mathsf{w}(f) + \mathsf{w}(g) + \mathsf{w}(B)$, for $f : A \to B$ and $g : B \to C$; and
2. $\mathsf{w}(f \otimes g) \leq \mathsf{w}(f) + \mathsf{w}(g)$.

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

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 $f ⨟_B g$ 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.

[1]We indicate with Mor(C) the set of morphisms of a small category C. If the category C is essentially small, we can still define a weight function for 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 $\mathsf{w}(A) := \mathsf{w}(\mathbb{1}_A)$, 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 $f : A \to B$ in a monoidal category C with a weight function w is defined inductively below.

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

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

$$\mathsf{size}(d) := \begin{cases} 1 & \text{if } d = (f) \\ \mathsf{size}(d_1) + 1 + \mathsf{size}(d_2) & \text{if } d = (d_1 - \otimes - d_2) \text{ or } d = (d_1 - ;_C - d_2) \end{cases}$$

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 $f : A \to B$ in a monoidal category* C. *Then,*

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

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

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

□

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 $\mathsf{w}(f) = \mathsf{w}(g) = 2$, 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.

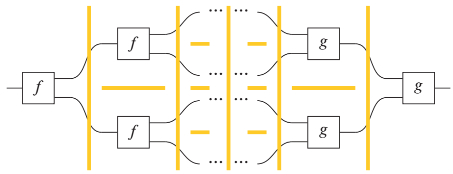
Definition 3.7. The *monoidal width* of a morphism f in a monoidal category C with a weight function w is the width of a cheapest decomposition:

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

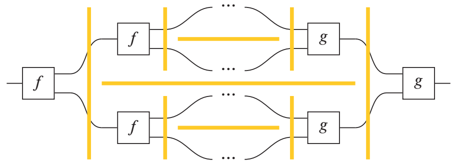
Example 3.8. With the morphisms f and g as in Example 3.2, we define a family of morphisms $h_n : 1 \to 1$ inductively:

- $h_0 := f \,⨟_2\, g$;
- $h_{n+1} := f \,⨟_2\, (h_n \otimes h_n) \,⨟_2\, g$.

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 $\mathsf{mwd}(h_n) \leq 2$ for any n.

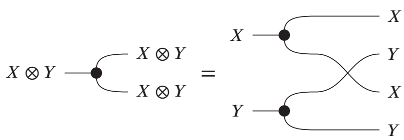


3.2 Categories with copy

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 C has *coherent copying* if there is a class of *copiable* objects $\Delta_{\mathsf{C}} \subseteq \mathsf{Obj}(\mathsf{C})$ such that

- $X, Y \in \Delta_{\mathsf{C}}$ iff $X \otimes Y \in \Delta_{\mathsf{C}}$;
- every object $X \in \Delta_{\mathsf{C}}$ is endowed with a *copy* morphism $-\!\!\bullet\!\!<_X : X \to X \otimes X$;
- the copy morphisms are *coherent*: for every $X, Y \in \Delta_{\mathsf{C}}$, $-\!\!\bullet\!\!<_{X \otimes Y} = (-\!\!\bullet\!\!<_X \otimes -\!\!\bullet\!\!<_Y) \,⨟\, (\mathbb{1}_X \otimes \sigma_{X,Y} \otimes \mathbb{1}_Y)$.



For props with coherent copy, we assume that the weight of copy morphisms, symmetries and identities is given by $\mathsf{w}(-\!\!\bullet\!\!<_X) := 2 \cdot \mathsf{w}(X)$, $\mathsf{w}(\sigma_{X,Y}) := \mathsf{w}(X) + \mathsf{w}(Y)$ and $\mathsf{w}(\mathbb{1}_X) := \mathsf{w}(X)$. 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: $-\!\!\bullet\!\!<_n := \langle \mathbb{1}_n, \mathbb{1}_n \rangle : n \to n + n$. 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 $\gamma_{n,m} := (-\!\!\bullet\!\!<_n \otimes \mathbb{1}_m) \,⨟\, (\mathbb{1}_n \otimes \sigma_{n,m}) : n + m \to n + m + n$ 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

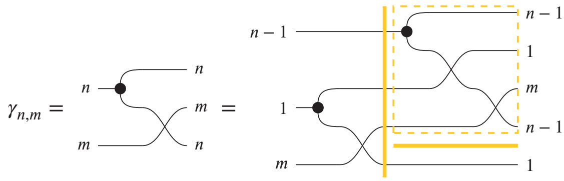


Figure 3.2: Decomposing copy morphisms.

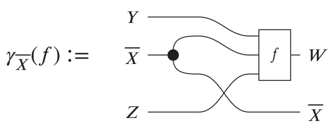
$-\!\!\bullet\!\!\!<_1$ and the symmetry $\sigma_{1,1}$, by cutting along at most $n+1+m$ wires:

$$\gamma_{n,m} = (\mathbb{1}_{n-1} \otimes ((-\!\!\bullet\!\!\!<_1 \otimes \mathbb{1}_1) \mathbin{⨟} (\mathbb{1}_1 \otimes \sigma_{1,1}))) \mathbin{⨟}_{n+1+m} (g_{n-1,m+1} \otimes \mathbb{1}_1).$$

By induction, we decompose $-\!\!\bullet\!\!\!<_n = \gamma_{n,0}$ cutting along only $n+1$ wires. In particular, this means that $\mathsf{mwd}(-\!\!\bullet\!\!\!<_n) \leq n+1$.

The following result generalises the reasoning in Example 3.10.

Lemma 3.11. *Let* C *be a symmetric monoidal category with coherent copying and* $d \in D_f$ *be a monoidal decomposition of a morphism* $f : Y \otimes \overline{X} \otimes Z \to W$*, with* $\overline{X} := X_1 \otimes \cdots \otimes X_n$*. Then we can construct a monoidal decomposition* $\mathcal{C}_{\overline{X}}(d)$ *of the morphism* $\gamma_{\overline{X}}(f) := (\mathbb{1}_Y \otimes -\!\!\bullet\!\!\!<_{\overline{X}} \otimes \mathbb{1}_Z) \mathbin{⨟} (\mathbb{1}_{Y\otimes\overline{X}} \otimes \sigma_{\overline{X},Z}) \mathbin{⨟} (f \otimes \mathbb{1}_{\overline{X}})$

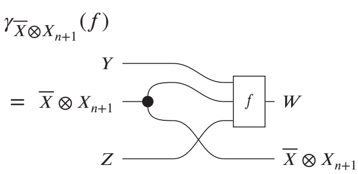


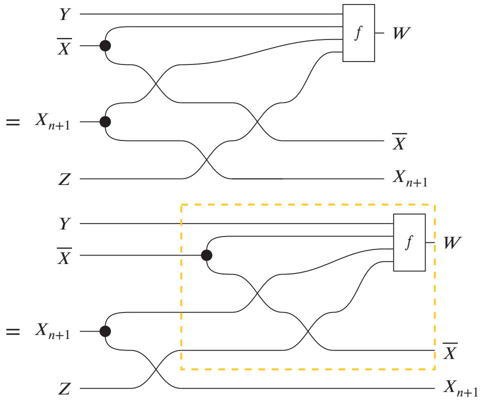
of bounded width:

$$\mathsf{wd}(\mathcal{C}_{\overline{X}}(d)) \leq \max\{\mathsf{wd}(d), \mathsf{w}(Y) + \mathsf{w}(Z) + (n+1) \cdot \max_{i=1,\ldots,n} \mathsf{w}(X_i)\}.$$

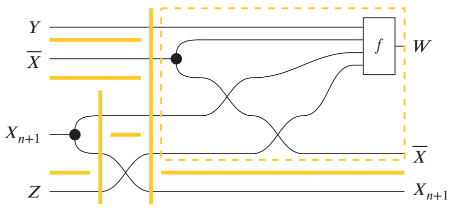
Proof. Proceed by induction on the number n of objects being copied. If $n = 0$, then we are done because we can keep the decomposition d: $\mathcal{C}_I(d) := d$.

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





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



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

$$
\begin{aligned}
&\mathsf{wd}(\mathcal{C}_{\overline{X} \otimes X_{n+1}}(d)) \\
&= \max\{\mathsf{w}(\mathbb{1}_{Y \otimes \overline{X}}), \mathsf{w}(-\!\!\bullet\!\!\!\subset_{X_{n+1}}), \mathsf{w}(\mathbb{1}_Z), \mathsf{w}(\mathbb{1}_{X_{n+1}}), \mathsf{w}(\sigma_{X_{n+1},Z}), \mathsf{wd}(\mathcal{C}_{\overline{X}}(d)), \\
&\qquad \mathsf{w}(Y \otimes \overline{X} \otimes Z \otimes X_{n+1}), \mathsf{w}(X_{n+1} \otimes Z \otimes X_{n+1})\} \\
&\leq \max\{\mathsf{w}(Y) + \mathsf{w}(Z) + \mathsf{w}(X_{n+1}) + \sum_{i=1}^{n+1} \mathsf{w}(X_i), \mathsf{wd}(\mathcal{C}_{\overline{X}}(d))\} \\
&\leq \max\{\mathsf{w}(Y) + \mathsf{w}(Z) + (n+2) \cdot \max_{i=1,\dots,n+1} \mathsf{w}(X_i), \mathsf{wd}(\mathcal{C}_{\overline{X}}(d))\} \\
&\leq \max\{\mathsf{w}(Y) + \mathsf{w}(Z) + (n+2) \cdot \max_{i=1,\dots,n+1} \mathsf{w}(X_i), \\
&\qquad \mathsf{wd}(d), \mathsf{w}(Y) + \mathsf{w}(X_{n+1} \otimes Z) + (n+1) \cdot \max_{i=1,\dots,n} \mathsf{w}(X_i)\} \\
&= \max\{\mathsf{w}(Y) + \mathsf{w}(Z) + (n+2) \cdot \max_{i=1,\dots,n+1} \mathsf{w}(X_i), \mathsf{wd}(d)\}
\end{aligned}
$$

□

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[2].

Example 3.12. The category Mat_R of matrices over a semiring R is a prop where the monoidal product is a biproduct. Its morphisms $n \to m$ are m rows by n columns matrices with entries in the semiring R and the biproduct of matrices, $A \oplus B := \left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$, 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.*

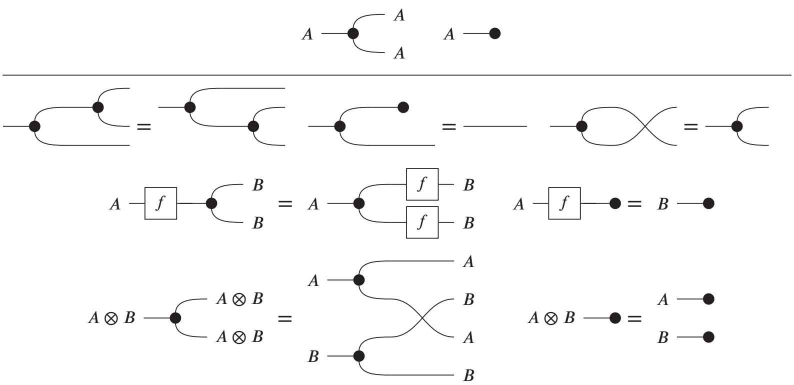


Figure 3.3: Structure and equations for a natural and uniform cocommutative comonoid.

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 $I \to I$ 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.

[2]We thank JS Lemay for suggesting to generalise this result for matrices to categories with biproducts.

Definition 3.14. For a prop P, define a weight function $\mathsf{w} : \mathcal{A} \to \mathbb{N}$ as $\mathsf{w}(g) := \max\{m, n\}$, for $g : n \to m$ in P. For a monoidal category 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 $X = X_1 \otimes \ldots \otimes X_k$, $|X| := k$. A weight function for C is $\mathsf{w} : \mathcal{A} \to \mathbb{N}$ as $\mathsf{w}(g) := \max\{|X|, |Y|\}$, for $g : X \to Y$ in C.

This definition satisfies the conditions for a weight function.

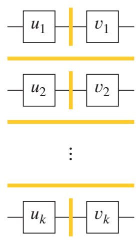
Lemma 3.15. *In a monoidal category whose objects are a unique factorisation monoid, the function* w *in Definition 3.14 satisfies the conditions for a weight function in Definition 3.3.*

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

$$\begin{aligned} &\mathsf{w}(f \,;\, g) \\ :=\ &\max\{l, n\} \\ \le\ &\max\{l, m, n\} + m \\ \le\ &\max\{l, m\} + \max\{m, n\} + m \\ =:\ &\mathsf{w}(f) + \mathsf{w}(g) + m \end{aligned} \qquad \begin{aligned} &\mathsf{w}(f \otimes f') \\ :=\ &\max\{l + l', m + m'\} \\ \le\ &\max\{l + l', l + m', l' + m, m + m'\} \\ =\ &\max\{l, m\} + \max\{l', m'\} \\ =:\ &\mathsf{w}(f) + \mathsf{w}(f') \end{aligned}$$

□

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 $f = f_1 \otimes \cdots \otimes f_k$ 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.

$f = f_1 \otimes \cdots \otimes f_k =$ 

Every factor f_i can be minimally split as a composition $f_i = u_i \,;_{r_i}\, v_i$ and give a decomposition of f of width at least $\max_{i=1,\ldots,k} r_i$. 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 $\max_{i=1,\ldots,k} r_i \le \mathsf{mwd}(f) \le \max_{i=1,\ldots,k} r_i + 1$.

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* $A : n \to m$ *in* $\mathsf{Mat}_{\mathbb{N}}$*. Then* $\min\{k \in \mathbb{N} : A = B \,;_k\, C\} = \mathsf{rk}(A)$.

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

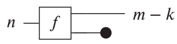
$$\mathsf{rk}(f) := \min\{k \in \mathbb{N} : f = g \,;_k\, h\}\,.$$

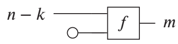
Similarly, for a morphism $f : X \to Y$ in a monoidal category C, whose objects are a unique factorisation monoid, its *rank* is its minimal vertical cut:

$$\mathsf{rk}(f) := \min\{k \in \mathbb{N} : f = g \,\text{;}_C\, h \wedge |C| = k\}\,.$$

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 $f : n \to m$ in a prop P where 0 is both initial and terminal and $d \in D_f$. Let $f_D := f \,\text{;}\, (\mathbb{1}_{m-k} \otimes \multimap\!\bullet_k)$ and $f_Z := (\mathbb{1}_{n-k} \otimes \circ\!-_k) \,\text{;}\, f$, with $k \leq m$ and $k \leq n$, respectively.*

$f_D :=$ 

, $f_Z :=$ 

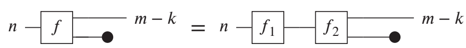
.

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

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 P^{op}. If the decomposition has only one node, $d = (f)$, then we define $\mathcal{D}(d) := (f_D)$ and obtain that

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

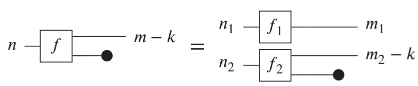
If the decomposition starts with a composition node, $d = (d_1 - \text{;}_j - d_2)$, then $f = f_1 \,\text{;}_j\, f_2$, where d_i is a monoidal decomposition of f_i.



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

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

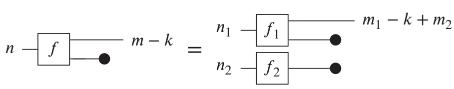
If the decomposition starts with a tensor node, $d = (d_1 - \otimes - d_2)$, then $f = f_1 \otimes f_2$, with d_i monoidal decomposition of $f_i : n_i \to m_i$. There are two possibilities: either $k \leq m_2$ or $k > m_2$. If $k \leq m_2$, then $f \,\text{;}\, (\mathbb{1}_{m-k} \otimes \multimap\!\bullet_k)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



By induction hypothesis, there is a monoidal decomposition $\mathcal{D}(d_2)$ of $f_2 \,\text{;}\, (\mathbb{1}_{m-k} \otimes \multimap\!\bullet_k)$ such that $\mathsf{wd}(\mathcal{D}(d_2)) \leq \mathsf{wd}(d_2)$. 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.

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

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



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

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

□

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

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

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

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

□

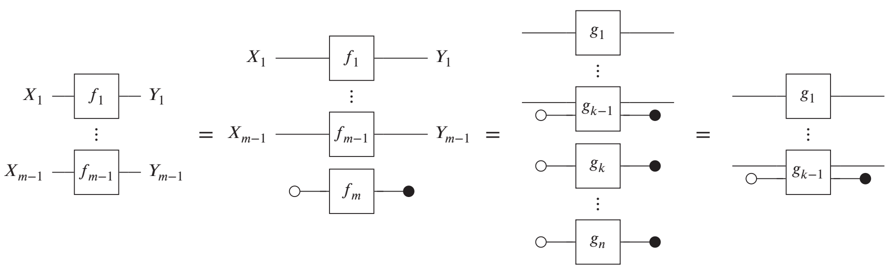
In monoidal categories where the monoidal unit is a zero object and the objects are a unique factorisation monoid, morphisms have a unique $\otimes$-decomposition.

Lemma 3.20. *Let* 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* C *has a unique $\otimes$-decomposition.*

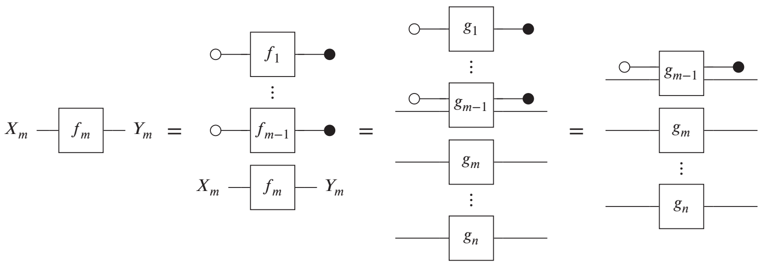
Proof. Suppose that $f : X \to Y$ has two $\otimes$-decompositions $f = f_1 \otimes \cdots \otimes f_m = g_1 \otimes \cdots \otimes g_n$ with $f_i : X_i \to Y_i$ and $g_j : Z_j \to W_j$ that are non $\otimes$-decomposables. Suppose $m \leq n$ and proceed by induction on m.

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

For the induction step, suppose that $\bar{f} := f_1 \otimes \ldots \otimes f_{m-1}$ has a unique $\otimes$-decomposition. Let $A_1 \otimes \ldots \otimes A_\alpha$ and $B_1 \otimes \ldots \otimes B_\beta$ be the unique $\otimes$-decompositions of $X_1 \otimes \ldots \otimes X_m = Z_1 \otimes \ldots \otimes Z_n$ and $Y_1 \otimes \ldots \otimes Y_m = W_1 \otimes \ldots \otimes W_n$, respectively. Then, there are $x \leq \alpha$ and $y \leq \beta$ such that $A_1 \otimes \ldots \otimes A_x = X_1 \otimes \ldots \otimes X_{m-1}$ and $B_1 \otimes \ldots \otimes B_y = Y_1 \otimes \ldots \otimes Y_{m-1}$. Then, we can rewrite $\bar{f}$ in terms of g_is, 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 $f_{m-1} = (\mathbb{1} \otimes \circ\!-) \,;\, g_k \,;\, (\mathbb{1} \otimes -\!\bullet)$ because g_i are not $\otimes$-decomposable. Then, we can express f_m in terms of $g_m, \ldots, g_n$:



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

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

Corollary 3.21. *Let $f = f_1 \otimes \cdots \otimes f_k$ 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 $d = (d_1 - \otimes - (d_2 - \otimes - \cdots\, d_k))$, 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 $\mathsf{mwd}(f) \geq \max_i\{\mathsf{rk}(f_i)\}$. For the upper bound, we show that every f_i can be decomposed with width at most $\mathsf{rk}(f_i) + 1$. 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.

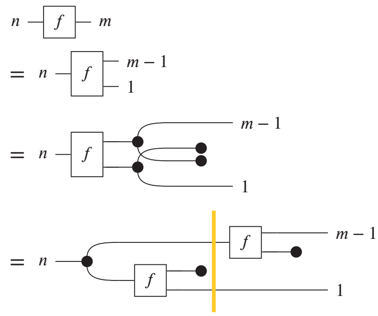
Proposition 3.22. *The monoidal width of a morphism $f : n \to m$ in a bicartesian prop* P *is bounded by its domain and codomain:* $\mathsf{mwd}(f) \leq \min\{m, n\} + 1$.

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

For the induction steps, suppose that the statement is true for any $f' : n' \to m'$ 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 $-\!\!\bullet\!\!\!<_{n+1}$ is uniform and morphisms are copiable because P is cartesian by hypothesis.



This corresponds to $f = -\!\!\bullet\!\!\!<_n \,⨟\, (\mathbb{1}_n \otimes h_1) ⨟_{n+1} (h_2 \otimes \mathbb{1}_1)$, where $h_1 := f ⨟ (-\!\!\bullet_{m-1} \otimes \mathbb{1}_1) : n \to 1$ and $h_2 := f ⨟ (\mathbb{1}_{m-1} \otimes -\!\!\bullet_1) : n \to m-1$.
Then, $\mathsf{mwd}(f) \leq \max\{\mathsf{mwd}(-\!\!\bullet\!\!\!<_n ⨟ (\mathbb{1}_n \otimes h_1)), n+1, \mathsf{mwd}(h_2 \otimes \mathbb{1}_1)\}$. 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 $h_1 : n \to 1$ and $1, n < k$ and we apply Lemma 3.11. For the second morphism, we apply the induction hypothesis because $h_2 : n \to m-1$ and $n, m-1 < k$.

$\mathsf{mwd}(-\!\!\bullet\!\!\!<_n ⨟ (\mathbb{1}_n \otimes h_1))$
$\leq$ (by Lemma 3.11)
$\max\{\mathsf{mwd}(h_1), n+1\}$
$\leq$ (by induction hypothesis)
$\max\{\min\{n, 1\} + 1, n+1\}$
$=$ (because $0 < n$)
$n+1$

$\mathsf{mwd}(h_2 \otimes \mathbb{1}_1)$
$=$ (by Definition 3.4)
$\mathsf{mwd}(h_2)$
$\leq$ (by induction hypothesis)
$\min\{n, m-1\} + 1$
$=$ (because $n \leq m-1$)
$n+1$

Then, $\mathsf{mwd}(f) \leq n+1 = \min\{m, n\} + 1$ because $n < m$.
2. If $0 < m < n = k$, we can apply Case 1 to P^{op} with the same assumptions on the set of atoms because P^{op} is also bicartesian. We obtain that $\mathsf{mwd}(f) \leq m+1 = \min\{m, n\} + 1$ 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 $\mathsf{mwd}(h_1)$ and $\mathsf{mwd}(h_2)$, one applies Case 2 (or Case 1). Then, $\mathsf{mwd}(f) \leq m+1 = \min\{m, n\} + 1$ because $m = n$.

□

Lemma 3.23. *The monoidal width of a morphism $f : n \to m$ in a bicartesian prop P is bounded by its rank:* $\mathsf{mwd} f \leq \mathsf{rk}(f) + 1$. *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* $\mathsf{mwd} f \geq \mathsf{rk}(f)$.

Proof. For the first inequality, observe that there is a monoidal decomposition $d = ((g) - ⨟_k - (h))$ of f attaining the minimum of $k = \mathsf{rk}(f)$. 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.

$\mathsf{wd}(d_1)$
$\leq \min\{n, k\} + 1$

$\mathsf{wd}(d_2)$
$\leq \min\{k, m\} + 1$

$= k + 1$ $\qquad$ $= k + 1$

By definition of monoidal width and weight function,

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

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, $(d_1 - \fatsemi_k - d_2)$, or be a leaf, (f). Then,

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

□

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 $f = f_1 \otimes \ldots \otimes f_k$ and its monoidal width is, up to 1, the maximum of the ranks of its factors, $\max_{i=1,\ldots,k} \mathsf{rk}(f_i) \leq \mathsf{mwd}(f) \leq \max_{i=1,\ldots,k} \mathsf{rk}(f_i) + 1$.*

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

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

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 τ, a *relational τ-structure* G is a finite set V with an α_R-ary relation $R^G \subseteq V^{\alpha_R}$ for each relational symbol R of arity α_R in the signature τ. A *morphism* of relational τ-structures $h \colon G \to H$ is a function $h \colon V_G \to V_H$ 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 τ-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 $\mathbf{T} \colon$ FinSet $\to$ FinSet defined by the pullback below.

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

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 α_R of elements $v_1, \dots, v_{\alpha_R} \in V$. A relational structure is a function $G \colon E_G \to \mathbf{T}(V_G)$ and a morphism $h \colon G \to H$ is a pair of functions $h_E \colon E_G \to E_H$ and $h_V \colon V_G \to V_H$ such that $G \mathbin{⨟} \mathbf{T}(h_V) = h_E \mathbin{⨟} H$.

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

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

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

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 C is a pair of morphisms, the *legs* $f \colon X \to E$ and $g \colon Y \to E$, in 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* C *has finite colimits, cospans form a symmetric monoidal category* Cospan(C) *whose objects are the objects of* C *and morphisms are cospans in* C*. More precisely, a morphism $X \to Y$ in* Cospan(C) *is an equivalence class of cospans $f \colon X \to E \leftarrow Y : g$, up to isomorphism of the head of the cospan. The composition of $f \colon X \to E \leftarrow Y : g$ and $h \colon Y \to F \leftarrow Z : l$ 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.

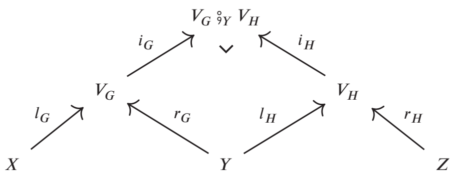
Proposition 4.6. *The category* Struct$_\tau$ *has all finite colimits.*

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

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

Explicitly, the composition of two morphisms $l_G\colon X \to V_G \leftarrow Y : r_G$ and $l_H\colon Y \to V_H \leftarrow Z : r_H$ in sStruct$_\tau$ is the morphism $l\colon X \to V_G \,{}^{\circ}_{9}{}_Y\, V_H \leftarrow Z : r$ defined by the pushout of r_G and l_H.



The apex of the cospan, $V_G \,;_Y\, V_H$, 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: $l := l_G ; i_G$ and $r := r_H ; i_H$. The monoidal product of two morphisms $l\colon X \to V \leftarrow Y : r$ and $l'\colon X' \to V' \leftarrow Y' : r'$ is their component-wise coproduct: $l + l'\colon X + X' \to V + V' \leftarrow Y + Y' : r + r'$.

Chapter 5 is dedicated to showing that monoidal width in the category 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 Cospan(UHGraph)$_*$ has sets as objects and discrete cospans of hypergraphs as morphisms. It is equivalent to the category sStruct$_{\tau_{hyp}}$ of discrete cospans of relational structures on the relational signature τ_{hyp} for hypergraphs.

A syntax for relational structures

The skeleton of the category Cospan(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 τ.

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

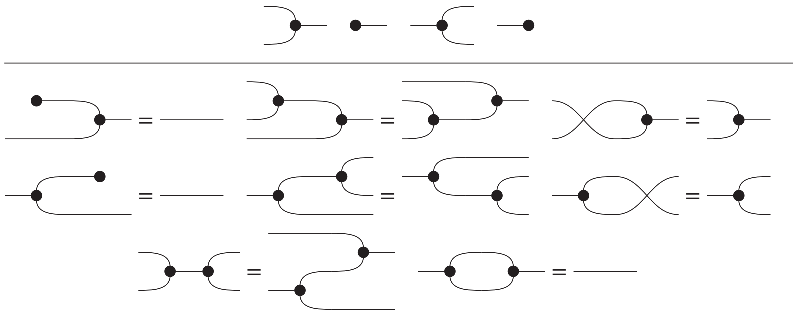


Figure 4.1: Generators and equations for a special Frobenius monoid.

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

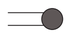
The prop of relational structures with sources is obtained by freely adding a generator $\mathsf{e}_R : \alpha_R \to 0$ for each $(R, \alpha_R) \in \tau$ to the prop sFrob.

Definition 4.11. Given a relational signature τ, the category LHedge_τ is the free prop generated by a "labelled hyperedge" generator $\mathsf{e}_R : \alpha_R \to 0$ for every relational symbol R of arity α_R in the signature τ (Figure 4.2).

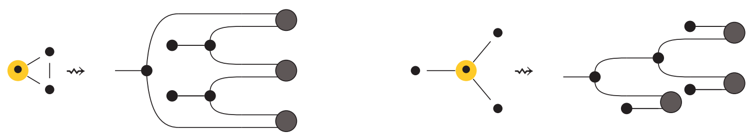
for every $(R, \alpha_R) \in \tau$

Figure 4.2: The labelled hyperedge generators.

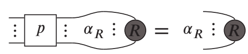
Definition 4.12. For a relational signature τ, the prop $\mathsf{sFrob}_\tau := \mathsf{sFrob} + \mathsf{LHedge}_\tau$ is the coproduct of the prop sFrob generated by a special Frobenius monoid and the prop LHedge_τ generated by the labelled hyperedges in τ.

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

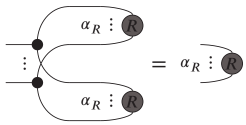
Example 4.13. The 3-clique with one source and the 3-star with one source are morphisms $1 \to 0$ in $\mathsf{sFrob}_{\tau_{gr}}$.



Remark 4.14. We can impose additional equations to sFrob_τ to constrain the behaviour of some relational symbols. For a symmetric relational symbol R, we impose that $p \,;\, \mathsf{e}_R = \mathsf{e}_R$, for every permutation p of the α_R inputs of e_R.



If we want to impose that there may not be parallel edges of the same type R, we add that α_R ⨟ $(\mathsf{e}_R \otimes \mathsf{e}_R) = \mathsf{e}_R$.



The prop sFrob_τ 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* $\mathsf{sStruct}_\tau$ *of* τ*-structures with sources is isomorphic to the free special Frobenius prop* sFrob_τ *on the signature* τ.

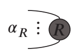
The operations for tree width

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 sFrob_τ of relational structures with sources. This correspondence defines inductively a function from structures with n constants to morphisms of type $n \to 0$ in sFrob_τ, which maps a structure (G, c) with n constants to the morphism $g : n \to 0$ in sFrob_τ that corresponds to the discrete cospan of structures $g = c : n \to G \leftarrow 0 : {¡}$[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.

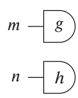
The generating structures of the algebraic tree decompositions correspond to specific morphisms in sFrob_τ. The empty structure with no constants $\emptyset$ is the identity morphism on the monoidal unit $\mathbb{1}_I$, and the structure e_R with α_R constants is the generator $\mathsf{e}_R : \alpha_R \to 0$.

$\emptyset \mapsto$ and $\mathsf{e}_R \mapsto$



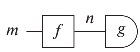
The operations are derived from the categorical structure. The disjoint union $(G, c) + (H, d)$ of structures (G, c) with m constants and (H, d) with n constants is their monoidal product as morphisms $g \otimes h : m + n \to 0$.

$(G, c) + (H, d) \mapsto$



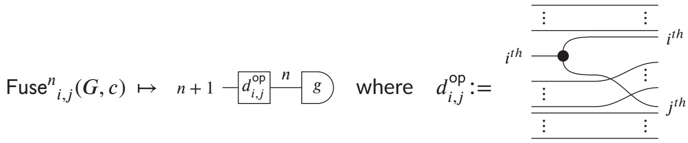
The redefinition of constants $\mathsf{Relab}^n{}_f(G, c)$ by a function f is obtained by precomposing the corresponding morphism g with the cospan $f : m \to n \leftarrow n : \mathbb{1}$. This cospan is composed only of the monoid operations, i.e. it is covariantly lifted from the function $f : m \to n$.

$\mathsf{Relab}^n{}_f(G, c) \mapsto$

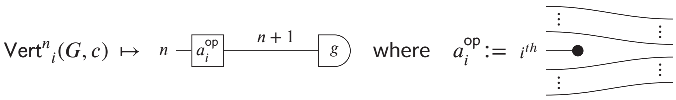


[1] We indicate with $¡_A : 0 \to A$ the unique morphism from the initial object 0 to an object A. Similarly, we indicate with $!_A : A \to 1$ the unique morphism from an object A to the terminal object 1

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



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



The operations of redefinition, fusion and addition of constants together are as expressive as the operation of precomposition with edge-less morphisms in sFrob_τ. In fact, these operations can construct all morphisms $n \to 0$ 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* $\mathsf{Mat}_\mathbb{N}$ is the prop whose morphisms $n \to m$ are m by n matrices. Composition is the usual product of matrices and the monoidal product is the biproduct of matrices $A \oplus B := \left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$.

A syntax for matrices

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

Definition 4.17. The prop 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].

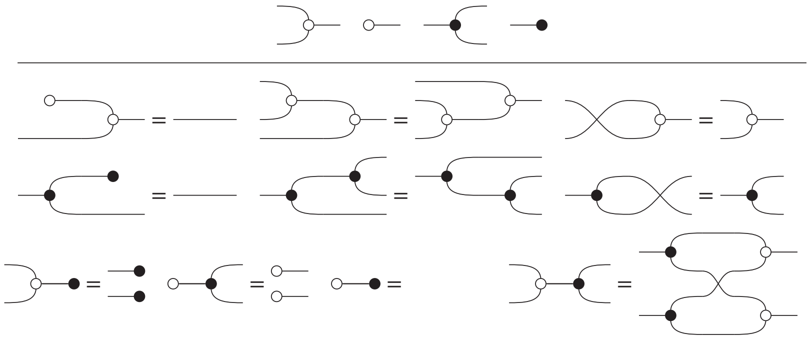
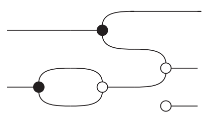


Figure 4.3: Generators and equations of a bialgebra.

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

Every morphism $f : n \to m$ in Bialg corresponds to a matrix $A = \mathbf{Mat}(f) \in \mathsf{Mat}_{\mathbb{N}}(m, n)$: 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 $A = \begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 0 \end{pmatrix} \in \mathsf{Mat}_{\mathbb{N}}(3, 2)$ corresponds to the morphism $a : 2 \to 3$ 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.

$a =$ 

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

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

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, MGraph, is the coproduct of a prop of adjacency matrices, MAdj, and a prop of bounded permutations, boundP; Theorem 4.39 and Proposition 4.41 give equivalent syntactic descriptions of adjacency matrices, Adj, and bounded permutations, Vert, based on the bialgebra characterisation of matrices; finally, the syntactic presentation of graphs with dangling edges is defined as their coproduct, BGraph := Adj + Vert.

$$\begin{array}{ccccc} \mathsf{MAdj} & \xrightarrow{\iota_1} & \mathsf{MGraph} & \xleftarrow{\iota_2} & \mathsf{boundP} \\ {\scriptstyle\text{Theorem 4.39}}\downarrow & & \downarrow{\scriptstyle\text{Theorem 4.44}} & & \downarrow{\scriptstyle\text{Proposition 4.41}} \\ \mathsf{Adj} & \longrightarrow & \mathsf{BGraph} & \longleftarrow & \mathsf{Vert} \end{array}$$

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 $G \in \mathsf{Mat}_{\mathbb{N}}(m, m)$ 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 $0 \to m$ 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* MAdj *where morphisms $\alpha : n \to m$ are pairs $\alpha = (B, [G])$ of an m by n matrix $B \in \mathsf{Mat}_{\mathbb{N}}(m, n)$ and an m by m adjacency matrix $[G]$.*

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

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

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

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

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

$$
\begin{aligned}
&(B, [G]) \,⨾\, (\mathbb{1}_m, [\mathbb{0}]) \\
&= (\mathbb{1}_m \cdot B, \left[\mathbb{1}_m \cdot G \cdot \mathbb{1}_m^\top + \mathbb{0}\right]) \\
&= (B, [G])
\end{aligned}
\qquad
\begin{aligned}
&(\mathbb{1}_n, [\mathbb{0}]) \,⨾\, (B, [G]) \\
&= (B \cdot \mathbb{1}_n, \left[B \cdot \mathbb{0} \cdot B^\top + G\right]) \\
&= (B, [G])
\end{aligned}
$$

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

$$
\begin{aligned}
&((B, [G]) \otimes (B', \left[G'\right])) \,⨾\, ((C, [H]) \otimes (C', \left[H'\right])) \\
&= (B \oplus B', \left[G \oplus G'\right]) \,⨾\, (C \oplus C', \left[H \oplus H'\right]) \\
&= ((C \oplus C')(B \oplus B'), \left[(C \oplus C')(G \oplus G')(C \oplus C')^\top + (H \oplus H')\right]) \\
&= ((CB) \oplus (C'B'), \left[(CGC^\top) \oplus (C'G'C'^\top) + (H \oplus H')\right]) \\
&= ((CB) \oplus (C'B'), \left[(CGC^\top + H) \oplus (C'G'(C')^\top + H')\right]) \\
&= (CB, \left[CGC^\top + H\right]) \otimes (C'B', \left[C'G'(C')^\top + H'\right]) \\
&= ((B, [G]) \,⨾\, (C, [H])) \otimes ((B', \left[G'\right]) \,⨾\, (C', \left[H'\right]))
\end{aligned}
$$

The monoidal product preserves identities.

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

The monoidal product is associative and unital because the objects are natural numbers and the monoidal product is addition. □

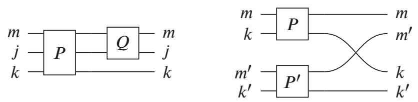
The ordering of vertices in a graph is immaterial, but adjacency matrices fix one. Graphs are adjacency matrices where the vertices can be arbitrarily permuted, so they are obtained by adding to the prop of adjacency matrices the possibility of permuting some of the wires, those connected to the vertices. 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 $P \in \mathsf{Mat}_{\mathbb{N}}(m + k, m + k)$. Two bounded permutations $p = (k, P)$ and $q = (k, Q)$ are equivalent if there is a permutation $\sigma \in \mathsf{Mat}_{\mathbb{N}}(k, k)$ such that $P = \begin{pmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{pmatrix} \cdot Q$.

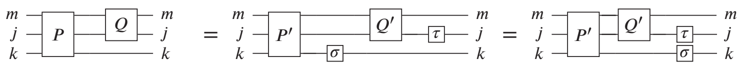
In a bounded permutation (k, P), the number k gives the number of outputs that are "bounded" and can, thus, be permuted without changing the morphism. Bounded permutations are the morphisms of a prop.

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

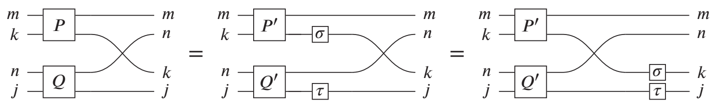
Proof. The composition of two bounded permutations $(k, P)\colon m+j+k \to m+j$ and $(j, Q)\colon m+j \to m$ is defined as $(k, P) ⨟ (j, Q) := (k+j, (Q \oplus \mathbb{1}_k) \cdot P)$, and the identity morphism on m is $(0, \mathbb{1}_m)\colon m \to m$. The monoidal product on objects is addition, the monoidal unit is 0 and, for two bounded permutations $(k, P)\colon m+k \to m$ and $(k', P')\colon m'+k' \to m'$, 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,



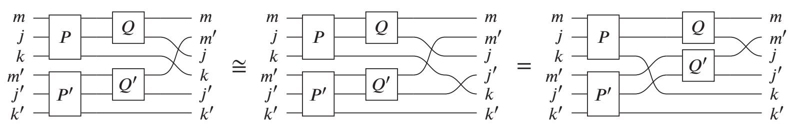
With this, it easy to see that composition is associative and unital. Composition is well-defined because, if $(k, P) = (k, P')$ and $(j, Q) = (j, Q')$, then $P = (\mathbb{1}_{m+j} \oplus \sigma) \cdot P'$, $Q = (\mathbb{1}_m \oplus \tau) \cdot Q'$,



and $(\mathbb{1}_k \oplus Q') \cdot P' = (\mathbb{1}_m \oplus \tau \oplus \sigma) \cdot ((\mathbb{1}_k \oplus Q') \cdot P')$. So $(k, P) ⨟ (j, Q) = (k, P') ⨟ (j, Q')$. 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: $((k, P) \otimes (k', P')) ⨟ ((j, Q) \otimes (j', Q')) = ((k, P) ⨟ (j, Q)) \otimes ((k', P') ⨟ (j', Q'))$ 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. □

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 MGraph of graphs with dangling edges is the coproduct of MAdj and boundP. Graphs with dangling edges have three connectivity points: the left and right boudaries, 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:

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

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

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

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

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* $g \colon n \to m$ *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 $g \colon n \to m$ and $h \colon m \to l$, with $g = \left([G], L_g, R_g, P_g, \left[S_g\right]\right)$ and $h = \left([H], L_h, R_h, P_h, \left[S_h\right]\right)$, their composition $g \mathbin{⨟} h \colon n \to l$ is

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

Composition is associative.

$$\begin{aligned}
&(f \mathbin{⨟} g) \mathbin{⨟} h \\
&= \left(\left[\begin{pmatrix} F & R_f L_g^{\top} \\ \mathbb{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, \left[S_g + P_g S_f P_g^{\top}\right]\right) \mathbin{⨟} h \\
&= \left(\left[\begin{pmatrix} F & R_f L_g^{\top} & R_f P_g^{\top} L_h^{\top} \\ \mathbb{0} & G + L_g S_f L_g^{\top} & (R_g + L_g (S_f + S_f^{\top}) P_g^{\top}) L_h^{\top} \\ \mathbb{0} & \mathbb{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 \\ L_h P_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. \\
&\qquad \left. P_h P_g P_f, \left[S_h + P_h (S_g + P_g S_f P_g^{\top}) P_h^{\top}\right]\right) \\
&= \left(\left[\begin{pmatrix} F & R_f L_g^{\top} & R_f P_g^{\top} L_h^{\top} \\ \mathbb{0} & G + L_g S_f L_g^{\top} & (R_g + L_g S_f P_g^{\top}) L_h^{\top} \\ \mathbb{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 \\ L_h P_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. \\
&\qquad \left. P_h P_g P_f, \left[S_h + P_h S_g P_h^{\top} + P_h P_g S_f P_g^{\top} P_h^{\top}\right]\right) \\
&= f \mathbin{⨟} \left(\left[\begin{pmatrix} G & R_g L_h^{\top} \\ \mathbb{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, \left[S_h + P_h S_g P_h^{\top}\right]\right) \\
&= f \mathbin{⨟} (g \mathbin{⨟} h)
\end{aligned}$$

Composition is unital.

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

$$
\begin{aligned}
&= \left([G], L_g, R_g, P_g, \left[S_g\right]\right) \\
&= g
\end{aligned}
$$

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

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

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

The monoidal product is functorial.

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

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

where $\tau = \left(\begin{smallmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{smallmatrix} \right)$ permutes the order of the vertices. □

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* $\iota_1 : \mathsf{MAdj} \to \mathsf{MGraph}$ *and* $\iota_2 : \mathsf{boundP} \to \mathsf{MGraph}$.

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

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

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

$$
\begin{aligned}
&\iota_1(B, [G]) \fatsemi \iota_1(C, [H]) \\
&:= ([()], !, !, B, [G]) \fatsemi ([()], !, !, C, [H]) \\
&= \left([()], !, !, CB, \left[H + CGC^\top\right]\right) \\
&=: \iota_1(CB, \left[H + CGC^\top\right]) \\
&= \iota_1((B, [G]) \fatsemi (C, [H]))
\end{aligned}
$$

$$
\begin{aligned}
&\iota_2(k, P) \fatsemi \iota_2(j, Q) \\
&:= ([\mathbb{0}_k], P_2, \mathbb{0}, P_1, [\mathbb{0}]) \fatsemi ([\mathbb{0}_j], Q_2, \mathbb{0}, Q_1, [\mathbb{0}]) \\
&= \left([\mathbb{0}_{k+j}], \left(\begin{smallmatrix} P_2 \\ Q_2 P_1 \end{smallmatrix} \right), \mathbb{0}, Q_1 P_1, [\mathbb{0}] \right) \\
&\cong \left([\mathbb{0}_{k+j}], \left(\begin{smallmatrix} Q_2 P_1 \\ P_2 \end{smallmatrix} \right), \mathbb{0}, Q_1 P_1, [\mathbb{0}] \right) \\
&=: \iota_2\left(k + j, \left(\begin{smallmatrix} Q_1 P_1 \\ Q_2 P_1 \\ P_2 \end{smallmatrix} \right)\right) \\
&= \iota_2\left(k + j, \left(\begin{smallmatrix} Q \\ \mathbb{1}_k \end{smallmatrix} \right) P\right) \\
&= \iota_2((k, P) \fatsemi (j, Q))
\end{aligned}
$$

They respect identities.

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

They respect the monoidal product.

$$
\begin{aligned}
&\iota_1(B, [G]) \otimes \iota_1(B', [G']) \\
&:= ([()], !, !, B, [G]) \otimes ([()], !, !, B', [G']) \\
&= ([()], !, !, B \oplus B', [G \oplus G']) \\
&=: \iota_1(B \oplus B', [G \oplus G'])
\end{aligned}
$$

$$
\begin{aligned}
&\iota_2(k, P) \otimes \iota_2(k', P') \\
&:= ([\mathbb{0}_k], P_2, \mathbb{0}, P_1, [\mathbb{0}]) \otimes ([\mathbb{0}_{k'}], P'_2, \mathbb{0}, P'_1, [\mathbb{0}]) \\
&= ([\mathbb{0}_{k+k'}], P_2 \oplus P'_2, \mathbb{0}, P_1 \oplus P'_1, [\mathbb{0}]) \\
&=: \iota_2\left(k + k', \left(\begin{smallmatrix} P_1 \oplus P'_1 \\ P_2 \oplus P'_2 \end{smallmatrix} \right)\right)
\end{aligned}
$$

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

□

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

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

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

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

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

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

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

Then, we can rewrite the components of g in terms of P_τ, B and T.

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

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

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

$$= P_\tau B \qquad\qquad = \left[\begin{pmatrix} P_1TP_1^\top & P_1TP_2^\top \\ P_2TP_1^\top & P_2TP_2^\top \end{pmatrix}\right] = \left[P_\tau T P_\tau^\top\right]$$

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

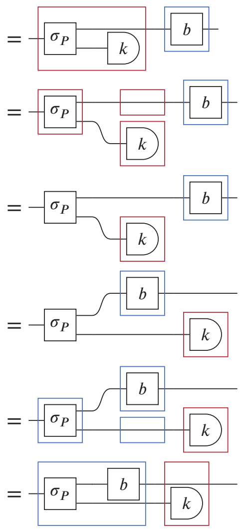
By Theorem 2.16, this result means that MGraph is a composite prop. The next result ensures that MGraph is, in particular, the coproduct of MAdj and boundP.

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

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

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 σ_P the permutation in MAdj, boundP or P corresponding to the permutation matrix P.

$$\mathbf{v}(k, P) \fatsemi \mathbf{a}(B, [S])$$
$$= \mathbf{v}(\sigma_P \fatsemi (\mathbb{1}_n \otimes (k, \mathbb{1}_k))) \fatsemi \mathbf{a}(B, [S])$$



$$= \mathbf{a}(\sigma_P \fatsemi ((B, [S]) \otimes \mathbb{1}_k)) \fatsemi \mathbf{v}(\mathbb{1}_m \otimes (k, \mathbb{1}_k))$$
$$= \mathbf{a}((B \oplus \mathbb{1}_k)P, \left[S \oplus \mathbb{0}_k\right]) \fatsemi \mathbf{v}(k, \mathbb{1}_{m+k})$$

□

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:* MGraph $\cong$ MAdj + 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 MGraph to obtain that it is the composition of MAdj and boundP via a distributive law $\lambda \colon (\iota_2(v) \mid \iota_1(a)) \mapsto (\iota_1(\hat{a}) \mid \iota_2(\hat{v}))$. In particular, for any two prop morphisms $\mathbf{a} \colon$ MAdj $\to$ P and $\mathbf{v} \colon$ boundP $\to$ P such that $\mathbf{v}(v) \,⨾\, \mathbf{a}(a) = \mathbf{a}(\hat{a}) \,⨾\, \mathbf{v}(\hat{v})$, there is a unique prop morphism $\mathbf{h} \colon$ MGraph $\to$ P such that $\mathbf{a} = \iota_1 \,⨾\, \mathbf{h}$ and $\mathbf{v} = \iota_2 \,⨾\, \mathbf{h}$. By Lemma 4.29, any two prop morphisms $\mathbf{a} \colon$ MAdj $\to$ P and $\mathbf{v} \colon$ boundP $\to$ P satisfy $\mathbf{v}(v) \,⨾\, \mathbf{a}(a) = \mathbf{a}(\hat{a}) \,⨾\, \mathbf{v}(\hat{v})$, which means that any two such morphisms define a unique prop morphism $\mathbf{h} \colon$ MGraph $\to$ P such that $\mathbf{a} = \iota_1 \,⨾\, \mathbf{h}$ and $\mathbf{v} = \iota_2 \,⨾\, \mathbf{h}$. This is equivalent to say that MGraph satisfies the universal property of the coproduct of MAdj and boundP. □

A syntax for graphs with dangling edges

We give a syntactic presentation of graphs with dangling edges by giving syntactic presentations of its components MAdj and 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 $\subset \colon 0 \to 2$ (Figure 4.4) that captures the equivalence relation of adjacency matrices (Lemma 4.36). Theorem 4.39 shows that Adj is a syntactic presentation of MAdj.

Definition 4.31. The prop Adj is presented by the generators and equations in Figures 4.3 and 4.4.

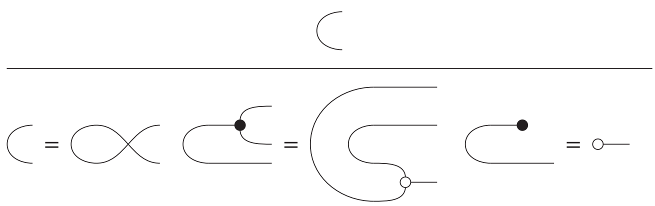
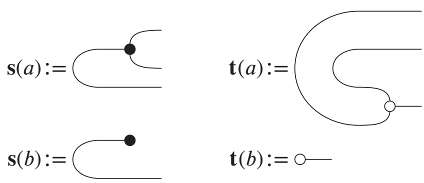


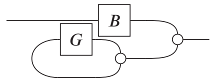
Figure 4.4: Additional generator and equations for the prop of adjacency matrices (Figure 4.3 contains the rest of generators and equations).

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



(4.1)

The isomorphism between the props Adj and MAdj is proven in [CS15, Theorem 4.2] by defining a prop morphism Adj $\to$ MAdj inductively and showing that it is an isomorphism. We rely on the same arguments but give a slightly different proof. We show that MAdj also satisfies the universal property of the coequaliser of $\mathbf{s}$ and $\mathbf{t}$. The isomorphism $\phi\colon$ MAdj $\to$ Adj defined in Theorem 4.39 captures the normal form of morphisms in Adj.

$$\phi\colon (B,[G]) \mapsto$$  (4.2)

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

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

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

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

$$\begin{aligned}&\mathbf{j}(A) ⨟ \mathbf{j}(B)\\ &:= (A,[\mathbb{0}]) ⨟ (B,[\mathbb{0}])\\ &:= (BA,[B\mathbb{0}B^{\top}+\mathbb{0}])\\ &= (BA,[\mathbb{0}])\\ &=: \mathbf{j}(A ⨟ B)\end{aligned} \qquad \begin{aligned}&\mathbf{j}(\mathbb{1}_n)\\ &:= (\mathbb{1}_n,[\mathbb{0}_n])\\ &= \mathbb{1}_n\end{aligned} \qquad \begin{aligned}&\mathbf{j}(A)\otimes\mathbf{j}(A')\\ &:= (A,[\mathbb{0}])\otimes(A',[\mathbb{0}])\\ &:= (A\oplus A',[\mathbb{0}\oplus\mathbb{0}])\\ &= (A\oplus A',[\mathbb{0}])\\ &=: \mathbf{j}(A\oplus A')\end{aligned}$$

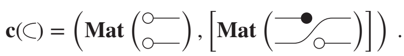
□

There is a morphism in MAdj that behaves like the cup ⊂.

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

$$\mathbf{c}(\subset) := \left(\mathrm{i}_2, \left[\begin{pmatrix}0&1\\0&0\end{pmatrix}\right]\right).$$

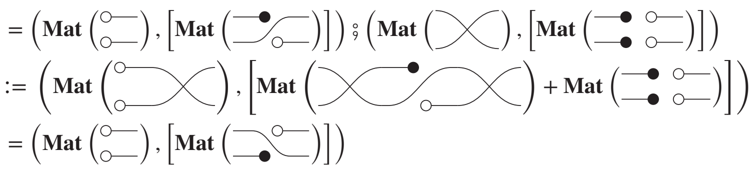
Proof. We define the mapping $\mathbf{c}$ on the generator as $\mathbf{c}(\subset) := \left(\mathrm{i}_2, \left[\begin{pmatrix}0&1\\0&0\end{pmatrix}\right]\right)$, which, using the isomorphism $\mathbf{Mat}\colon$ Bialg $\to$ Mat$_{\mathbb{N}}$, becomes



The image on the rest of the morphisms of 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 $\begin{pmatrix}0&0\\1&0\end{pmatrix} \sim \begin{pmatrix}0&1\\0&0\end{pmatrix}$.

$\mathbf{c}(\subset ⨟ \times)$

$:= \mathbf{c}(\subset) ⨟ \mathbf{c}(\times)$



$$= \left(\mathrm{i}_2, \left[\begin{pmatrix}0&0\\1&0\end{pmatrix}\right]\right)$$

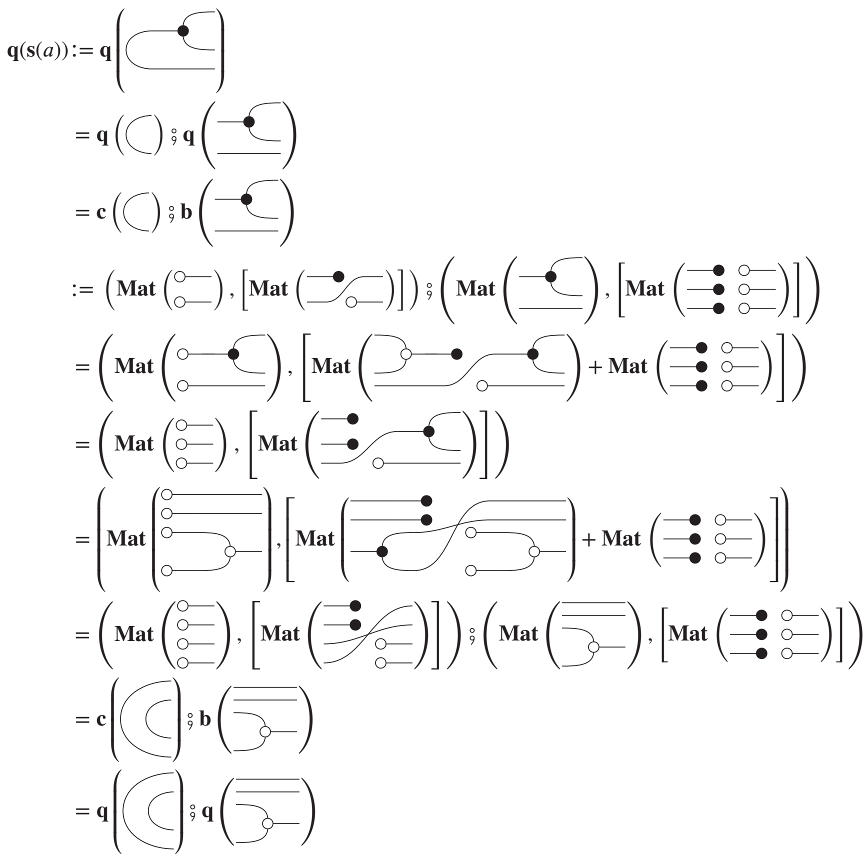
$$= (\mathrm{i}_2, [\left(\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right)])$$
$$=: \mathbf{c}(\subset)$$

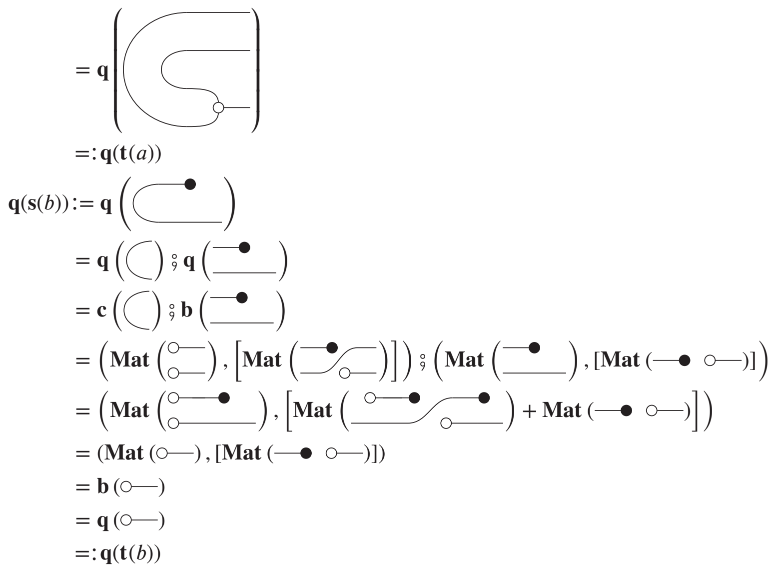
□

The prop morphism $\mathbf{q}\colon$ Bialg+Cup → MAdj 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* $\mathbf{q}\colon$ Bialg + Cup → MAdj *of the pair* $\mathbf{s}, \mathbf{t}\colon$ A → Bialg + Cup.

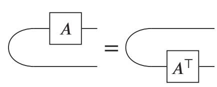
Proof. The prop morphism $\mathbf{b}\colon$ Bialg → MAdj is defined as the composition of the isomorphism $\mathbf{Mat}\colon$ Bialg → $\mathrm{Mat}_{\mathbb{N}}$, recalled in Proposition 4.18, with the morphism $\mathbf{j}\colon \mathrm{Mat}_{\mathbb{N}} \to$ MAdj, defined in Lemma 4.32. The prop morphism $\mathbf{c}\colon$ Cup → MAdj 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 $\mathbf{s} \,; \mathbf{q}$ and $\mathbf{t} \,; \mathbf{q}$ on both the morphisms $a\colon 0 \to 2$ and $b\colon 0 \to 1$ of the prop A. For both computations, we use the definition of $\mathbf{q}$ as a coproduct map.





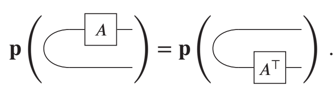
∎

We show that the prop morphism $\mathbf{q}$ satisfies the universal property of coequalisers. For every coequalising prop morphism $\mathbf{p}\colon$ Bialg + Cup → P of the pair $\mathbf{s}, \mathbf{t}\colon$ A → Bialg + Cup, Proposition 4.38 defines a candidate extension $\bar{\mathbf{p}}\colon$ MAdj → P of $\mathbf{p}$ to MAdj. 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.



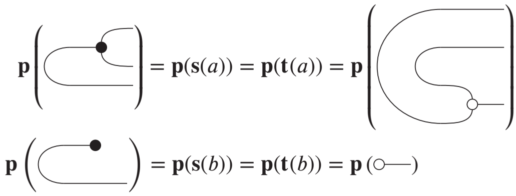
This equation holds in Adj and also in the image of any coequalising morphism of $\mathbf{s}$ and $\mathbf{t}$.

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

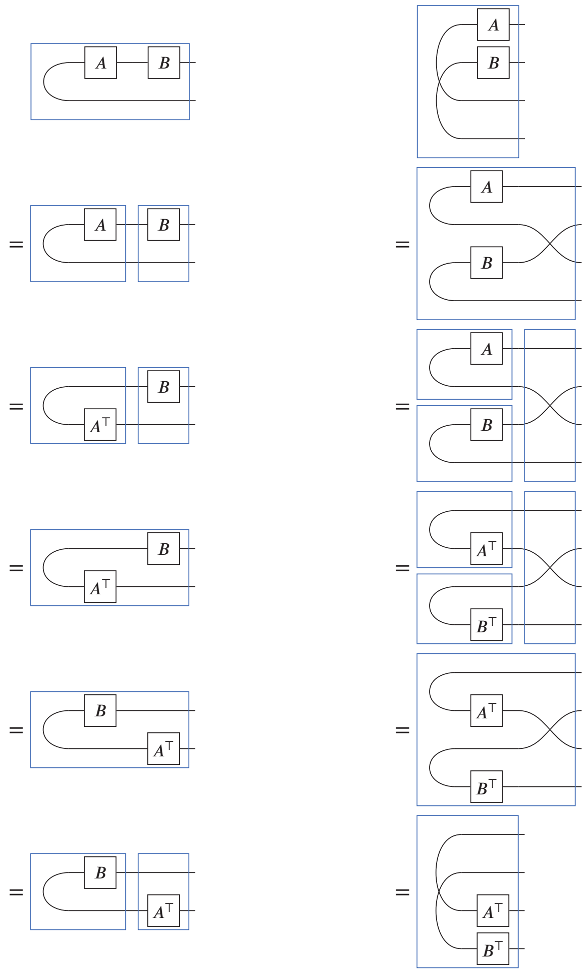


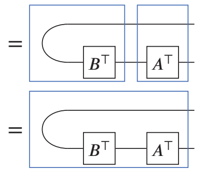
(4.3)

Proof. By Proposition 4.18, every morphism $A\colon n \to m$ in $\mathrm{Mat}_{\mathbb{N}}$ can be written as compositions and monoidal products of finitely many of its generators. These generators are the images under the isomorphism $\mathbf{Mat}\colon$ Bialg → 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 $\mathbf{s}$ and $\mathbf{t}$, $\mathbf{s} \,⨟\, \mathbf{p} = \mathbf{t} \,⨟\, \mathbf{p}$.



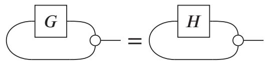
The two remaining equations follow by commutativity of the cup. For the inductive steps, suppose that Equation (4.3) holds for $A : n \to m$, $B : m \to l$ and $A' : n' \to m'$. We show that it holds for $A \,⨟\, B$ and for $A \oplus A'$. We indicate $\mathbf{p}$ with a blue functor box.





□

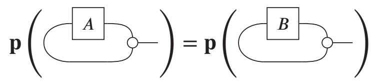
A consequence of this result is that equality in the prop of adjacency matrices captures the equivalence relation of adjacency matrices. Recalling Definition 4.21, two adjacency matrices are equivalent, $[G] = [H]$, if and only if they are equal up to transposition, $G + G^\top = H + H^\top$. In string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

$[G] = [H]$ iff 

,

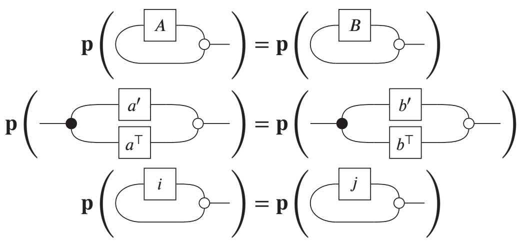
and holds in Adj and the image of any coequalising prop morphism of $\mathbf{s}$ and $\mathbf{t}$.

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

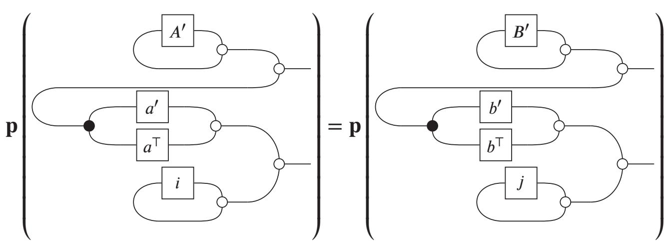
if $[A] = [B]$ *then* 

.

Proof. Proceed by induction on the size n of the matrices. For $n = 0$, there is only one morphism $0 \to 0$ in $\mathsf{Mat}_{\mathbb{N}}$ 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 $A = \left(\begin{smallmatrix} A' & a \\ a' & i \end{smallmatrix}\right)$ and $B = \left(\begin{smallmatrix} B' & b \\ b' & j \end{smallmatrix}\right)$. Notice that the two matrices are equivalent, $[A] = [B]$, if and only if $\left[A'\right] = \left[B'\right]$, $a' + a^\top = b' + b^\top$ and $i = j$, because $2 \cdot i = 2 \cdot j$ implies $i = j$. By induction hypothesis, $\left[A'\right] = \left[B'\right]$, $a' + a^\top = b' + b^\top$ and $i = j$ imply the corresponding equalities in the image of $\mathbf{p}$.

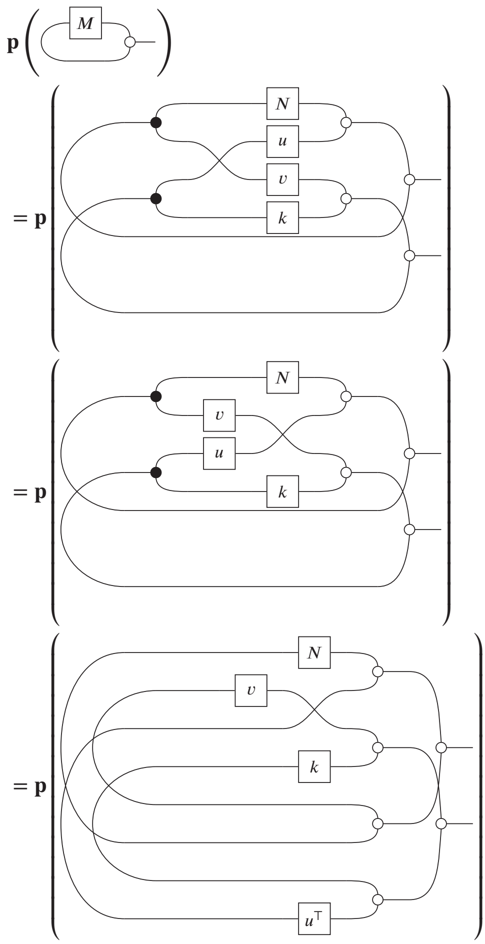


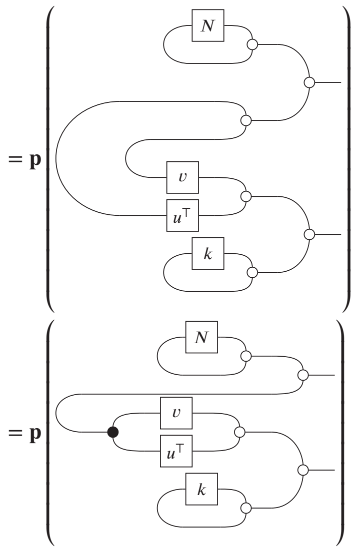
By functoriality of $\mathbf{p}$, we obtain



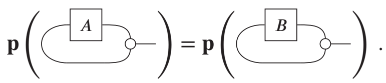
.

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 $M = \left(\begin{smallmatrix} N & u \\ v & k \end{smallmatrix}\right)$.



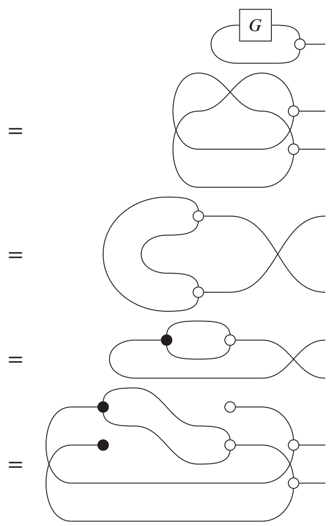


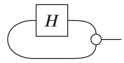
With these equalities, we obtain that



□

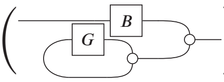
Example 4.37. The matrices $G = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$ and $H = \left(\begin{smallmatrix} 0 & 0 \\ 2 & 0 \end{smallmatrix}\right)$ are equivalent as adjacency matrices. In fact, their string diagrams are equal up to the equations of Adj.



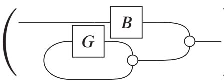
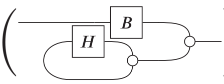
$=$ 

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

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

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

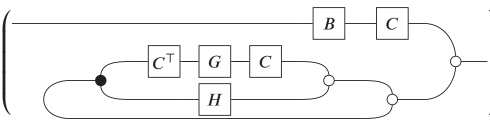
Proof. By Lemma 4.36 and functoriality of $\mathbf{p}$, the assignment $\bar{\mathbf{p}}$ is well-defined on equivalence classes of adjacency matrices: if $(B, [G]) = (B, [H])$, then $\bar{\mathbf{p}}(B, [G]) = \bar{\mathbf{p}}(B, [H])$ because

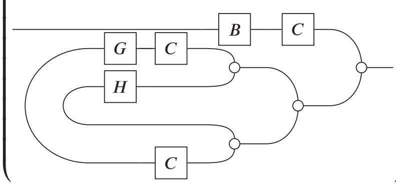
$\mathbf{p}$ () $= \mathbf{p}$ () .

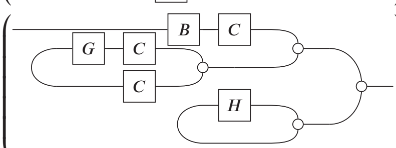
Applying Lemma 4.35, we check that $\bar{\mathbf{p}}$ preserves compositions.

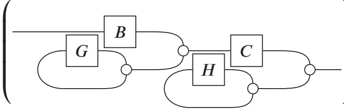
$$\bar{\mathbf{p}}((B, [G]) \,;\, (C, [H]))$$

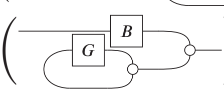
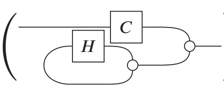
$$:= \bar{\mathbf{p}}(CB, \left[CGC^{\top} + H\right])$$

$:= \mathbf{p}$ ()

$= \mathbf{p}$ ()

$= \mathbf{p}$ ()

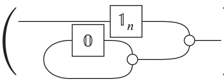
$= \mathbf{p}$ ()

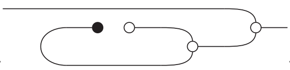
$= \mathbf{p}$ () $;\ \mathbf{p}$ ()

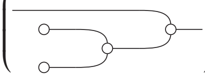
$$=: \bar{\mathbf{p}}(B, [G]) \,⨟\, \bar{\mathbf{p}}(C, [H])$$

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

$$\bar{\mathbf{p}}(\mathbb{1}_n, [\mathbb{0}_n])$$

$:= \mathbf{p}\Big($  $\Big)$

$= \mathbf{p}\Big($  $\Big)$

$= \mathbf{p}\Big($  $\Big)$

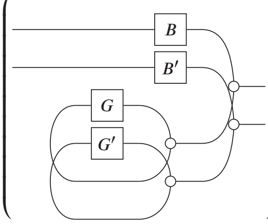
$= \mathbf{p}(—_n)$

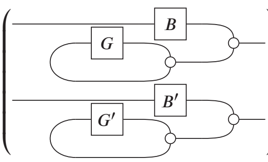
$= \mathbb{1}_n$

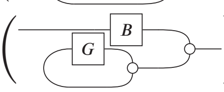
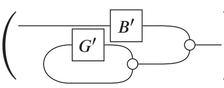
Finally, we check that $\bar{\mathbf{p}}$ preserves monoidal products.

$$\bar{\mathbf{p}}((B, [G]) \otimes (B', [G']))$$

$$:= \bar{\mathbf{p}}(B \oplus B', [G \oplus G'])$$

$:= \mathbf{p}\Big($  $\Big)$

$= \mathbf{p}\Big($  $\Big)$

$= \mathbf{p}\Big($  $\Big) \otimes \mathbf{p}\Big($  $\Big)$

$$=: \bar{\mathbf{p}}(B, [G]) \otimes \bar{\mathbf{p}}(B', [G'])$$

□

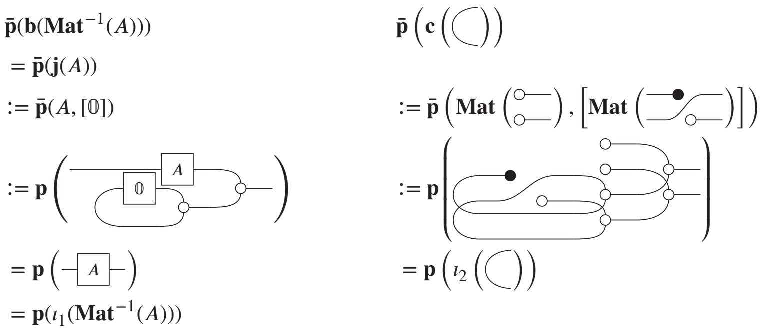
The candidate coequaliser of Proposition 4.34 is, indeed, a coequaliser. This follows from checking that the candidate extension of a coequalising prop morphism $\mathbf{p}$ in Proposition 4.38 is an extension of $\mathbf{p}$ along $\mathbf{q}$ and is unique. In particular, the prop morphism ϕ defined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

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

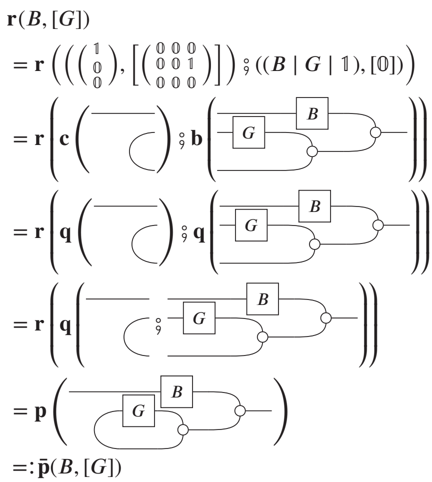
Proof. Proposition 4.34 provides a candidate $\mathbf{q}\colon$ Bialg + Cup $\to$ MAdj for the coequaliser of $\mathbf{s}$ and $\mathbf{t}$, and Proposition 4.38 defines a morphism $\bar{\mathbf{p}}\colon$ MAdj $\to$ P for any coequalising morphism $\mathbf{P}\colon$ Bialg + Cup $\to$ P. The prop morphism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

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

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



Suppose that there is another morphism $\mathbf{r}\colon$ MAdj $\to$ P such that $\mathbf{q} \fatsemi \mathbf{r} = \mathbf{p}$. We check that $\mathbf{r}$ must coincide with $\bar{\mathbf{p}}$.



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

We will add vertices to adjacency matrices to obtain graphs, but, first, we study vertices on their own. The prop Vert is freely generated by a "vertex" $1 \to 0$ generator, so morphisms are permutations with some outputs bounded by vertices.

Definition 4.40. The prop Vert is freely generated by one $\mathsf{v} \colon 1 \to 0$ generator and no extra equations (Figure 4.5).

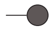
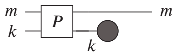


Figure 4.5: Generator of the one-vertex prop.

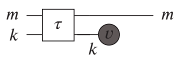
Graphs are adjacency matrices with vertices. The prop Vert is isomorphic to that of bounded permutations, boundP, via the isomorphism that composes the bounded outputs with vertices.

$\psi \colon (k, P) \mapsto$ 

This defines an isomorphism because we can check initiality of boundP.

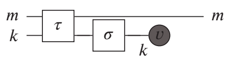
Proposition 4.41. *The freely generated prop* Vert *is isomorphic to that of bounded permutations,* boundP $\cong$ Vert.

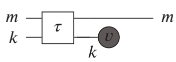
Proof. We show that boundP also satisfies the universal property of Vert: it is initial among the props with a $1 \to 0$ morphism. Let P be a prop with a morphism $v \colon 1 \to 0$ and define $\mathbf{H} \colon$ boundP $\to$ P as identity on objects and, on morphisms, as

$\mathbf{H}(k, P_\tau) := \tau \,\mathbin{⨾}\, (\mathbb{1}_m \otimes v^k) =$  ,

where P_τ is the permutation matrix corresponding to the permutation τ and v^k is the k-fold monoidal product of v with itself. Then, $\mathbf{H}(1, \mathbb{1}_1) = v$ and $\mathbf{H}$ is well-defined on equivalence classes by naturality of the symmetries in P.

$\mathbf{H}(k, (\mathbb{1}_m \oplus P_\sigma) P_\tau)$

$:=$ 

$=$ 

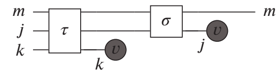
$=: \mathbf{H}(k, P_\tau)$

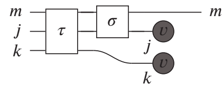
The definition above gives a functor because $\mathbf{H}$ preserves identities,

$$\mathbf{H}(0, \mathbb{1}_m) := \mathbb{1}_m ,$$

and preserves compositions.

$$\mathbf{H}(k, P_\tau) \mathbin{⨾} \mathbf{H}(j, P_\sigma)$$





$$= \mathbf{H}(k + j, (P_\sigma \otimes \mathbb{1}_k)P_\tau)$$
$$= \mathbf{H}((k, P_\tau) \mathbin{⨟} (j, P_\sigma)) \,.$$

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

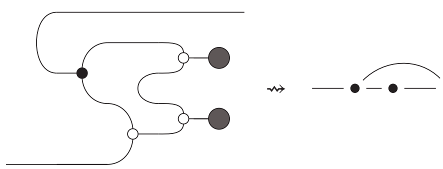
$$\begin{aligned}
&\mathbf{F}(k, P_\tau)\\
&= \mathbf{F}((0, P_\tau) \mathbin{⨟} ((0, \mathbb{1}_m) \otimes (k, \mathbb{1}_k)))\\
&= \mathbf{F}(\tau \mathbin{⨟} (\mathbb{1}_m \otimes (1, \mathbb{1}_1)^k))\\
&= \mathbf{F}(\tau) \mathbin{⨟} (\mathbf{F}(\mathbb{1}_m) \otimes \mathbf{F}(1, \mathbb{1}_1)^k)\\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

□

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]).

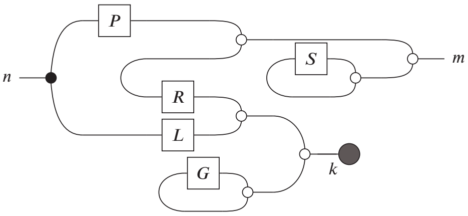
Definition 4.42. The prop of graphs BGraph is the coproduct of the props Adj and Vert, BGraph := Adj+Vert. Its generators and equations are in Figures 4.3 to 4.5.

Example 4.43. The string diagram below on the left is a morphism $1 \to 1$ in 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 MGraph ≅ BGraph gives a normal form for morphisms in BGraph. This is the coproduct of the isomorphisms $\phi\colon$ MAdj ≅ Adj and $\psi :$ boundP $\cong$ Vert.

$\theta : ([G], L, R, P, [S]) \mapsto$



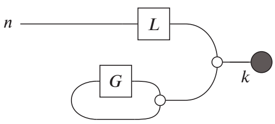
Theorem 4.44. *The prop of graphs* BGraph *is isomorphic to* MGraph.

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). □

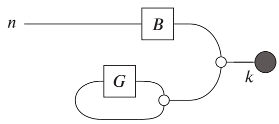
The operations for clique width and rank width

This section repeats the prodedure 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 $n \to 0$ in BGraph. An n-labelled graph (G, l) corresponds to the morphism $\big([G], L, ¡, !, [()]\big)$,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 $\big([G], B, ¡, !, [()]\big)$.

$(G, l) \mapsto$



and $(G, B) \mapsto$



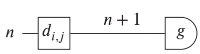
The various presentations of the operations for clique width [CER93; CO00; CV03] and rank width [CK07; CK09] define equivalent complexity measures. This becomes apparent when we express these operations as compositions and monoidal products in BGraph and its categorical structure becomes the canonical 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 $-\!\!\bullet_1 : 1 \to 0$, while the 1-labelled single vertex graph v_1 is the vertex generator $\mathsf{v}_1 : 1 \to 0$.

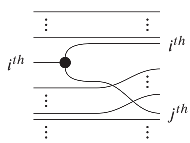
$\emptyset_1 \mapsto 1 -\!\!\bullet$ and $\mathsf{v}_1 \mapsto 1 -\!\!\!\bullet$

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

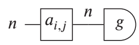
$\mathsf{Rename}^n{}_{j \to i}(G, l) \mapsto$

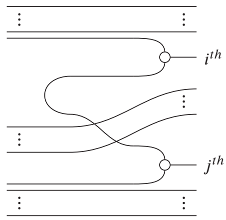


where $d_{i,j} :=$

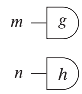


The creation of edges $\mathsf{Edge}^n{}_{i,j}(G,l)$ between the labels i and j is also a precomposition. We compose the morphism $a_{i,j} : n \to n$, which connects the i^{th} and j^{th} outputs through a cup, with the morphism g that corresponds to (G,l).

$\mathsf{Edge}^n{}_{i,j}(G,l) \mapsto$ 

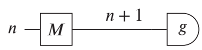
where $a_{i,j} :=$ 

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.

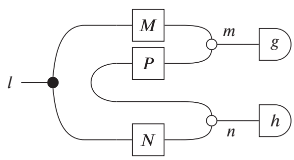
$(G,l)+(H,l') \mapsto$ 

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 $n \to 0$ 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 $\mathsf{Recol}_M(G,B)$ 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.

$\mathsf{Recol}_M(G,B) \mapsto$ 

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.

$(G,B)+_{M,P,N}(H,C) \mapsto$ 

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 $n \to 0$ 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

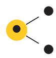
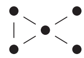
Similarly to the Courcelle's graph expressions recalled in Section 2.3 ([BC87, Definition 3.4] and [Cou90, Definition 2.7]), monoidal decompositions in cospans of hypergraphs are also terms for hypergraphs, but where the operations are compositions and monoidal product in $\mathsf{Cospan}(\mathsf{UHGraph})_*$. 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 $\mathsf{Cospan}(\mathsf{UHGraph})_*$ of discrete cospans of hypergraphs instead of the category $\mathsf{sStruct}_\tau$ 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

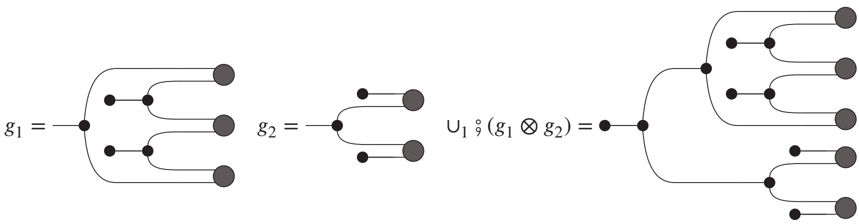
$X \subseteq V$ of its vertices, called the *sources*. Given two graphs with sources $\Gamma = (G, X)$ and $\Gamma' = (G', X')$, we say that Γ' is a subgraph of Γ whenever G' is a subgraph of G.

Note that the sources of a subhypergraph Γ' of Γ need not to appear as sources of Γ, nor vice versa. In fact, if Γ is obtained by identifying all the sources of a hypergraph Γ_1 with some of the sources of another hypergraph Γ_2, the sources of Γ and Γ_1 will be disjoint. A hypergraph with sources $\Gamma = (G, X)$ can be seen as a morphism $g : X \to \emptyset$ in Cospan(UHGraph)$_*$: $g = : X \to G \leftarrow \emptyset :$, where the legs of the cospan are $\iota : X \to V$ and $¡ : \emptyset \to V$.

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":

 glued with  gives  .

These two graphs correspond to two morphisms $g_1, g_2 : 1 \to \emptyset$ in Cospan(UHGraph)$_*$ that can be composed to obtain the rightmost graph $\subset_1 ⨾ (g_1 \otimes g_2)$.



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

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

An inductive branch decomposition of a hypergraph with sources Γ is a binary tree $T \in \mathcal{T}_\Gamma$ satisfying some conditions such that, identifying the common sources in Γ_1 and Γ_2, we obtain Γ.

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 Γ 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 Γ 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 Γ 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:

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

Expanding this expression, we obtain

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

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 Γ and T_0 be a full subtree of T. Then,*

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

where $T' \not\gtrless T_0$ 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 $\mathsf{sources}(\lambda(T_0)) = X = V \cap X$. 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

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

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:* $\mathsf{wd}(\mathcal{I}^{\dagger}(T)) \leq \mathsf{wd}(T)$.

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 Γ 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 $b\colon \mathsf{leaves}(Y) \to \mathsf{edges}(G)$ 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 $e \in \mathsf{edges}(Y)$ then $e \in \mathsf{edges}(T)$. Let $\{v, w\} = \mathsf{ends}(e)$ 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, $E_w = \mathsf{edges}(\lambda(T_w))$ and $E_v = \bigcup_{T' \ngeq T_w} \mathsf{edges}(\lambda(T'))$, and, for the vertices sets, $\mathsf{ends}(E_w) \subseteq \mathsf{vertices}(\lambda(T_w))$ and $\mathsf{ends}(E_v) \subseteq \bigcup_{T' \ngeq T_w} \mathsf{vertices}(\lambda(T'))$. Using these inclusions and applying Lemma 5.7,

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

□

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 Γ of bounded width:* $\mathsf{wd}(\mathcal{I}(Y, b)) \leq \mathsf{wd}(Y, b) + |X|$.

Proof. Proceed by induction on $|\mathsf{edges}(Y)|$. 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 $\mathsf{wd}(\mathcal{I}(Y, b)) := |X| \leq \mathsf{wd}(Y, b) + |X|$.

If Y has at least one edge e, then $Y = Y_1 \overset{e}{-} Y_2$ with Y_i a subcubic tree. Let $E_i = b(\mathsf{leaves}(Y_i))$ 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 $V_1 := \mathsf{ends}(E_1)$, $V_2 := \mathsf{ends}(E_2) \cup (V \setminus V_1)$, $X_i := (V_1 \cap V_2) \cup (V_i \cap X)$ 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 Γ and, applying Lemma 5.7,

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

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

□

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 $\mathsf{Cospan}(\mathsf{UHGraph})_*$ to count the number of vertices of the apex graph in each cospan.

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

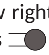
With this definition, the identity on X weights $|X|$ and compositions along X cost $|X|$. This definition gives a weight function.

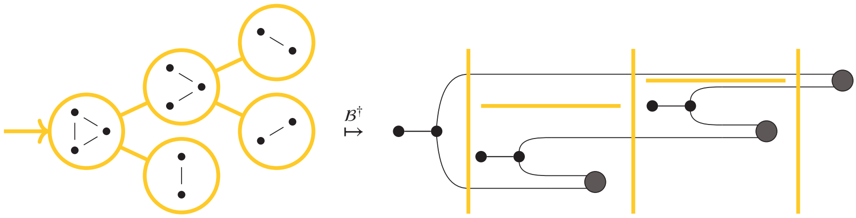
Lemma 5.12. *The function* w *in Definition 5.11 satisfies the conditions in Definition 3.3 for a weight function in the monoidal category* $\mathsf{Cospan}(\mathsf{UHGraph})_*$.

Proof. For $f : X \to Y$, $g : Y \to Z$ and $f' : X' \to Y'$ with sets of vertices V, W and V', we can bound the weights of $f \mathbin{⨟} g$ and $f \otimes f'$.

$$\begin{aligned} &\mathsf{w}(f \mathbin{⨟}_Y g) \\ &:= |V +_Y W| \\ &\leq |V| + |W| + |Y| \\ &=: \mathsf{w}(f) + \mathsf{w}(g) + \mathsf{w}(Y) \end{aligned} \qquad \begin{aligned} &\mathsf{w}(f \otimes f') \\ &:= |V + V'| \\ &= |V| + |V'| \\ &=: \mathsf{w}(f) + \mathsf{w}(f') \end{aligned}$$

□

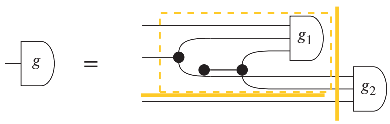
A branch decomposition divides a hypergraph into one-edge subhypergraphs. Given a branch decomposition of a hypergraph Γ with sources, the corresponding monoidal decomposition is defined by taking all the one-edge subhypergraphs and composing them according to the tree structure of the branch decomposition. For example, the monoidal decomposition shown below right corresponds to the inductive branch decomposition of 3-clique at its left: the three edge generators  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 $g := \iota : X \to G \leftarrow \emptyset :$ 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:* $\mathsf{wd}(\mathcal{B}^\dagger(T)) \leq \max\{\mathsf{wd}(T) + 1, \gamma(G)\}$.

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 Γ of this leaf must have at most one hyperedge with $\gamma(G)$ endpoints and $\mathsf{wd}(T) := |X|$. We define the corresponding monoidal decomposition to also consist of only a leaf, $\mathcal{B}^\dagger(T) := (g)$, and obtain the desired bound $\mathsf{wd}(\mathcal{B}^\dagger(T)) = \max\{|X|, \gamma(G)\} = \max\{\mathsf{wd}(T), \gamma(G)\}$.

If $T = (T_1 \text{—} \Gamma \text{—} T_2)$, 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 $g_i = \iota : X_i \to G_i \leftarrow \emptyset :$ be the morphism in $\mathsf{Cospan}(\mathsf{UHGraph})_*$ corresponding to Γ_i. Then, we decompose g in terms of identities, the structure of $\mathsf{Cospan}(\mathsf{UHGraph})_*$, 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: $\mathsf{wd}(\mathcal{B}^\dagger(T_i)) \leq \max\{\mathsf{wd}(T_i) + 1, \gamma(G_i)\}$. 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: $\mathsf{wd}(\mathcal{C}(\mathcal{B}^\dagger(T_1))) \leq \max\{\mathsf{wd}(\mathcal{B}^\dagger(T_1)), |X_1| + 1\}$. Using this decomposition, we can define the monoidal decomposition given by the cuts in the figure above.

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

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

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

□

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 $\phi \colon V \to W$ and define the hypergraph with sources $\phi(\Gamma) := ((\phi(V), E), \phi(X))$. Suppose there is an inductive branch decomposition T of Γ. Then, there is an inductive branch decomposition $\phi(T)$ of $\phi(\Gamma)$ of bounded width:* $\mathsf{wd}(\phi(T)) \leq \mathsf{wd}(T)$.

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: $\mathsf{wd}(\phi(T)) := |\phi(X)| \leq |X| =: \mathsf{wd}(T)$.

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 $\phi_1 : V_1 \to W$ and $\phi_2 : V_2 \to W$ the compositions of ϕ 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, $\mathsf{wd}(\phi_i(T_i)) \leq \mathsf{wd}(T_i)$. Define $\phi(T) := (\phi_1(T_i) — \phi(\Gamma) — \phi_2(T_2))$ 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:

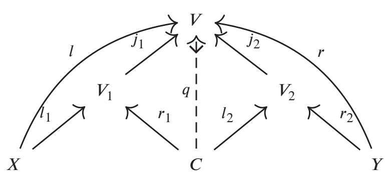
$$
\begin{aligned}
&\mathsf{wd}(\phi(T)) \\
&:= \max\{\mathsf{wd}(\phi_1(T_1)), \mathsf{wd}(\phi_2(T_2)), |\phi(X)|\} \\
&\leq \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2), |X|\} \\
&=: \mathsf{wd}(T)
\end{aligned}
$$

□

Proposition 5.15. *Let $d \in D_g$ be a monoidal decomposition of a morphism $g = l : X \to G \leftarrow Y : r$ in* Cospan(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 Γ of bounded width:* $\mathsf{wd}(\mathcal{B}(d)) \leq 2 \cdot \max\{\mathsf{wd}(d), |X|, |Y|\}$.

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 Γ. The width of an inductive branch decomposition of Γ is bounded by the number of vertices of Γ and, as a consequence, by the width of d: $\mathsf{wd}(\mathcal{B}(d)) \leq |V| =: \mathsf{wd}(d) \leq 2 \cdot \max\{\mathsf{wd}(d), |X|, |Y|\}$.

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



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: $\mathsf{wd}(\mathcal{B}(d_1)) \leq 2 \cdot \max\{\mathsf{wd}(d_1), |X|, |C|\}$ and $\mathsf{wd}(\mathcal{B}(d_2)) \leq 2 \cdot \max\{\mathsf{wd}(d_2), |Y|, |C|\}$. 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: $\mathsf{wd}(j_i(\mathcal{B}(d_i))) \leq \mathsf{wd}(\mathcal{B}(d_i))$. 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 Γ 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,

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

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

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

Suppose that $d = (d_1— \otimes —d_2)$ starts with a monoidal product node. Then, $g = g_1 \otimes g_2$ for two morphisms $g_1 = l_1 : X_1 \to G_1 \leftarrow Y_1 : r_1$ and $g_2 = l_2 : X_2 \to G_2 \leftarrow Y_2 : r_2$. 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: $\mathsf{wd}(\mathcal{B}(d_1)) \leq 2 \cdot \max\{\mathsf{wd}(d_1), |X_1|, |Y_1|\}$ and $\mathsf{wd}(\mathcal{B}(d_2)) \leq 2 \cdot \max\{\mathsf{wd}(d_2), |X_2|, |Y_2|\}$. 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 Γ because it satisfies the conditions of Definition 5.4: $E = E_1 \sqcup E_2$, $V = V_1 \cup V_2$ and $l_i(X_i) \cup r_i(Y_i) = ((l(X) \cup r(Y)) \cap V_i) \cup (V_1 \cap V_2)$. The width of $\mathcal{B}(d)$ is bounded.

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

□

Theorem 5.16 summarises Propositions 5.10, 5.13 and 5.15.

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

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

Corollary 5.17. *Tree width is equivalent to monoidal width in* Cospan(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 $\mathsf{Cospan}(\mathsf{UHGraph})_*$ 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 BGraph. Here, we show that the prop 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.

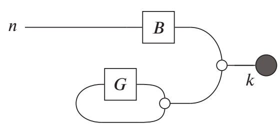
6.1 Inductive rank decompositions

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".

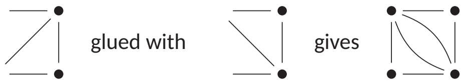
Definition 6.1. A *graph with dangling edges* is a pair $\Gamma = ([G], B)$ of an adjacency matrix $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ that records the connectivity of the graph and a matrix $B \in \mathsf{Mat}_{\mathbb{N}}(k,n)$ that records the dangling edges connected to n boundary ports. Two graphs 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 $P \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ such that $G = P \cdot G' \cdot P^{\top}$ and $B = P \cdot B'$.

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

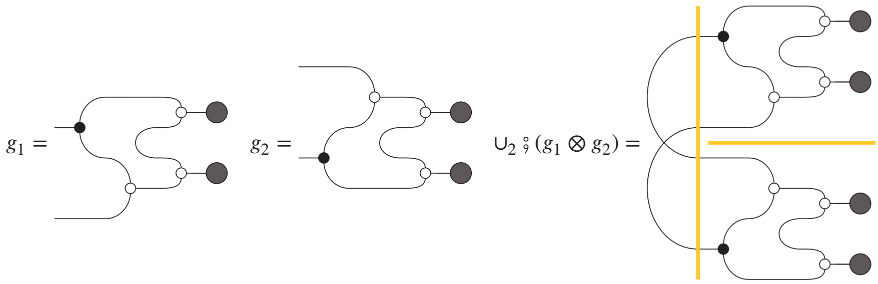
A graph with dangling edges $\Gamma = ([G], B)$ can be seen as a morphism $n \to 0$ in BGraph.



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



These two graphs correspond to two morphisms $g_1, g_2 : 2 \to 0$ in BGraph that can be composed to obtain the rightmost graph $\subset_2 \fatsemi (g_1 \otimes g_2)$.



An inductive rank decomposition of Γ is a binary tree satisfying some conditions that ensure that composing the dangling edges of Γ_1 with those of Γ_2 gives Γ.

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

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

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

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

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 Γ 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 Γ in the decomposition to the boundary and adjacency matrices of Γ.

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 Γ 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 Γ': $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$.*

Proof. Proceed by induction on the decomposition tree T. If it is just a leaf, $T = (\Gamma)$, then Γ 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 Γ 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, $\mathsf{rk}(B') = \mathsf{rk}(A' \mid E' \mid C_L^\top \mid D_R)$. The rank is invariant to permuting the order of columns, thus $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid D_R \mid E') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$. We proceed analogously if T' is a full subtree of T_2. ☐

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 Γ. Then, there is a rank decomposition $\mathcal{I}^\dagger(T)$ of G of bounded width: $\mathsf{wd}(\mathcal{I}^\dagger(T)) \leq \mathsf{wd}(T)$.*

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 $r : \mathsf{leaves}(Y) \to \mathsf{vertices}(G)$ 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 Γ_l of Γ with one vertex v_l. These subgraphs need to give Γ 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 $\{v, v_b\} = \mathsf{ends}(b)$ 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 Γ identified by T_b. We can express the adjacency and boundary matrices of Γ in terms of those of Γ_b:

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

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

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

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

□

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 Γ of bounded width:* $\mathsf{wd}(\mathcal{I}(Y, r)) \leq \mathsf{wd}(Y, r) + \mathsf{rk}(B)$.

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 Γ, $\mathcal{I}(Y, r) := (\Gamma)$. We compute its width by definition: $\mathsf{wd}(\mathcal{I}(Y, r)) := \mathsf{rk}(B) \leq \mathsf{wd}(Y, r) + \mathsf{rk}(B)$.

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 $V_i := r(\mathsf{leaves}(Y_i))$ 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 $\Gamma_i = (\left[G_i\right], B_i)$ 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 Γ. Let T' be a full subtree of T corresponding to $\Gamma' = (\left[G'\right], B')$. By Lemma 6.7, we can compute the rank of its boundary matrix $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$, 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 $\mathsf{rk}(B') \leq \mathsf{rk}(B) + \mathsf{rk}(C_L^\top \mid C_R)$. 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 $\mathsf{ord}(b')$. This means that the rank of this matrix is the order of the edge b': $\mathsf{rk}(C_L^\top \mid C_R) = \mathsf{ord}(b')$. With these observations, we can compute the width of T.

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

□

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 $g : n \to m$ in MGraph, the weight function w is defined as $\mathsf{w}(g) := \mathsf{rk}(G) + \mathsf{rk}(L) + \mathsf{rk}(R) + \mathsf{rk}(P) + \mathsf{rk}(F)$, where $g = ([G], L, R, P, [F])$.

With this definition, the identity on n weights n because $\mathsf{rk}(\mathbb{1}_n) = n$, 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 $g : n \to m$, $h : m \to l$ and $g' : n' \to m'$, 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 $g \mathbin{;} h$ and the monoidal product $g \otimes g'$.

$$g \mathbin{;} h := \left(\left[\begin{pmatrix} G & RM^\top \\ \mathbb{0} & H+MFM^\top \end{pmatrix} \right], \begin{pmatrix} L \\ MP \end{pmatrix}, \begin{pmatrix} RQ^\top \\ S+M(F+F^\top)Q^\top \end{pmatrix}, QP, \left[E + QFQ^\top\right] \right)$$

$$g \otimes g' := \left([G \oplus G'], L \oplus L', R \oplus R', P \oplus P', [F \oplus F'] \right)$$

We bound the ranks of these matrices individually.

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

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

$$\mathsf{w}(g \mathbin{;}_m h) \leq \mathsf{w}(g) + \mathsf{w}(h) + m \qquad \mathsf{w}(g \otimes g') \leq \mathsf{w}(g) + \mathsf{w}(g')$$

□

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 $n \to 0$ 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.

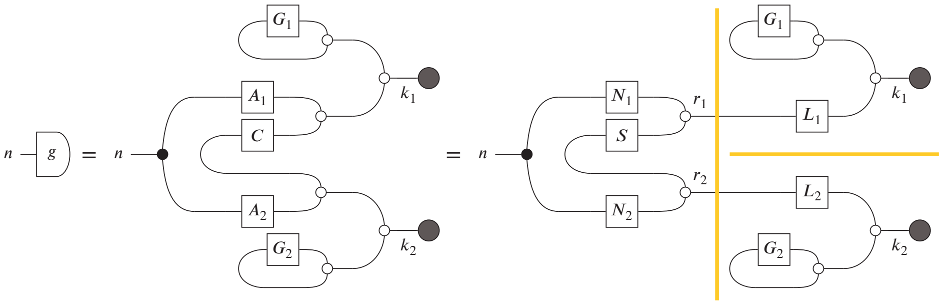


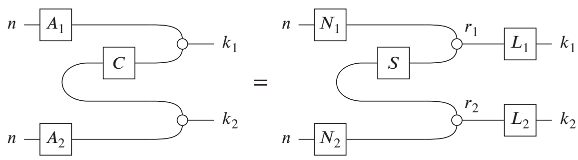
Figure 6.1: Splitting a graph with dangling edges optimally into subgraphs.

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

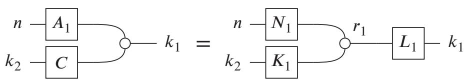
Lemma 6.14. *Let $A_i \in \mathsf{Mat}_{\mathbb{N}}(k_i, n)$, for $i = 1, 2$, and $C \in \mathsf{Mat}_{\mathbb{N}}(k_1, k_2)$. Then, there are rank decompositions of $(A_1 \mid C)$ and $(A_2 \mid C^\top)$ of the form*

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

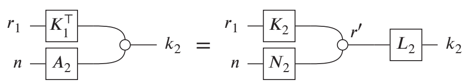
This ensures that we can decompose the diagram below on the left-hand-side as the one on the right-hand-side, where $r_1 = \mathsf{rk}(A_1 \mid C)$ and $r_2 = \mathsf{rk}(A_2 \mid C^\top)$.



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



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

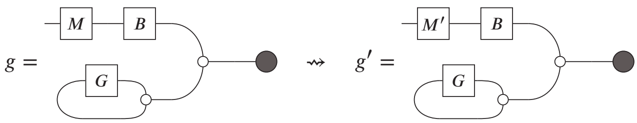


with $L_2 \in \mathsf{Mat}_{\mathbb{N}}(k_2, r')$, $N_2 \in \mathsf{Mat}_{\mathbb{N}}(r', n)$ and $K_2 \in \mathsf{Mat}_{\mathbb{N}}(r', k_1)$. We show that $r' = r_2$. By the first factorisation, we obtain that $C = L_1 \cdot K_1$, and

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

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

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 $\mathsf{wd}(T) \leq \mathsf{wd}(T')$ and such that T and T' have the same underlying tree structure. If, moreover, M' has full rank, then $\mathsf{wd}(T) = \mathsf{wd}(T')$.*

Proof. Proceed by induction on the decomposition tree T. If the tree T is just a leaf with label Γ, then we define the corresponding tree to be just a leaf with label Γ': $T' := (\Gamma')$. 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: $\mathsf{wd}(T') := \mathsf{rk}(B \cdot M') \leq \mathsf{rk}(B) = \mathsf{rk}(B \cdot M) =: \mathsf{wd}(T)$. If, moreover, M' has full rank, the inequality becomes an equality and $\mathsf{wd}(T') = \mathsf{wd}(T)$.

If $T = (T_1 \text{—} \Gamma \text{—} T_2)$, then the adjacency and boundary matrices of Γ 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: $G = \begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}$, $B \cdot M = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \cdot M = \begin{pmatrix} A_1 \cdot M \\ A_2 \cdot M \end{pmatrix}$, 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 Γ_i can also be expressed as a composition with a full-rank matrix: $D_1 = (A_1 \cdot M \mid C) = (A_1 \mid C) \cdot \begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix}$ and $D_2 = (A_2 \cdot M \mid C^\top) = (A_2 \mid C^\top) \cdot \begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix}$. The matrices $\begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_i} \end{pmatrix}$ 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 $\Gamma_1' = ([G_1], B_1 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix})$ and $\Gamma_2' = ([G_2], B_2 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix})$ with the same underlying tree structure as T_1 and T_2, respectively. Moreover, their width is bounded, $\mathsf{wd}(T_i') \leq \mathsf{wd}(T_i)$, and if, additionally, M' has full rank, $\mathsf{wd}(T_i') = \mathsf{wd}(T_i)$. Then, we can use these decompositions to define an inductive rank decomposition $T' := (T_1' \text{—} \Gamma' \text{—} T_2')$ of Γ' because its adjacency and boundary matrices can be expressed in terms of those of Γ_i' as in the definition of inductive rank decomposition: $G = \begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}$, $B_1 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix} = (A_1 \cdot M' \mid C)$ and $B_2 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix} = (A_2 \cdot M' \mid C^\top)$. Applying the induction hypothesis and Remark 6.13, we compute the width of this decomposition.

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

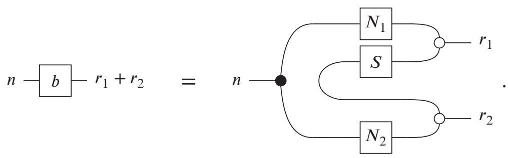
If, moreover, M' has full rank, the inequality becomes an equality and $\mathsf{wd}(T') = \mathsf{wd}(T)$. □

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 $g : n \to 0$ be the morphism in* BGraph *corresponding to Γ. Let T be a inductive rank decomposition of Γ. Then, there is a monoidal decomposition $\mathcal{R}^\dagger(T)$ of g of bounded width* $\mathsf{wd}(\mathcal{R}^\dagger(T)) \leq 2 \cdot \mathsf{wd}(T)$.

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

If $T = (T_1 — \Gamma — T_2)$, then we can relate the adjacency and boundary matrices of Γ to those of $\Gamma_i := \lambda(T_i) = ([G_i], B_i)$, by definition of inductive rank decomposition: $G = \begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}$, $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$, $B_1 = (A_1 \mid C)$ 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 $r_i = \mathsf{rk}(B_i)$. 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 $\mathsf{wd}(T_i) = \mathsf{wd}(T_i')$. Let $g_i : r_i \to 0$ be the morphisms in BGraph corresponding to Γ_i' and let $b : n \to r_1 + r_2$ be defined as



By induction hypothesis, there are monoidal decompositions $\mathcal{R}^\dagger(T_i')$ of the morphisms g_i of bounded width: $\mathsf{wd}(\mathcal{R}^\dagger(T_i')) \leq 2 \cdot \mathsf{wd}(T_i') = 2 \cdot \mathsf{wd}(T_i)$. Then, $g = b \,;_{r_1+r_2} (g_1 \otimes g_2)$ and $\mathcal{R}^\dagger(T) := (b— ;_{r_1+r_2} —(\mathcal{R}^\dagger(T_1')— \otimes —\mathcal{R}^\dagger(T_2')))$ is a monoidal decomposition of g. Its width can be computed.

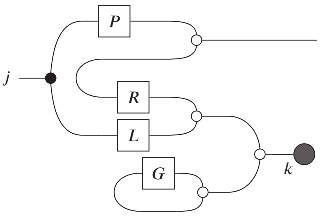
$$\begin{aligned}
&\mathsf{wd}(\mathcal{R}^\dagger(T)) \\
&:= \max\{\mathsf{w}(b), \mathsf{w}(r_1 + r_2), \mathsf{wd}(\mathcal{R}^\dagger(T_1')), \mathsf{wd}(\mathcal{R}^\dagger(T_2'))\} \\
&\leq \max\{\mathsf{w}(b), \mathsf{w}(r_1 + r_2), 2 \cdot \mathsf{wd}(T_1'), 2 \cdot \mathsf{wd}(T_2')\} \\
&=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

□

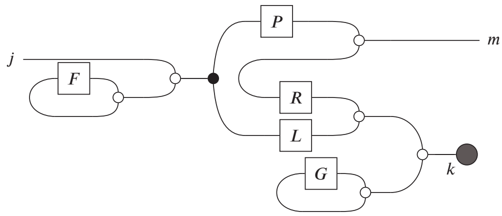
Each node in a monoidal decomposition of a graph g determines a cut in g. This correspondence maps monoidal decompositions to inductive rank decompositions. However, bounding their widths requires some care because the splitting determined by a monoidal decomposition may be not the canonical one needed 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 $f \,;\, g$ below right, of at most the same width.

$g =$



$f \mathbin{⨟} g =$



Lemma 6.17. *Let T be an inductive rank decomposition of $\Gamma = ([G], (L \mid R))$, with $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$, $L \in \mathsf{Mat}_{\mathbb{N}}(k,j)$ and $R \in \mathsf{Mat}_{\mathbb{N}}(k,m)$. Let $F \in \mathsf{Mat}_{\mathbb{N}}(j,j)$, $P \in \mathsf{Mat}_{\mathbb{N}}(m,j)$ and define the graph Γ' 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 Γ' such that $\mathsf{wd}(T') \leq \mathsf{wd}(T)$.*

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

Proceed by induction on the decomposition tree T.

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

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 Γ_i, and we can relate the adjacency and boundary matrices of Γ to those of Γ_1 and Γ_2, by definition of inductive rank decomposition: $[G] = \left[\begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}\right]$ 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 Γ' in terms of the same components: $[G + L \cdot F \cdot L^\top] = \left[\begin{pmatrix} G_1 + L_1 \cdot F \cdot L_1^\top & C + L_1 \cdot (F+F^\top) \cdot L_2^\top \\ \mathbb{0} & G_2 + L_2 \cdot F \cdot L_2^\top \end{pmatrix}\right]$ 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 Γ' 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 Γ_i' such that $\mathsf{wd}(T_i') \leq \mathsf{wd}(T_i)$. We defined Γ_i' so that $T' := (T_1' — \Gamma' — T_2')$ would be an inductive rank decomposition of Γ'. We can bound its width as desired.

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

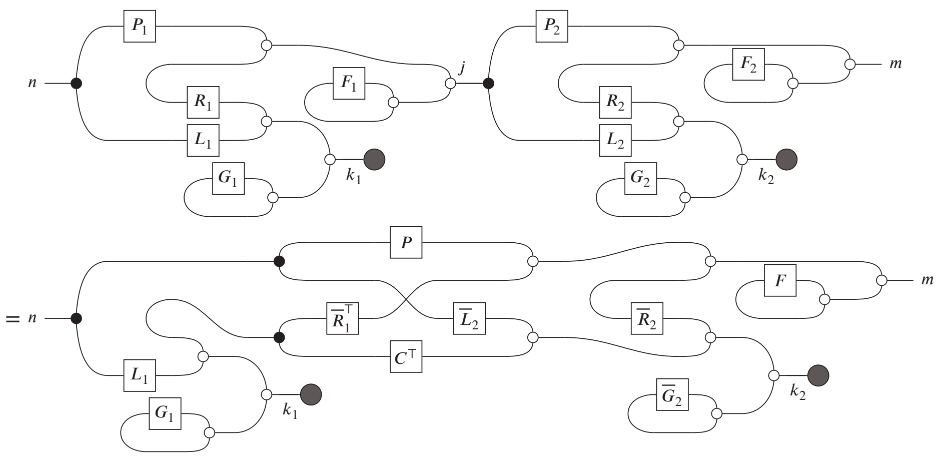
□

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 $g : n \to m$ in* BGraph *given by $g = ([G], L, R, P, [F])$, and let $\Gamma = ([G], (L \mid R))$ be its corresponding graph with dangling edges. Then, there exist an inductive rank decomposition $\mathcal{R}(d)$ of Γ of bounded width:* $\mathsf{wd}(\mathcal{R}(d)) \leq 2\cdot\max\{\mathsf{wd}(d), \mathsf{rk}(L), \mathsf{rk}(R)\}$.

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, $\mathsf{wd}(d) := k$. Pick any inductive rank decomposition of Γ and define $\mathcal{R}(d) := T$. Surely, $\mathsf{wd}(T) \leq k =: \mathsf{wd}(d)$

If $d = (d_1 - ⨟_j - d_2)$ starts with a composition node, then g is the composition of two morphisms: $g = g_1 ⨟ g_2$, with $g_i = \left(\left[G_i\right], L_i, R_i, P_i, \left[F_i\right]\right)$. Given the partition of the vertices determined by g_1 and g_2, we can decompose g in another way, by writing $[G] = \left[\begin{pmatrix} \overline{G}_1 & C \\ 0 & \overline{G}_2 \end{pmatrix}\right]$ and $B = (L \mid R) = \begin{pmatrix} \overline{L}_1 & \overline{R}_1 \\ \overline{L}_2 & \overline{R}_2 \end{pmatrix}$. Then, we have that $\overline{G}_1 = G_1$, $\overline{L}_1 = L_1$, $P = P_2 \cdot P_1$, $C = R_1 \cdot L_2^\top$, $\overline{R}_1 = R_1 \cdot P_2^\top$, $\overline{L}_2 = L_2 \cdot P_1$, $\overline{R}_2 = R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top$, $\overline{G}_2 = G_2 + L_2 \cdot F_1 \cdot L_2^\top$, 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 $\overline{B}_1 := (\overline{L}_1 \mid \overline{R}_1 \mid C)$ and $\overline{B}_2 := (\overline{L}_2 \mid \overline{R}_2 \mid C^\top)$. In order to build an inductive rank decomposition of Γ, we need rank decompositions of $\overline{\Gamma}_i = (\left[\overline{G}_i\right], \overline{B}_i)$. We obtain these in three steps. Firstly, we apply induction to obtain inductive rank decompositions $\mathcal{R}(d_i)$ of $\Gamma_i = (\left[G_i\right], (L_i \mid R_i))$ such that $\mathsf{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\mathsf{wd}(d_i), \mathsf{rk}(L_i), \mathsf{rk}(R_i)\}$. Secondly, we apply Lemma 6.17 to obtain an inductive rank decomposition T_2' of $\Gamma_2' = (\left[G_2 + L_2 \cdot F_1 \cdot L_2^\top\right], (L_2 \mid R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top))$ such that $\mathsf{wd}(T_2') \leq \mathsf{wd}(\mathcal{R}(d_2))$. Lastly, we observe that $(\overline{R}_1 \mid C) = R_1 \cdot (P_2^\top \mid L_2^\top)$ and $(\overline{L}_2 \mid C^\top) = L_2 \cdot (P_1 \mid R_1^\top)$. Then we obtain that $\overline{B}_1 = (L_1 \mid R_1) \cdot \begin{pmatrix} \mathbb{1}_n & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & P_2^\top & L_2^\top \end{pmatrix}$ and $\overline{B}_2 = (L_2 \mid R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top) \cdot \begin{pmatrix} P_1 & \mathbb{0} & R_1^\top \\ \mathbb{0} & \mathbb{1}_m & \mathbb{0} \end{pmatrix}$, and we can apply Lemma 6.15 to get inductive rank decompositions T_i of $\overline{\Gamma}_i$ such that $\mathsf{wd}(T_1) \leq \mathsf{wd}(\mathcal{R}(d_1))$ and $\mathsf{wd}(T_2) \leq \mathsf{wd}(T_2') \leq \mathsf{wd}(\mathcal{R}(d_2))$.

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

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

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 $g_i = \left(\left[G_i\right], L_i, R_i, P_i, \left[F_i\right]\right) : n_i \to m_i$. By exlicitly computing the monoidal product, we obtain that $[G] = \left[\left(\begin{smallmatrix} G_1 & \mathbb{0} \\ \mathbb{0} & G_2 \end{smallmatrix}\right)\right]$, $L = \left(\begin{smallmatrix} L_1 & \mathbb{0} \\ \mathbb{0} & L_2 \end{smallmatrix}\right)$, $R = \left(\begin{smallmatrix} R_1 & \mathbb{0} \\ \mathbb{0} & R_2 \end{smallmatrix}\right)$, $P = \left(\begin{smallmatrix} P_1 & \mathbb{0} \\ \mathbb{0} & P_2 \end{smallmatrix}\right)$ and $F = \left(\begin{smallmatrix} F_1 & \mathbb{0} \\ \mathbb{0} & F_2 \end{smallmatrix}\right)$. By induction, we have inductive rank decompositions $\mathcal{R}(d_i)$ of $\Gamma_i := (\left[G_i\right], B_i)$, where $B_i = (L_i \mid R_i)$, of bounded width: $\mathsf{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\mathsf{wd}(d_i), \mathsf{rk}(L_i), \mathsf{rk}(R_i)\}$. Let $\overline{B}_1 := (L_1 \mid \mathbb{0}_{n_2} \mid R_1 \mid \mathbb{0}_{m_2} \mid \mathbb{0}_{k_2}) = B_1 \cdot \left(\begin{smallmatrix} \mathbb{1}_{n_1} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{m_1} & \mathbb{0} & \mathbb{0} \end{smallmatrix}\right)$ and $\overline{B}_2 := (\mathbb{0}_{n_1} \mid L_2 \mid \mathbb{0}_{m_1} \mid R_2 \mid \mathbb{0}_{k_1}) = B_2 \cdot \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1}_{n_2} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{m_2} & \mathbb{0} \end{smallmatrix}\right)$. By Lemma 6.15, we can obtain inductive rank decompositions T_i of $\overline{\Gamma}_i := (\left[G_i\right], \overline{B}_i)$ such that $\mathsf{wd}(T_i) \leq \mathsf{wd}(\mathcal{R}(d_i))$. If $k_1, k_2 > 0$, then we define $\mathcal{R}(d) := (T_1 — \Gamma — T_2)$, which is an inductive rank decomposition of Γ because $\overline{\Gamma}_i$ satisfy the conditions in Definition 6.4. If $k_1 = 0$, then $\Gamma = \overline{\Gamma}_2$ and we can define $\mathcal{R}(d) := T_2$. Similarly, if $k_2 = 0$, then $\Gamma = \overline{\Gamma}_1$ 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 $\mathsf{wd}(T_i) = 0$) using the inductive hypothesis and Lemma 6.15.

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

□

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 $g = \left([G], ¡, ¡, (\,), [(\,)]\right)$ be the corresponding morphism in* BGraph. *Then,* $\frac{1}{2} \cdot \mathsf{rwd}(G) \leq \mathsf{mwd}(g) \leq 2 \cdot \mathsf{rwd}(G)$.

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 C and, for a morphism $f : A \to B$, 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 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 $\mathbf{S} : \mathsf{C} \to \mathsf{D}$, for some monoidal category D.
   (b) Show that combining solutions $\mathbf{S}(f_1)$ and $\mathbf{S}(f_2)$ with the operations of the monoidal category 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 C is *functorial* if the mapping from morphisms to solutions is a monoidal functor $\mathbf{S} : \mathsf{C} \to \mathsf{D}$, for some monoidal category 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 $\mathsf{C}(A, B)$ of morphisms of a monoidal category C that respects the categorical structure: if $f \sim_{A,B} f'$ and $g \sim_{B,C} g'$, then $f ⨾ g \sim_{A,C} f' ⨾ g'$; 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 C by these equivalence relations gives a monoidal category $\mathsf{C}/\sim$ and a functor $\mathbf{Q} : \mathsf{C} \to \mathsf{C}/\sim$.*

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 $n \to 0$ in the category of cospans of relational structures $\mathsf{sStruct}_\tau$. This monoidal category is equivalent to the finitely presented prop sFrob_τ. In Section 4.3, morphisms $n \to 0$ in the category MGraph are interpreted as graphs with n labels. The monoidal category MGraph is equivalent to the finitely presented prop BGraph.

With these interpretations for morphisms in sFrob_τ and BGraph in mind, we define logical equivalence for morphisms in these two categories. Two morphisms $g = c : m \to G \leftarrow n : d$ and $g' = c' : m \to G' \leftarrow n : d'$ in $\mathsf{Cospan(UHGraph)}_*$ 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 $([G], L, R, P, [S]) : m \to n$ and $\left([G'], L', R', P, [S]\right) : m \to n$ in 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 sFrob_τ and 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 sFrob_τ and BGraph. More in detail, preservation by the disjoint union of relational structures corresponds to preservation by the monoidal product in sFrob_τ, while preservation by the disjoint union and fuse operations together gives preservation by compositions in sFrob_τ. Similarly, preservation by disjoint union of labelled graphs gives preservation by monoidal product in BGraph, while preservation by disjoint union, edge creation and bilinear product gives preservation by compositions in BGraph.

These considerations show that logical equivalence respects the structure of both monoidal categories sFrob_τ and BGraph, and we can apply Lemma 7.2 to obtain that MSO logical equivalence defines quotient categories $\mathsf{sFrob}_\tau / \equiv_{MSO}$ and $\mathsf{BGraph} / \equiv_{MSO}$, and monoidal functors $\mathbf{T} : \mathsf{sFrob}_\tau \to \mathsf{sFrob}_\tau / \equiv_{MSO}$ and $\mathbf{R} : \mathsf{BGraph} \to \mathsf{BGraph} / \equiv_{MSO}$.

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 S(f)
if d = (G) then
  | compute s := S(f) by brute force
else if d = (d_1 — ;_C — d_2) then
  | compute s_1 := MonoidalSolve(d_1)
  | compute s_2 := MonoidalSolve(d_2)
  | compute s := s_1 ;_{S(C)} s_2
else if d = (d_1 — ⊗ — d_2) then
  | compute s_1 := MonoidalSolve(d_1)
  | compute s_2 := MonoidalSolve(d_2)
  | compute s := s_1 ⊗ 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. Our result does not require this computational cost to be constant but at most linear in the size of the inputs. 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 $\mathsf{w} : \mathcal{A} \to \mathbb{N}$ is *compositional* if there is some function $c : \mathbb{N} \to \mathbb{N}$ such that:

1. computing $\mathbf{S}(f)$ takes $\mathcal{O}(c(\mathsf{w}(f)) \cdot \mathsf{w}(f))$;
2. for $f : A \to C$ and $g : C \to B$ in C, and given $s = \mathbf{S}(f)$ and $t = \mathbf{S}(g)$, computing the composition $s \mathbin{;}_{\mathbf{S}(C)} t$ along the object $\mathbf{S}(C)$ in D takes $\mathcal{O}(c(\mathsf{w}(C)) \cdot (\mathsf{w}(f) + \mathsf{w}(C) + \mathsf{w}(g)))$;
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 $\mathcal{O}(c(0) \cdot (\mathsf{w}(f) + \mathsf{w}(g)))$.

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 $\mathsf{C}_k(A, B)$ the set of morphisms $A \to B$ in C of monoidal width at most k together with a witness decomposition $d \in D_f$ of width at most k.

$$\mathsf{C}_k(A, B) := \{(f, d) : f \in \mathsf{C}(A, B) \text{ and } d \in D_f \text{ and } \mathsf{wd}(d) \leq k\}$$

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* $\mathbf{S}$ *on* $\mathsf{C}_k(A, B)$ *with a compositional algorithm is linear in* $\mathsf{size}(d)$. *Explicitly, given an optimal monoidal decomposition of* f, *computing* $\mathbf{S}(f)$ *takes* $\mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d))$, *for some* $c : \mathbb{N} \to \mathbb{N}$.

Proof. Let $d \in D_f$ be a monoidal decomposition of a morphism $f : A \to B$ with $\mathsf{wd}(d) \leq k$. We show by induction on d that running Algorithm 2 takes $\mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d))$.

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, $\mathsf{w}(f) =: \mathsf{wd}(d) \leq k$, and computing $\mathbf{S}(f)$ takes $\mathcal{O}(c(\mathsf{w}(f)) \cdot \mathsf{w}(f)) = \mathcal{O}(k \cdot c(k) \cdot 1)$ by Assumption 1.

Suppose that the first node is a composition, $d = (d_1 \text{—} \,⨾_C\, \text{—} d_2)$. Then, the widths of d_1 and d_2, and the weight of C are bounded by k because the width of d is: $\mathsf{wd}(d) := \max\{\mathsf{wd}(d_1), \mathsf{w}(C), \mathsf{wd}(d_2)\} \leq k$ by hypothesis. We apply Assumption 2, the induction hypothesis and Lemma 3.5, to bound the time complexity of computing $\mathbf{S}(f)$ as the composition $\mathbf{S}(f_1) ⨾_C \mathbf{S}(f_2)$ in D.

$$\begin{aligned}
&\mathcal{O}(c(\mathsf{w}(C)) \cdot (\mathsf{w}(f_1) + \mathsf{w}(C) + \mathsf{w}(f_2))) + \mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d_1)) + \mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d_2)) \\
&= \mathcal{O}(c(k) \cdot (k \cdot \mathsf{size}(d_1) + k + k \cdot \mathsf{size}(d_2)) + k \cdot c(k) \cdot \mathsf{size}(d_1) + k \cdot c(k) \cdot \mathsf{size}(d_2)) \\
&= \mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d))
\end{aligned}$$

Suppose that the first node is a monoidal product, $d = (d_1 \text{—} \otimes \text{—} d_2)$. Then, the widths of d_1 and d_2 are bounded by k because the width of d is: $\mathsf{wd}(d) := \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2)\} \leq k$ by hypothesis. We apply Assumption 3, the induction hypothesis and Lemma 3.5, to calculate the time complexity of computing $\mathbf{S}(F)$ as the monoidal product $\mathbf{S}(f_1) \otimes \mathbf{S}(f_2)$.

$$\begin{aligned}
&\mathcal{O}(c(0) \cdot (\mathsf{w}(f_1) + \mathsf{w}(f_2))) + \mathcal{O}(c(k) \cdot \mathsf{w}(f_1)) + \mathcal{O}(c(k) \cdot \mathsf{w}(f_2)) \\
&= \mathcal{O}(c(k) \cdot (\mathsf{w}(f_1) + \mathsf{w}(f_2))) \\
&= \mathcal{O}(c(k) \cdot (k \cdot \mathsf{size}(d_1) + k \cdot \mathsf{size}(d_2))) \\
&= \mathcal{O}(k \cdot c(k) \cdot \mathsf{size}(d))
\end{aligned}$$

□

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 Diag(E) with colimits. There is a functor that takes a diagram as inputs and returns and object of E, its colimit. However, to make this problem compositional, we need to lift this functor to discrete cospans.

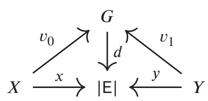
The graph |E| underlying the category E is an object of the category Graph_∞ of possibly infinite graphs and their homomorphisms. We consider the slice category $\mathsf{Graph}_\infty/|\mathsf{E}|$, where objects are diagrams $d : G \to |\mathsf{E}|$ and morphisms are commutative triangles. We restrict to finite diagrams, diagrams $d : G \to |\mathsf{E}|$ where the graph G is finite.

Definition 7.7. The category Diag(E) of diagrams in E is the full subcategory of $\mathsf{Graph}_\infty/|\mathsf{E}|$ on finite diagrams.

There is a functor $\mathrm{colim} : \mathsf{Diag}(\mathsf{E}) \to \mathsf{E}$ that assigns to each diagram d an object in E that is its colimit[1]. This functor is unique up to isomorphism. In order to decompose diagrams, we consider discrete cospans of them. A diagram $d : G \to |\mathsf{E}|$ is discrete if the graph G is discrete.

Definition 7.8. The category CDiag(E) is the full subcategory of Cospan(Diag(E)) on discrete cospans of diagrams in E.

Explicitly, objects are graph morphisms $X \to |\mathsf{E}|$, i.e. functions $X \to \mathsf{Obj}(\mathsf{E})$ 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* 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.*

Theorem 7.10 ([RSW08])**.** *There is a monoidal functor* $\mathbf{Colim} : \mathsf{CDiag} \to \mathsf{Cospan}(\mathsf{E})$ *from discrete cospans of diagrams to cospans in* E *that extends the colimit functor* $\mathrm{colim} : \mathsf{Diag}(\mathsf{E}) \to \mathsf{E}$.

The functor $\mathbf{Colim}$ makes the problem of computing colimits functorial. For some choices of the category E, we can show that there is a compositional algorithm for computing colimits.

Colimits in Set**.** Computing the colimit of a finite diagram $d : G \to |\mathsf{Set}|$ by brute force means to take the disjoint union of the sets $d(v)$ for $v \in \mathsf{vertices}(G)$ and then quotienting by the equivalence relation given by the edges of G.

$$\mathrm{colim}\, d = \left(\coprod_{v \in \mathsf{vertices}(G)} d(v) \right) \Big/ \sim$$

[1] Note that we are using the axiom of choice in this definition.

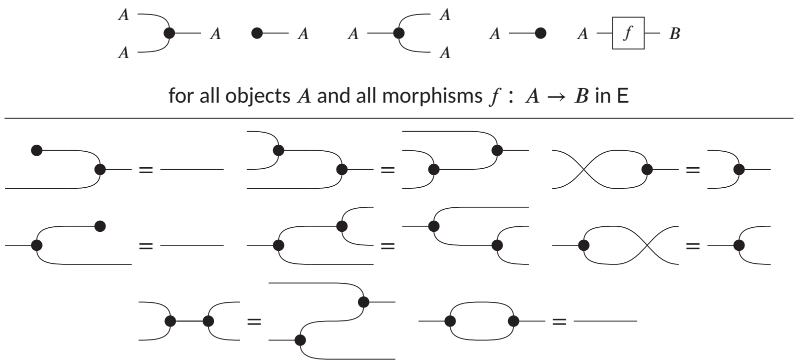


Figure 7.1

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 Set. The pushout $U +_Y V$ of $u : Y \to U$ and $v : Y \to V$ in 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 $\mathcal{O}(|Y|^5 + |U| + |V|) \leq \mathcal{O}(|Y|^5 \cdot (|U| + |V|))$ and we have shown Assumption 2.

As just mentioned, computing disjoint unions takes $\mathcal{O}(|U| + |V|)$, which satisfies Assumption 3 about computing monoidal products.

We have shown that colimits in Set 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 [C, Set] means computing the same colimit in Set for each object A in C and computing the corresponding unique morphism for every morphism $f : A \to B$ in C. When the category 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 $d : G \to |[\mathsf{C}, \mathsf{Set}]|$ in the presheaf category $[\mathsf{C}, \mathsf{Set}]$. 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 C, the diagram d in $[\mathsf{C}, \mathsf{Set}]$ determines a diagram $d_A : G \to |\mathsf{Set}|$ of the same shape in Set, defined on vertices by $d_A(v) := \mathbf{D}_v(A)$ and on edges by $d_A(e) := \gamma_e(A)$. For a functor $\mathbf{F} : \mathsf{C} \to \mathsf{Set}$, we let its cardinality to be the maximum cardinality of the sets in its image, $|\mathbf{F}| := \max_{A \in \mathsf{Obj}(\mathsf{C})} |\mathbf{F}(A)|$.

The colimit of d is computed component-wise: for every object A of C, we compute the colimit in Set of the diagram d_A, and, for every morphism $f : A \to B$ in C, we compute the unique colimit function $\operatorname{colim} f : \operatorname{colim} d_A \to \operatorname{colim} d_B$. The time complexity of computing $\operatorname{colim} d_A$ for each object A is $\mathcal{O}(n_A^5)$, where $n_A := \sum_{v \in \mathsf{vertices} G} |d_A(v)|$. As a consequence, computing all these colimits takes $\mathcal{O}(s \cdot n^5)$, where $n := \sum_{v \in \mathsf{vertices} G} |\mathbf{D}_v|$. The computation of the corresponding morphisms is irrelevant as it is linear in $s \cdot n$. When computing $\operatorname{colim} d$, we recorded all the injections $\iota_v^A : d_A(v) \to \operatorname{colim} d_A$, Then, for each object A of C and each vertex v of G, we define the colimit of f thanks to the universal property: $\operatorname{colim} f(\iota_v^A(x)) := \iota_v^B(\mathbf{D}_v f(x))$. The image on some elements in $\operatorname{colim} d_A$ is computed more than once, but, thanks to the universal property, all these values coincide and we have computed $\operatorname{colim} f$ going through at most $s \cdot n$ elements.

For compositions in $\mathsf{Cospan}([\mathsf{C}, \mathsf{Set}])$, we need to compute pushouts in $[\mathsf{C}, \mathsf{Set}]$. As explained above, we can reuse the complexity bounds for Set 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 $\mathcal{O}(s \cdot |\mathbf{Y}|^5(|\mathbf{U}| + |\mathbf{V}|))$. Similarly, monoidal products in $\mathsf{Cospan}([\mathsf{C}, \mathsf{Set}])$ correspond to coproducts in $[\mathsf{C}, \mathsf{Set}]$, and computing $\mathbf{U} + \mathbf{V}$ takes $\mathcal{O}(s \cdot (|\mathbf{U}| + |\mathbf{V}|))$.

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, pathwidth 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 Ingólfsdóttir, 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 Obtułowicz. “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.LO].

[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 Pawel Sobocinski. “Functorial semantics for partial theories”. In: *Proceedings of the ACM on Programing 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.LO].

[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 Fraıssé". 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 (1 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 isomorphisms 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 restreints, 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`.

[GR00] 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 Hliněnỳ, 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, Universita 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, December13–17, 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 λ-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. “Arity 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] Yoàv 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-O.

[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.

[RS04] 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.

[Şte86a] Gheorghe Ştefănescu. "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.

[Şte86b] Gheorghe Ştefănescu. *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.

Appendix A

Publications and academic activities

As customary in mathematics, all my publications list the authors in alphabetical order. Three of these publications [DHS21; DS22; DS23] contributed to the contents of this thesis. My contributions to these articles have been to write most of the prose, statements and proofs. Most of my publications concern topics that are too different from the topic of this thesis to be included in it coherently. I wrote the majority of the proofs in [DR23], while my coauthors did in [DFR22]. However, all the authors have contributed evenly to the development of the ideas present in both the articles. I contributed to the writing of the remaining articles [Di +21a; Di +23; Fel+21].

Talks

Aug 2023: Applied Category Theory 2023 conference (distinguished).
Jul 2023: Coresources workshop.
Jun 2023: Logic in Computer Science 2023 Conference.
Jun 2023: Quantitative Logic workshop.
Jun 2023: Categories Networking Project Workshop.
Apr 2023: Italian Category Theory fest.
Aug 2022: Women in Logic 2022 workshop.
Jul 2022: Applied Category Theory 2022 Conference.
Jun 2022: Foundational Methods in Computer Science workshop.
May 2022: Comonads Meetup Seminar.
May 2022: Mathematical Foundations Seminar.
Dec 2021: Symposium on Compositional Structures 8.
Jan 2021: Computer Science Logic 2021 conference.

Academic service

May 2023: Member of the executive board of the Compositionality Journal.
Feb 2023: Teaching assistant for the category theory and applications course.
Sep 2022: Local co-organiser of the Symposium on Compositional Structures 9.
May 2022: Program committee member of the Applied Category Theory Conference.
Sep 2021: Co-organiser of the Adjoint school 2022 and 2023.
May 2021: Teaching assistant for the introductory course on category theory.

Events

Nov 2022: Academic visit to Jamie Vicary at the University of Cambridge.

Aug 2022: Kleene Award and distinguished paper at LiCS2022.
Jun 2022: Academic visit to Fabio Gadducci at the University of Pisa.
Mar 2020: Participant in the Adjoint school supervised by Valeria de Paiva.

List of publications

[1] 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].

[2] 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.

[3] 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.

[4] 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.LO].

[5] 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.

[6] 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].

[7] 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].

[8] Giovanni de Felice, Elena Di Lavore, Mario Román, and Alexis Toumi. "Functorial Language Games for Question Answering". In: *Electronic Proceedings in Theoretical Computer Science*. Vol. 333. Open Publishing Association, Feb. 2021, pp. 311–321. DOI: 10.4204/eptcs.333.21.

Author's contributions to publications

We include the publications whose content is featured in this thesis. As customary in mathematics, authors are listed in alphabetical order, and papers are assumed to be equal collaborations between all of the listed authors.

Article 1

[DS23]: The author identified the significance of the work, wrote the statements and proofs of all the results, prepared the figures, and wrote the manuscript.

[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.

Article 2

[DS22]: The author wrote the statements and proofs of all the results, prepared the figures and wrote the manuscript.

[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.

Article 3

[DHS21]: The author wrote the statements and proofs of all the results, included concrete examples, prepared the figures and wrote the manuscript.

[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].

Logical Methods in Computer Science
Volume 19, Issue 3, 2023, pp. 15:1–15:46
https://lmcs.episciences.org/

Submitted Dec. 28, 2022
Published Sep. 04, 2023

MONOIDAL WIDTH

ELENA DI LAVORE AND PAWEŁ SOBOCIŃSKI

Tallinn University of Technology, Tallinn, Estonia

Abstract. We introduce monoidal width as a measure of complexity for morphisms in monoidal categories. Inspired by well-known structural width measures for graphs, like tree width and rank width, monoidal width is based on a notion of syntactic decomposition: a monoidal decomposition of a morphism is an expression in the language of monoidal categories, where operations are monoidal products and compositions, that specifies this morphism. Monoidal width penalises the composition operation along "big" objects, while it encourages the use of monoidal products. We show that, by choosing the correct categorical algebra for decomposing graphs, we can capture tree width and rank width. For matrices, monoidal width is related to the rank. These examples suggest monoidal width as a good measure for structural complexity of processes modelled as morphisms in monoidal categories.

Contents

Key words and phrases: monoidal categories, tree width, rank width.

This research was supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001). The second author was additionally supported by the Estonian Research Council grant PRG1210.

DOI:10.46298/LMCS-19(3:15)2023

1. Introduction

In recent years, a current of research has emerged with focus on the interaction of *structure* — especially algebraic, using category theory and related subjects — and *power*, that is algorithmic and combinatorial insights stemming from graph theory, game theory and related subjects. Recent works include [ADW17, AS21, MS22].

The algebra of monoidal categories is a fruitful source of *structure* — it can be seen as a general process algebra of concurrent processes, featuring a sequential (;) as well as a parallel (⊗) composition. Serving as a process algebra in this sense, it has been used to describe artefacts of a computational nature as *arrows* of appropriate monoidal categories. Examples include Petri nets [FS18], quantum circuits [CK17, DKPvdW20], signal flow graphs [FS18, BSZ21], electrical circuits [CK22, BS21], digital circuits [GJL17], stochastic processes [Fri20, CJ19] and games [GHWZ18].

Given that the algebra of monoidal categories has proved its utility as a language for describing computational artefacts in various applications areas, a natural question is to examine its relationship with *power*: can monoidal structure help us to design efficient algorithms? To begin to answer this question, let us consider a mainstay of computer science: *divide-and-conquer* algorithms. Such algorithms rely on the internal geometry of the global artefact under consideration to ensure the ability to *divide*, that is, decompose it consistently into simpler components, inductively compute partial solutions on the components, and then recombine these local results to obtain a global solution.

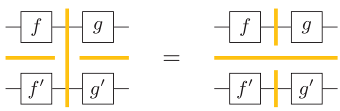


Figure 1. This morphism can be decomposed in two different ways: $(f \otimes f') \,;(g \otimes g') = (f \,; g) \otimes (f' \,; g')$.

Let us now return to systems described as arrows of monoidal categories. In applications, the parallel (⊗) composition typically means placing systems side-by-side with no explicit interconnections. On the other hand, the sequential (;) composition along an object typically means communication, resource sharing or synchronisation, the complexity of which is determined by the object along which the composition is performed. Based on examples in the literature, our basic motivating intuition is:

> *An algorithmic problem on an artefact that is a '⊗' lends itself to a divide-and-conquer approach more easily than one that is a ';'.*

Moreover, the "size" of the object along which the ';' occurs matters; typically the "larger" the object, the more work is needed in order to recombine results in any kind of divide-and-conquer approach. An example is compositional reachability checking in Petri nets of

Rathke et. al. [RSS14]: calculating the sequential composition is exponential in the size of the boundary. Another recent example is the work of Master [Mas22] on a compositional approach to calculating shortest paths.

On the other hand, (monoidal) category theory equates different descriptions of systems. Consider what is known as middle-four interchange, illustrated in Figure 1. Although monoidal category theory asserts that $(f \otimes f') \,;(g \otimes g') = (f \,; g) \otimes (f' \,; g')$, considering the two sides of the equations as decomposition blueprints for a divide-and-conquer approach, the right-hand side of the equation is clearly preferable since it maximises parallelism by minimising the size of the boundary along which composition occurs. This, roughly speaking, is the idea of *width* – expressions in the language of monoidal categories are assigned a natural number that measures "how good" they are as decomposition blueprints. The *monoidal width* of an arrow is then the width of its most efficient decomposition. In concrete examples, arrows with low width lend themselves to efficient divide-and-conquer approaches, following a width-optimal expression as a decomposition blueprint.

The study of efficient decompositions of combinatorial artefacts is well-established, especially in graph theory. A number of *graph widths* — by which we refer to related concepts like tree width, path with, branch width, cut width, rank width or twin width — have become known in computer science because of their relationship with algorithmic properties. All of them share a similar basic idea: in each case, a specific notion of legal decomposition is priced according to the most expensive operation involved, and the price of the cheapest decomposition is the width.

Perhaps the most famous of these is tree width, a measure of complexity for graphs that was independently defined by different authors [BB73, Hal76, RS86]. Every nonempty graph has a tree width, which is a natural number. Intuitively, a tree decomposition is a recipe for decomposing a graph into smaller subgraphs that form a tree shape. These subgraphs, when some of their vertices are identified, need to compose into the original graph, as shown in Figure 2. Courcelle's theorem

> *Every property expressible in the monadic second order logic of graphs can be verified in linear time on graphs with bounded tree width.*

is probably the best known among several results that establish links with algorithms [Bod92, BK08, Cou90] thus illustrating its importance for computer science.

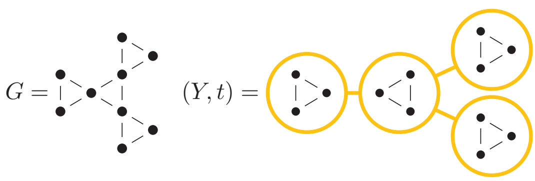
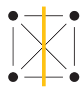


FIGURE 2. A tree decomposition cuts the graph along its vertices.

Another important measure is rank width [OS06] — a relatively recent development that has attracted significant attention in the graph theory community. A rank decomposition is a recipe for decomposing a graph into its single-vertex subgraphs by cutting along edges. The cost of a cut is the rank of the adjacency matrix that represents it, as illustrated in Figure 3. An intuition for rank width is that it is a kind of "Kolmogorov complexity" for graphs, with higher rank widths indicating that the connectivity data of the graph cannot be easily compressed. For example, while the family of cliques has unbounded tree width, their connectivity rather simple: in fact, all cliques have rank width 1.



$$\mathsf{rk}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix} = 1$$

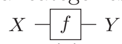
Figure 3. A cut and its matrix in a rank decomposition.

Contribution. Building on our conference paper [DLS22], our goals are twofold. Firstly, to introduce the concept of monoidal width and begin to develop techniques for reasoning about it.

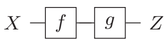
Before describing concrete, technical contributions, let us take a bird's eye view. It is natural for the seasoned researcher to be sceptical of a new abstract framework that seeks to generalise known results. The best abstract approaches *(i)* simplify existing known arguments, *(ii)* clean up the research landscape by connecting existing notions, or *(iii)* introduce techniques that allow one to prove new theorems. This paper does not (yet) bring strong arguments in favour of monoidal width if one uses these three points as yardsticks. Our high-level, conceptual contribution is, instead, the fact that the algebra of monoidal categories – already used in several contexts in theoretical computer science – is a multi-purpose algebra for specifying decompositions of graph-like structures important for computer scientists. There are several ways of making this work, and making these monoidal *algebras* of "open graphs" explicit as *monoidal categories* is itself a valuable endeavour. Indeed, identifying a monoidal category automatically yields a particular notion of decomposition: the instance of monoidal width in the monoidal category of interest. This point of view therefore demystifies ad hoc notions of decomposition that accompany each notion of width that we consider in this paper. Moreover, having an explicit algebra is also useful because it suggests a data structure — the expression in the language of monoidal categories — as a way of describing decompositions.

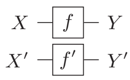
The results in this paper can be seen as a "sanity check" of these general claims, but can also be seen as taking the first technical steps in order to build towards points *(i)-(iii)* of the previous paragraph. To this end we examine monoidal width in the presence of common structure, such as coherent comultiplication on objects, and in a foundational setting such as the monoidal category of matrices. Secondly, connecting this approach with previous work, to examine graph widths through the prism of monoidal width. The two widths we focus on are tree width and rank width. We show that both can be seen as instances of monoidal width. The interesting part of this endeavour is identifying the monoidal category, and thus the relevant "decomposition algebra" of interest.

Unlike the situation with graph widths, it does not make sense to talk about monoidal width per se, since it is dependent on the choice of underlying monoidal category and thus a particular "decomposition algebra". The decomposition algebras that underlie tree and rank decompositions reflect their intuitive understanding. For tree width, this is a cospan category whose morphisms represent graphs with vertex interfaces, while for rank width it is a category whose morphisms represent graphs with edge interfaces, with adjacency matrices playing the role of tracking connectivity information within a graph. We show that the monoidal width of a morphism in these two categories is bounded, respectively, by the branch (Theorem 3.34) and rank width (Theorem 5.26) of the corresponding graph. In the first instance, this is enough to establish the connection between monoidal width and tree width, given that it is known that tree width and branch width are closely related. A small technical innovation is the definition of intermediate inductive notions of branch (Definition 3.14) and rank (Definition 5.7) decompositions, equivalent to the original definitions via "global" combinatorial notions of graph decomposition. The inductive presentations are closer in spirit to the inductive definition of monoidal decomposition, and allow us to give direct proofs of the main correspondences.

String diagrams. String diagrams [JS91] are a convenient syntax for monoidal categories, where a morphism $f: X \to Y$ is depicted as a box with input and output wires: 

. Morphisms in monoidal categories can be composed sequentially, using the composition of the category, and in parallel, using the monoidal structure. These two kinds of composition are reflected in the string diagrammatic syntax: the sequential composition $f \,;\, g$ is depicted by connecting the output wire of f with the input wire of g; the parallel composition $f \otimes f'$ is depicted by writing f on top of f'.

$f \,;\, g =$ 

$f \otimes f' =$ 

The advantage of this syntax is that all coherence equations for monoidal categories are trivially true when written with string diagrams. An example is the middle-four interchange law $(f \otimes f') \,;\, (g \otimes g') = (f \,;\, g) \otimes (g \,;\, g')$. These two expressions have one representation in terms of string diagrams, as shown in Figure 1. The coherence theorem for monoidal categories [Mac78] ensures that string diagrams are a sound and complete syntax for morphisms in monoidal categories.

Related work. This paper contains the results of [DLS21] and [DLS22] with detailed proofs. We generalise the results of [DLS21] to undirected hypergraphs and provide a syntactic presentation of the subcategory of the monoidal category of cospans of hypergraphs on discrete objects.

Previous syntactical approaches to graph widths are the work of Pudlák, Rödl and Savickỳ [PRS88] and the work of Bauderon and Courcelle [BC87]. Their works consider different notions of graph decompositions, which lead to different notions of graph complexity. In particular, in [BC87], the cost of a decomposition is measured by counting *shared names*, which is clearly closely related to penalising sequential composition as in monoidal width. Nevertheless, these approaches are specific to particular, concrete notions of graphs, whereas our work concerns the more general algebraic framework of monoidal categories.

Abstract approaches to width have received some attention recently, with a number of diverse contributions. Blume et. al. [BBFK11], similarly to our work, use (the category of) cospans of graphs as a formal setting to study graph decompositions: indeed, a major insight of loc. cit. is that tree decompositions are tree-shaped diagrams in the cospan category, and the original graph is reconstructed as a colimit of such a diagram. Our approach is more general, however, emphasising the relevance of the algebra of monoidal categories, of which cospan categories are just one family of examples.

The literature on comonads for game semantics characterises tree and path decompositions of relational structures (and graphs in particular) as coalgebras of certain comonads [ADW17, AS21, MS22, AM21, CD21]. Bumpus and Kocsis [BK21, Bum21] and, later, Bumpus, Kocsis and Master [BKM23] also generalise tree width to the categorical setting, although their approach is conceptually and technically removed from ours. Their work takes a combinatorial perspective on decompositions, following the classical graph theory literature. Given a shape of decomposition, called the spine in [BK21], a decomposition is defined globally as a functor out of that shape. This generalises the characterisation of tree width based on Halin's S-functions [Hal76]. In contrast, monoidal width is algebraic in flavour, following Bauderon and Courcelle's insights on tree decompositions [BC87]. Monoidal decompositions are syntax trees defined inductively and rely on the decomposition algebra given by monoidal categories.

Synopsis. The definition of monoidal width is introduced in Section 2, together with a worked out example. In Section 3 we recover tree width by instantiating monoidal width in a suitable category of cospans of hypergraphs. We recall it in Section 3.3 and provide a syntax for it in Section 3.4. Similarly, in Section 5 we recover rank width by instantiating monoidal width in a prop of graphs with boundaries where the connectivity information is stored in adjacency matrices, which we recall in Section 5.3. This motivates us to study monoidal width for matrices over the natural numbers in Section 4.

2. Monoidal width

We introduce monoidal width, a notion of complexity for morphisms in monoidal categories that relies on explicit syntactic *decompositions*, relying on the algebra of monoidal categories. We then proceed with a simple, yet useful examples of efficient monoidal decompositions in Section 2.1.

A monoidal decomposition of a morphism f is a binary tree where internal nodes are labelled with the operations of composition ; or monoidal product $\otimes$, and leaves are labelled with "atomic" morphisms. A decomposition, when evaluated in the obvious sense, results in f. We do not assume that the set of atomic morphisms $\mathcal{A}$ is minimal, they are merely morphisms that do not necessarily need to be further decomposed. We assume that $\mathcal{A}$ contains enough atoms to have a decomposition for every morphism. In most cases, we will take $\mathcal{A}$ to contain all the morphisms.

Definition 2.1 (Monoidal decomposition)**.** Let C be a monoidal category and $\mathcal{A}$ be a subset of its morphisms to which we refer as *atomic*. The set D_f of monoidal decompositions of $f\colon A \to B$ in C is defined inductively:

$$\begin{aligned} D_f \quad ::= \quad & (f) && \text{if } f \in \mathcal{A} \\ \mid \quad & (d_1 \text{—} \otimes \text{—} d_2) && \text{if } d_1 \in D_{f_1},\ d_2 \in D_{f_2} \text{ and } f =_{\mathsf{C}} f_1 \otimes f_2 \\ \mid \quad & (d_1 \text{—} ;_X \text{—} d_2) && \text{if } d_1 \in D_{f_1\colon A \to X},\ d_2 \in D_{f_2\colon X \to B} \text{ and } f =_{\mathsf{C}} f_1 \mathbin{;} f_2 \end{aligned}$$

In general, a morphism can be decomposed in different ways and decompositions that maximise parallelism are deemed more efficient. The monoidal width of a morphism is the cost of its cheapest monoidal decomposition.

Formally, each operation and atom in a decomposition is assigned a weight that will determine the cost of the decomposition. This is captured by the concept of a *weight function*.

Definition 2.2. Let C be a monoidal category and let $\mathcal{A}$ be its atomic morphisms. A *weight function* for $(\mathsf{C}, \mathcal{A})$ is a function $\mathsf{w} \colon \mathcal{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{C}) \to \mathbb{N}$ such that $\mathsf{w}(X \otimes Y) = \mathsf{w}(X) + \mathsf{w}(Y)$, and $\mathsf{w}(\otimes) = 0$.

A prop is a strict symmetric monoidal category where objects are natural numbers and the monoidal product on them is addition. If C is a prop, then, typically, we let $\mathsf{w}(1) := 1$. The idea behind giving a weight to an object $X \in \mathsf{C}$ is that $\mathsf{w}(X)$ is the cost paid for composing along X.

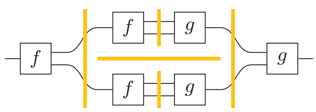
Definition 2.3 (Monoidal width)**.** Let w be a weight function for $(\mathsf{C}, \mathcal{A})$. Let f be in C and $d \in D_f$. The *width* of d is defined inductively as follows:

$$\begin{aligned} \mathsf{wd}(d) := {} & \mathsf{w}(f) && \text{if } d = (f) \\ & \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2)\} && \text{if } d = (d_1 \text{—} \otimes \text{—} d_2) \\ & \max\{\mathsf{wd}(d_1),\ \mathsf{w}(X),\ \mathsf{wd}(d_2)\} && \text{if } d = (d_1 \text{—} ;_X \text{—} d_2) \end{aligned}$$

The *monoidal width* of f is $\mathsf{mwd}(f) := \min_{d \in D_f} \mathsf{wd}(d)$.

Example 2.4. Let $f \colon 1 \to 2$ and $g \colon 2 \to 1$ be morphisms in a prop such that $\mathsf{mwd}(f) = \mathsf{mwd}(g) = 2$. The following figure represents the monoidal decomposition of $f ; (f \otimes f) ; (g \otimes g) ; g$ given by

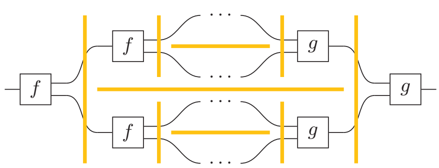
$$(f \text{—} ;_2 \text{—} (((f \text{—} ;_2 \text{—} g) \text{—} \otimes \text{—} (f \text{—} ;_2 \text{—} g)) \text{—} ;_2 \text{—} g)).$$



Indeed, taking advantage of string diagrammatic syntax, decompositions can be illustrated by enhancing string diagrams with additional annotations that indicate the order of decomposition. Throughout this paper, we use thick yellow dividing lines for this purpose.

Given that the width of a decomposition is the most expensive operation or atom, the above has width is 2 as compositions are along at most 2 wires.

Example 2.5. With the data of Example 2.4, define a family of morphisms $h_n \colon 1 \to 1$ inductively as $h_0 := f ;_2 g$, and $h_{n+1} := f ;_2 (h_n \otimes h_n) ;_2 g$.

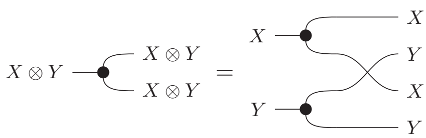


Each h_n has a decomposition of width 2^n where the root node is the composition along the middle wires. However — following the schematic diagram above — we have that $\mathsf{mwd}(h_n) \leq 2$ for any n.

2.1. **Monoidal width of copy.** Although monoidal width is a very simple notion, reasoning about it in concrete examples can be daunting because of the combinatorial explosion in the number of possible decompositions of any morphism. For this reason, it is useful to examine some commonly occurring structures that one encounters "in the wild" and examine their decompositions. One such situation is when the objects are equipped with a coherent comultiplication structure.

Definition 2.6. Let C be a symmetric monoidal category, with symmetries $\times_{X,Y} \colon X \otimes Y \to Y \otimes X$. We say that C has coherent copying if there is a class of objects $\Delta_{\mathsf{C}} \subseteq \mathsf{Obj}(\mathsf{C})$, satisfying

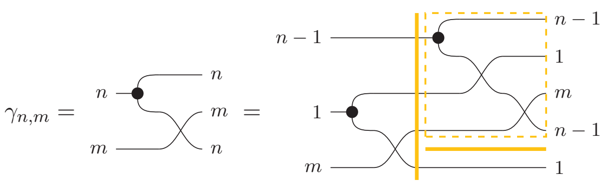
- $X, Y \in \Delta_{\mathsf{C}}$ iff $X \otimes Y \in \Delta_{\mathsf{C}}$;
- Every object $X \in \Delta_{\mathsf{C}}$ is endowed with a morphism $\multimap\!\!\!\subset_X \colon X \to X \otimes X$;
- For every $X, Y \in \Delta_{\mathsf{C}}$, $\multimap\!\!\!\subset_{X\otimes Y} = (\multimap\!\!\!\subset_X \otimes \multimap\!\!\!\subset_Y) \,;\, (\mathbb{1}_X \otimes \times_{X,Y} \otimes \mathbb{1}_Y)$ (coherence).



An example is any cartesian prop, where the copy morphisms are the universal ones given by the cartesian structure: $\multimap\!\!\!\subset_n := \langle \mathbb{1}_n, \mathbb{1}_n \rangle \colon n \to n + n$. For props with coherent copy, we assume that copy morphisms, symmetries and identities are atoms, $\multimap\!\!\!\subset_X, \times_{X,Y}, \mathbb{1}_X \in \mathcal{A}$, and that their weight is given by $\mathsf{w}(\multimap\!\!\!\subset_X) := 2 \cdot \mathsf{w}(X)$, $\mathsf{w}(\times_{X,Y}) := \mathsf{w}(X) + \mathsf{w}(Y)$ and $\mathsf{w}(\mathbb{1}_X) := \mathsf{w}(X)$.

Example 2.7. Let C be a prop with coherent copy and suppose that $1 \in \Delta_{\mathsf{C}}$. This implies that every $n \in \Delta_{\mathsf{C}}$ and there are copy morphisms $\multimap\!\!\!\subset_n \colon n \to 2n$ for all n. Let $\gamma_{n,m} := (\multimap\!\!\!\subset_n \otimes \mathbb{1}_m) \,;\, (\mathbb{1}_n \otimes \times_{n,m}) \colon n + m \to n + m + n$. We can decompose $\gamma_{n,m}$ in terms of $\gamma_{n-1,m+1}$ (in the dashed box), $\multimap\!\!\!\subset_1$ and $\times_{1,1}$ by cutting along at most $n + 1 + m$ wires:

$$\gamma_{n,m} = (\mathbb{1}_{n-1} \otimes ((\multimap\!\!\!\subset_1 \otimes \mathbb{1}_1) \,;\, (\mathbb{1}_1 \otimes \times_{1,1}))) \,;_{n+1+m}\, (g_{n-1,m+1} \otimes \mathbb{1}_1).$$



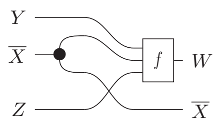
This allows us to decompose $\multimap\!\!\!\subset_n = \gamma_{n,0}$ cutting along only $n + 1$ wires. In particular, this means that $\mathsf{mwd}(\multimap\!\!\!\subset_n) \leq n + 1$.

The following lemma generalises the above example and is used in the proofs of some results in later sections, Proposition 3.30 and Proposition 4.6.

Lemma 2.8. *Let C be a symmetric monoidal category with coherent copying. Suppose that $\mathcal{A}$ contains $\multimap\!\!\!\subset_X$ for all $X \in \Delta_{\mathsf{C}}$, and $\times_{X,Y}$ and $\mathbb{1}_X$ for all $X \in \mathsf{Obj}(\mathsf{C})$. Let $\overline{X} := X_1 \otimes \cdots \otimes X_n$, with $X_i \in \Delta_{\mathsf{C}}$, $f \colon Y \otimes \overline{X} \otimes Z \to W$ and let $d \in D_f$. Let $\gamma_{\overline{X}}(f) :=$*

$(\mathbb{1}_Y \otimes \multimap\!\!\!\bullet\!\!<_{\overline{X}} \otimes \mathbb{1}_Z) \mathbin{;} (\mathbb{1}_{Y\otimes\overline{X}} \otimes \times_{\overline{X},Z}) \mathbin{;} (f \otimes \mathbb{1}_{\overline{X}})$.

$$\gamma_{\overline{X}}(f) \coloneqq$$

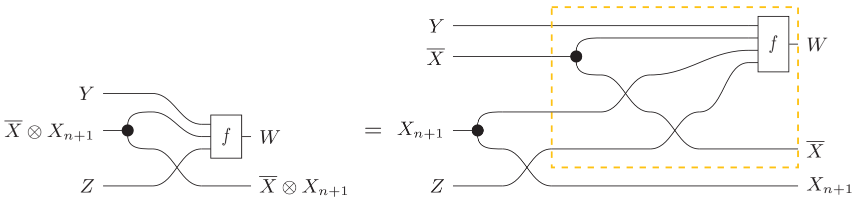


Then there is a monoidal decomposition $\mathcal{C}_{\overline{X}}(d)$ of $\gamma_{\overline{X}}(f)$ such that

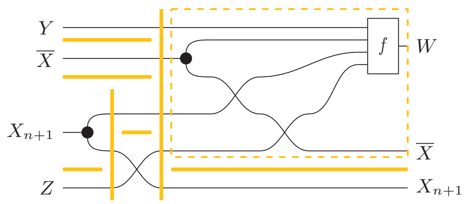
$$\mathsf{wd}(\mathcal{C}_{\overline{X}}(d)) \leq \max\{\mathsf{wd}(d), \mathsf{w}(Y) + \mathsf{w}(Z) + (n+1) \cdot \max_{i=1,\ldots,n} \mathsf{w}(X_i)\}.$$

Proof. Proceed by induction on the number n of objects being copied. If $n = 0$, then we are done because we keep the decomposition d and define $\mathcal{C}_I(d) \coloneqq d$.

Suppose that the statement is true for any $f' \colon Y \otimes \overline{X} \otimes Z' \to W$. Let $f \colon Y \otimes \overline{X} \otimes X_{n+1} \otimes Z \to W$. By coherence of $\multimap\!\!\!\bullet\!\!<$, we can rewrite $\gamma_{\overline{X}\otimes X_{n+1}}(f)$.



Let $\gamma_{\overline{X}}(f)$ be the morphism in the above dashed box. By the induction hypothesis, there is a monoidal decomposition $\mathcal{C}_{\overline{X}}(d)$ of $\gamma_{\overline{X}}(f)$ of bounded width: $\mathsf{wd}(\mathcal{C}_{\overline{X}}(d)) \leq \max\{\mathsf{wd}(d), \mathsf{w}(Y) + \mathsf{w}(X_{n+1} \otimes Z) + (n+1) \cdot \max_{i=1,\ldots,n} \mathsf{w}(X_i)\}$. We can use this decomposition to define a monoidal decomposition $\mathcal{C}_{\overline{X}\otimes X_{n+1}}(d)$ of $\gamma_{\overline{X}\otimes X_{n+1}}(f)$ as shown below.



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

$$\begin{aligned}
&\mathsf{wd}(\mathcal{C}_{\overline{X}\otimes X_{n+1}}(d)) \\
&= \max\left\{\mathsf{w}(Y) + \mathsf{w}(Z) + \mathsf{w}(X_{n+1}) + \sum_{i=1}^{n+1} \mathsf{w}(X_i), \mathsf{wd}(\mathcal{C}_{\overline{X}}(d))\right\} \\
&\leq \max\left\{\mathsf{w}(Y) + \mathsf{w}(Z) + (n+2) \cdot \max_{i=1,\ldots,n+1} \mathsf{w}(X_i), \mathsf{wd}(d),\right. \\
&\qquad \left.\mathsf{w}(Y) + \mathsf{w}(X_{n+1} \otimes Z) + (n+1) \cdot \max_{i=1,\ldots,n} \mathsf{w}(X_i)\right\}
\end{aligned}$$

$$= \max \left\{ \mathsf{w}(Y) + \mathsf{w}(Z) + (n+2) \cdot \max_{i=1,\ldots,n+1} \mathsf{w}(X_i), \mathsf{wd}(d) \right\} \qquad \square$$

3. A monoidal algebra for tree width

Our first case study is tree width of undirected hypergraphs. We show that monoidal width in a suitable monoidal category of hypergraphs is within constant factors of tree width. We rely on branch width, a measure equivalent to tree width, to relate the latter with monoidal width.

After recalling tree and branch width and the bounds between them in Section 3.1, we define the intermediate notion of inductive branch decomposition in Section 3.2 and show its equivalence to that of branch decomposition. Separating this intermediate step allows a clearer presentation of the correspondence between branch decompositions and monoidal decompositions. Section 3.3 recalls the categorical algebra of cospans of hypergraphs and Section 3.4 introduces a syntactic presentations of them. Finally, Section 3.5 contains the main result of the present section, which relates inductive branch decompositions, and thus tree decompositions, with monoidal decompositions.

Classically, tree and branch widths have been defined for finite undirected multihypergraphs, which we simply call hypergraphs. These have undirected edges that connect sets of vertices and they may have parallel edges.

Definition 3.1. A *(multi)hypergraph* $G = (V, E)$ is given by a finite set of vertices V, a finite set of edges E and an adjacency function $\mathsf{ends}\colon E \to \wp(V)$, where $\wp(V)$ indicates the set of subsets of V. A *subhypergraph* of G is a hypergraph $G' = (V', E')$ such that $V' \subseteq V$, $E' \subseteq E$ and $\mathsf{ends}'(e) = \mathsf{ends}(e)$ for all $e \in E'$.

Definition 3.2. Given two hypergraphs $G = (V, E)$ and $H = (W, F)$, a *hypergraph homomorphism* $\alpha\colon G \to H$ is given by a pair of functions $\alpha_V : V \to W$ and $\alpha_E\colon E \to F$ such that, for all edges $e \in E$, $\mathsf{ends}_H(\alpha_E(e)) = \alpha_V(\mathsf{ends}_G(e))$.

$$\begin{array}{ccc} E & \xrightarrow{f_E} & F \\ {\scriptstyle \mathsf{ends}_G}\downarrow & & \downarrow{\scriptstyle \mathsf{ends}_H} \\ \wp(V) & \xrightarrow{\wp(f_V)} & \wp(W) \end{array}$$

Hypergraphs and hypergraph homomorphisms form a category $\mathsf{UHGraph}$, where composition and identities are given by component-wise composition and identities.

Note that the category $\mathsf{UHGraph}$ is not the functor category $[\{\bullet \to \bullet\}, \mathsf{kl}(\wp)]$: their objects coincide but the morphisms are different.

Definition 3.3. The *hyperedge size* of a hypergraph G is defined as $\gamma(G) \coloneqq \max_{e \in \mathsf{edges}(G)} |\mathsf{ends}(e)|$. A *graph* is a hypergraph with hyperedge size 2.

Definition 3.4. A *neighbour* of a vertex v is a vertex w distinct from v with an edge e such that $v, w \in \mathsf{ends}(e)$. A *path* in a hypergraph is a sequence of vertices $(v_1, \ldots, v_n)$ such that, for every $i = 1, \ldots, n-1$, v_i and v_{i+1} are neighbours. A *cycle* in a hypergraph is a path where the first vertex v_1 coincides with the last vertex v_n. A hypergraph is *connected* if there is a path between every two vertices. A *tree* is a connected acyclic hypergraph. A tree is *subcubic* if every vertex has at most three neighbours.

Definition 3.5. The set of *binary trees* with labels in a set Λ is either: a leaf (λ) with label $\lambda \in \Lambda$; or a label $\lambda \in \Lambda$ with two binary trees T_1 and T_2 with labels in Λ, $(T_1 — \lambda — T_2)$.

3.1. **Background: tree width and branch width.** Intuitively, tree width measures "how far" a hypergraph G is from being a tree: a hypergraph is a tree iff it has tree width 1. Hypergraphs with tree widths larger than 1 are not trees; for example, the family of cliques has unbounded tree width.

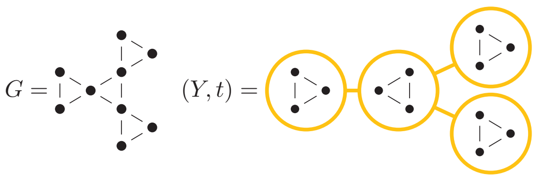
The definition relies on the concept of a *tree decomposition*. For Robertson and Seymour [RS86], a decomposition is itself a tree Y, each vertex of which is associated with a subhypergraph of G. Then G can be reconstructed from Y by identifying some vertices.

Definition 3.6 [RS86]**.** A *tree decomposition* of a hypergraph $G = (V, E)$ is a pair (Y, t) where Y is a tree and $t\colon \mathsf{vertices}(Y) \to \wp(V)$ is a function such that:

(1) Every vertex is in one of the components: $\bigcup_{i \in \mathsf{vertices}(Y)} t(i) = V$.
(2) Every edge has its endpoints in a component: $\forall e \in E\ \exists i \in \mathsf{vertices}(Y)\ \mathsf{ends}(e) \subseteq t(i)$.
(3) The components are glued in a tree shape: $\forall i, j, k \in \mathsf{vertices}(Y)\ i \rightsquigarrow j \rightsquigarrow k \Rightarrow t(i) \cap t(k) \subseteq t(j)$.

The cost is the maximum number of vertices of the component subhypergraphs.

Example 3.7. Consider the hypergraph G and its tree decomposition (Y, t) below. Its cost is 3 as its biggest component has three vertices.



Definition 3.8 (Tree width)**.** Given a tree decomposition (Y, t) of a hypergraph G, its width is $\mathsf{wd}(Y, t) := \max_{i \in \mathsf{vertices}(Y)} |t(i)|$. The tree width of G is given by the min-max formula:

$$\mathsf{twd}(G) := \min_{(Y,t)} \mathsf{wd}(Y, t).$$

Note that Robertson and Seymour subtract 1 from $\mathsf{twd}(G)$ so that trees have tree width 1. To minimise bureaucratic overhead, we ignore this convention.

We use branch width [RS91] as a technical stepping stone to relate monoidal width and tree width. Before presenting its definition, it is important to note that branch width and tree width are *equivalent*, i.e. they are within a constant factor of each other.

Theorem 3.9 [RS91, Theorem 5.1]**.** *Branch width is equivalent to tree width. More precisely, for a hypergraph $G = (V, E)$,*

$$\max\{\mathsf{bwd}(G), \gamma(G)\} \leq \mathsf{twd}(G) \leq \max\{\frac{3}{2}\mathsf{bwd}(G), \gamma(G), 1\}.$$

Branch width relies on branch decompositions, which, intuitively, record in a tree a way of iteratively partitioning the edges of a hypergraph.

Definition 3.10 [RS91]**.** A *branch decomposition* of a hypergraph $G = (V, E)$ is a pair (Y, b) where Y is a subcubic tree and $b \colon \mathsf{leaves}(Y) \cong E$ is a bijection.

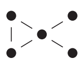
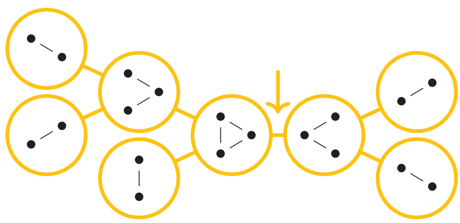
Each edge e in the tree Y determines a splitting of the hypergraph. More precisely, it determines a two partition of the leaves of Y, which, through b, determines a 2-partition $\{A_e, B_e\}$ of the edges of G. This corresponds to a splitting of the hypergraph G into two subhypergraphs G_1 and G_2. Intuitively, the order of an edge e is the number of vertices that are glued together when joining G_1 and G_2 to get 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 A_e and an edge in B_e that are both adjacent to v.

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

Definition 3.11 (Branch width)**.** Given a branch decomposition (Y, b) of a hypergraph $G = (V, E)$, define its width as $\mathsf{wd}(Y, b) := \max_{e \in \mathsf{edges}(Y)} \mathsf{ord}(e)$.

The branch width of G is given by the min-max formula: $\mathsf{bwd}(G) := \min_{(Y,b)} \mathsf{wd}(Y, b)$.

Example 3.12. If we start reading the decomposition from an edge in the tree Y, we can extend the labelling to internal vertices by labelling them with the glueing of the labels of their children.

$G =$  $(Y, b) =$ 

In this example, 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.

3.2. **Hypergraphs with sources and inductive definition.** We introduce a definition of decomposition that is intermediate between a branch decomposition and a monoidal decomposition. It adds to branch decompositions the algebraic flavour of monoidal decompositions by using an inductive data type, that of binary trees, to encode a decomposition.

Our approach follows closely Bauderon and Courcelle's hypergraphs with sources [BC87] and the corresponding inductive definition of tree decompositions [Cou92]. Courcelle's result [Cou92, Theorem 2.2] is technically involved as it translates between a combinatorial description of a decomposition to a syntactic one. Our results in this and the next sections are similarly technically involved.

We recall the definition of hypergraphs with sources and introduce inductive branch decompositions of them. Intuitively, the sources of a hypergraph are marked vertices that are allowed to be "glued" together with the sources of another hypergraph. Thus, the equivalence between branch decompositions and inductive branch decompositions formalises the intuition that a branch decomposition encodes a way of dividing a hypergraph into smaller subgraphs by "cutting" along some vertices.

Definition 3.13 [BC87]**.** A *hypergraph with sources* is a pair $\Gamma = (G, X)$ where $G = (V, E)$ is a hypergraph and $X \subseteq V$ is a subset of its vertices, called the sources (Figure 4). Given two hypergraphs with sources $\Gamma = (G, X)$ and $\Gamma' = (G', X')$, we say that Γ' is a subhypergraph of Γ whenever G' is a subhypergraph of G.

Note that the sources of a subhypergraph Γ' of Γ need not to appear as sources of Γ, nor vice versa. In fact, if Γ is obtained by identifying all the sources of Γ_1 with some of the sources of Γ_2, the sources of Γ and Γ_1 will be disjoint.

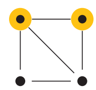


FIGURE 4. Sources are marked vertices in the graph and are thought of as an interface that can be glued with that of another graph.

An inductive branch decomposition is a binary tree whose vertices carry subhypergraphs Γ' of the ambient hypergraph Γ. This set of all such binary trees is defined as follows

$$T_\Gamma \ ::= \ () \ \mid \ (T_\Gamma, \Gamma', T_\Gamma)$$

where Γ' ranges over the non-empty subhypergraphs of Γ. An inductive branch decomposition has to satisfy additional conditions that ensure that "glueing" Γ_1 and Γ_2 together yields Γ.

Definition 3.14. Let $\Gamma = ((V, E), X)$ be a hypergraph with sources. An *inductive branch decomposition* of Γ is $T \in T_\Gamma$ where either:

- Γ is discrete (i.e. it has no edges) and $T = ()$;
- Γ has one edge and $T = (()—\Gamma—())$. We will use the shorthand $T = (\Gamma)$ in this case;
- $T = (T_1—\Gamma—T_2)$ and $T_i \in T_{\Gamma_i}$ are inductive branch decompositions of subhypergraphs $\Gamma_i = ((V_i, E_i), X_i)$ of Γ 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, $X_i = (V_1 \cap V_2) \cup (X \cap V_i)$.

Note that $\mathsf{ends}(E_i) \subseteq V_i$ 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 and indicate with $\lambda(T')$ the subhypergraph of Γ that T' is a decomposition of. We sometimes write $\Gamma_i = \lambda(T_i)$, $V_i = \mathsf{vertices}(\Gamma_i)$ and $X_i = \mathsf{sources}(\Gamma_i)$. Then,

$$\mathsf{sources}(\Gamma_i) = (\mathsf{vertices}(\Gamma_1) \cap \mathsf{vertices}(\Gamma_2)) \cup (\mathsf{sources}(\Gamma) \cap \mathsf{vertices}(\Gamma_i)). \tag{3.1}$$

Definition 3.15. Let $T = (T_1—\Gamma—T_2)$ be an inductive branch decomposition of $\Gamma = (G, X)$, with T_i possibly both empty. Define the *width* of T inductively: $\mathsf{wd}(()) := 0$, and $\mathsf{wd}(T) := \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2), |\mathsf{sources}(\Gamma)|\}$. Expanding this expression, we obtain

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

The *inductive branch width* of Γ is defined by the min-max formula $\mathsf{ibwd}(\Gamma) := \min_T \mathsf{wd}(T)$.

We show that this definition is equivalent to the original one by exhibiting a mapping from branch decompositions to inductive branch decompositions that preserve the width and vice versa. Showing that these mappings preserve the width is a bit involved because the order of the edges in a decomposition is defined "globally", while, for an inductive decomposition, the width is defined inductively. Thus, we first need to show that we can compute the inductive width globally.

Lemma 3.16. *Let $\Gamma = (G, X)$ be a hypergraph with sources and T be an inductive branch decomposition of Γ. Let T_0 be a full subtree of T and let $T' \not\gtrless T_0$ denote a full subtree T' of T such that its intersection with T_0 is empty. Then,*

$$\mathsf{sources}(\lambda(T_0)) = \mathsf{vertices}(\lambda(T_0)) \cap \left(X \cup \bigcup_{T' \not\gtrless T_0} \mathsf{vertices}(\lambda(T')) \right).$$

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

If $T = (T_1 \text{—} \Gamma \text{—} T_2)$, 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 boundaries coincide and the statement is satisfied because $\mathsf{sources}(\lambda(T_0)) = X = V \cap X$. Now suppose that T_0 is a full subtree of T_1. Then, by applying the induction hypothesis, Equation (3.1), and using the fact that $\lambda(T_0) \subseteq \lambda(T_1)$, we compute the sources of T_0:

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

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

Lemma 3.17. *Let $\Gamma = (G, X)$ be a hypergraph with sources and $G = (V, E)$ be its underlying hypergraph. Let T be an inductive branch decomposition of Γ. Then, there is a branch decomposition $\mathcal{I}^\dagger(T)$ of G such that $\mathsf{wd}(\mathcal{I}^\dagger(T)) \leq \mathsf{wd}(T)$.*

Proof. A binary tree is, in particular, a subcubic tree. Then, we can define Y to be the unlabelled tree underlying T. The label of a leaf l of T is a subhypergraph of Γ with one edge e_l. Then, there is a bijection $b \colon \mathsf{leaves}(T) \to \mathsf{edges}(G)$ 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, $e \in \mathsf{edges}(Y)$ if and only if $e \in \mathsf{edges}(T)$. Let $\{v, w\} = \mathsf{ends}(e)$ 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, $E_w = \mathsf{edges}(\lambda(T_w))$ and $E_v = \bigcup_{T' \not\gtrless T_w} \mathsf{edges}(\lambda(T'))$, and, for the vertices sets, $\mathsf{ends}(E_w) \subseteq \mathsf{vertices}(\lambda(T_w))$ and

$\mathsf{ends}(E_v) \subseteq \bigcup_{T' \nleq T_w} \mathsf{vertices}(\lambda(T'))$. Using these inclusions and applying Lemma 3.16,

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

□

Lemma 3.18. *Let $\Gamma = (G, X)$ be a hypergraph with sources and $G = (V, E)$ be its underlying hypergraph. Let (Y, b) be a branch decomposition of G. Then, there is a branch decomposition $\mathcal{I}(Y, b)$ of Γ such that $\mathsf{wd}(\mathcal{I}(Y, b)) \leq \mathsf{wd}(Y, b) + |X|$.*

Proof. Proceed by induction on $|\mathsf{edges}(Y)|$. 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 $\mathsf{wd}(\mathcal{I}(Y, b)) := |X| \leq \mathsf{wd}(Y, b) + |X|$.

If Y has at least one edge e, then $Y = Y_1 \overset{e}{—} Y_2$ with Y_i a subcubic tree. Let $E_i = b(\mathsf{leaves}(Y_i))$ 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 $V_1 := \mathsf{ends}(E_1)$, $V_2 := \mathsf{ends}(E_2) \cup (V \setminus V_1)$, $X_i := (V_1 \cap V_2) \cup (V_i \cap X)$ 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 Γ and, by applying Lemma 3.16,

$$\begin{aligned}
&\mathsf{wd}(\mathcal{I}(Y, b)) \\
&:= \max\{\mathsf{wd}(T_1), |X|, \mathsf{wd}(T_2)\} \\
&= \max_{T' \leq T} |\mathsf{sources}(\lambda(T'))| \\
&\leq \max_{T' \leq T} |\mathsf{vertices}(\lambda(T')) \cap \mathsf{ends}(E \setminus \mathsf{edges}(\lambda(T')))| + |X| \\
&= \max_{e \in \mathsf{edges}(Y)} \mathsf{ord}(e) + |X| \\
&=: \mathsf{wd}(Y, b) + |X|
\end{aligned}$$

□

Combining Lemma 3.17 and Lemma 3.18 we obtain:

Proposition 3.19. *Inductive branch width is equivalent to branch width.*

3.3. Cospans of hypergraphs.

We work with the category $\mathsf{UHGraph}$ of undirected hypergraphs and their homomorphisms (Definition 3.1). The monoidal category $\mathsf{Cospan}(\mathsf{UHGraph})$ of cospans is a standard choice for an algebra of "open" hypergraphs. Hypergraphs are composed by glueing vertices [RSW05, GH97, Fon15]. We do not need the full expressivity of $\mathsf{Cospan}(\mathsf{UHGraph})$ and restrict to $\mathsf{Cospan}(\mathsf{UHGraph})_*$, where the objects are sets, seen as discrete hypergraphs.

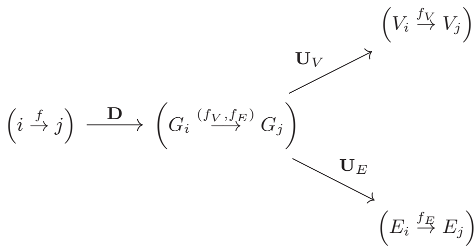
Definition 3.20. A *cospan* in a category C is a pair of morphisms in C that share the same codomain, called the *head*, $f \colon X \to E$ and $g \colon Y \to E$. When C has finite colimits, cospans form a symmetric monoidal category $\mathsf{Cospan}(\mathsf{C})$ whose objects are the objects of C

and morphisms are cospans in C. More precisely, a morphism $X \to Y$ in $\mathsf{Cospan}(\mathsf{C})$ is an equivalence class of cospans $X \xrightarrow{f} E \xleftarrow{g} Y$, up to isomorphism of the head of the cospan. The composition of $X \xrightarrow{f} E \xleftarrow{g} Y$ and $Y \xrightarrow{h} F \xleftarrow{l} Z$ is given by the pushout of g and h. Intuitively, the pushout of g and h "glues" E and F along the images of g and h (see Example 3.23). The monoidal product is given by component-wise coproducts.

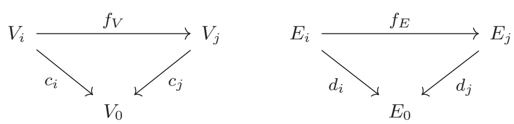
We can construct the category of cospans of hypergraphs $\mathsf{Cospan}(\mathsf{UHGraph})$ because the category of hypergraphs $\mathsf{UHGraph}$ has all finite colimits.

Proposition 3.21. *The category* $\mathsf{UHGraph}$ *has all finite colimits and they are computed pointwise.*

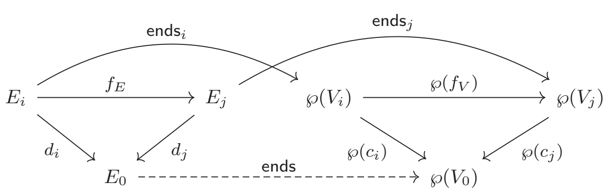
Proof. Let $\mathbf{D}\colon \mathsf{J} \to \mathsf{UHGraph}$ be a diagram in $\mathsf{UHGraph}$. Then, every object i in J determines a hypergraph $G_i \coloneqq \mathbf{D}(i) = (V_i, E_i, \mathsf{ends}_i)$ and every $f\colon i \to j$ in J, gives a hypergraph homomorphism $\mathbf{D}(f) = (f_V, f_E)$. Let the functors $\mathbf{U}_E\colon \mathsf{UHGraph} \to \mathsf{Set}$ and $\mathbf{U}_V\colon \mathsf{UHGraph} \to \mathsf{Set}$ associate the edges, resp. vertices, component to hypergraphs and hypergraph homomorphisms: for a hypergraph $G = (V, E)$, $\mathbf{U}_E(G) \coloneqq E$ and $\mathbf{U}_V(G) \coloneqq V$; and, for a morphism $f = (f_V, f_E)$, $\mathbf{U}_E(f) \coloneqq f_E$ and $\mathbf{U}_V(f) \coloneqq f_V$.



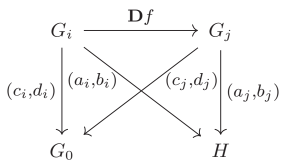
The category Set has all colimits, thus there are $E_0 \coloneqq \operatorname{colim}(\mathbf{D};\mathbf{U}_E)$ and $V_0 \coloneqq \operatorname{colim}(\mathbf{D};\mathbf{U}_V)$. Let $c_i\colon V_i \to V_0$ and $d_i\colon E_i \to E_0$ be the inclusions given by the colimits. Then, for any $i, j \in \mathsf{Obj}(\mathsf{J})$ the following diagrams commute:



By definition of hypergraph morphism, $f_E \,;\, \mathsf{ends}_j = \mathsf{ends}_i \,;\, \wp(f_V)$, and, by functoriality of $\wp$, $\wp(f_V) \,;\, \wp(c_j) = \wp(c_i)$. This shows that $\wp(V_0)$ is a cocone over $\mathbf{D} \,;\, \mathbf{U}_E$ with morphisms given by $\mathsf{ends}_i \,;\, \wp(c_i)$. Then, there is a unique morphism $\mathsf{ends}\colon E_0 \to \wp(V_0)$ that commutes with the cocone morphisms: $d_i \,;\, \mathsf{ends} = \mathsf{ends}_i \,;\, \wp(c_i)$.

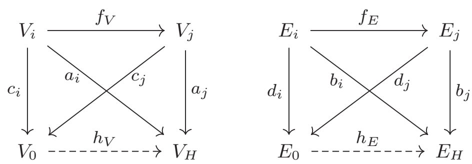


This shows that the pairs (c_i, d_i) are hypergraph morphisms and, with the hypergraph defined by $G_0 \coloneqq (V_0, E_0, \mathsf{ends})$, form a cocone over $\mathbf{D}$ in $\mathsf{UHGraph}$. Let $H = (V_H, E_H, \mathsf{ends}_H)$ be another cocone over $\mathbf{D}$ with morphisms $(a_i, b_i) \colon G_i \to H$.

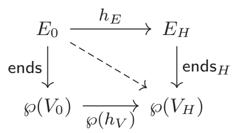


We show that G_0 is initial by constructing a morphism $(h_V, h_E) \colon G_0 \to H$ and showing that it is the unique one commuting with the inclusions.

By applying the functors $\mathbf{U}_E$ and $\mathbf{U}_V$ to the diagram above, we obtain the following diagrams in Set, where $h_V \colon V_0 \to V_H$ and $h_E \colon E_0 \to E_H$ are the unique morphism from the colimit cone.



We show that (h_V, h_E) is a hypergraph morphism. The object $\wp(V_H)$ is a cocone over $\mathbf{D} \,;\, \mathbf{U}_E$ in (at least) two ways: with morphisms $d_i \,;\, \mathsf{ends} \,;\, \wp(h_V)$ and morphisms $b_i \,;\, \mathsf{ends}_H$. By initiality of E_0, there is a unique morphism $E_0 \to \wp(V_H)$ and it must coincide with $h_E \,;\, \mathsf{ends}_H$ and $\mathsf{ends} \,;\, \wp(h_V)$.



This proves that (h_V, h_E) is a hypergraph morphism. It is, moreover, unique because any other morphism with this property would have the same components. In fact, let $(h'_V, h'_E) \colon G_0 \to H$ be another hypergraph morphism that commutes with the cocones, i.e. $(c_i, d_i) \,;\, (h'_V, h'_E) = (a_i, b_i)$. Then, its components must commute with the respective cocones in Set, by functoriality of $\mathbf{U}_E$ and $\mathbf{U}_V$: $c_i \,;\, h'_V = a_i$ and $d_i \,;\, h'_E = b_i$. By construction, V_0 and E_0 are the colimits of $\mathbf{D} \,;\, \mathbf{U}_V$ and $\mathbf{D} \,;\, \mathbf{U}_E$, so there are unique morphisms to any other cocone over the same diagrams. This means that $h'_V = h_V$ and $h'_E = h_E$, which shows the uniqueness of (h_V, h_E). □

Definition 3.22. The category $\mathsf{Cospan}(\mathsf{UHGraph})_*$ is the full subcategory of $\mathsf{Cospan}(\mathsf{UHGraph})$ on discrete hypergraphs. Objects are sets and a morphism $g \colon X \to Y$ is given by a hypergraph $G = (V, E)$ and two functions, $\partial_X \colon X \to V$ and $\partial_Y \colon Y \to V$.

Composition in $\mathsf{Cospan}(\mathsf{UHGraph})_*$ is given by identification of the common sources: if two vertices are pointed by a common source, then they are identified.

Example 3.23. The composition of two morphisms with a single edge along a common vertex gives a path of length two, obtained by identifying the vertex v of the first morphism

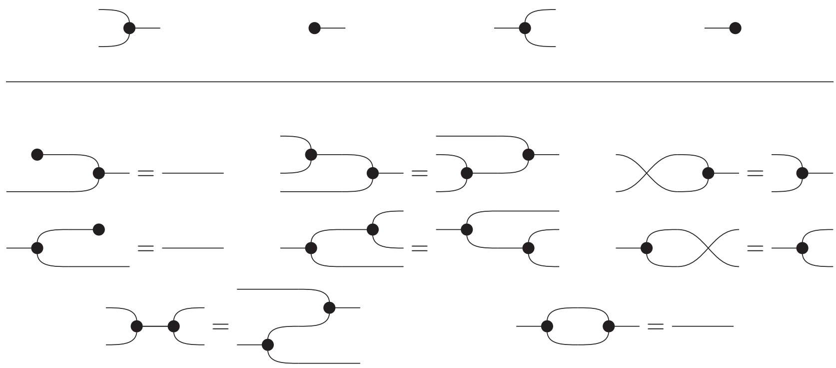
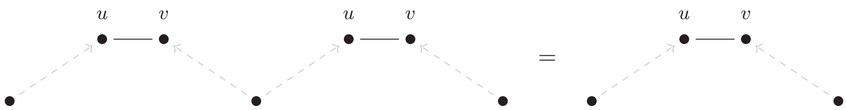


FIGURE 5. Generators and axioms of a special Frobenius monoid.

with the vertex u of the second.



3.4. **String diagrams for cospans of hypergraphs.** We introduce a syntax for the monoidal category $\mathsf{Cospan}(\mathsf{UHGraph})_*$, which we will use for proving some of the results in this section. We will show that the syntax for $\mathsf{Cospan}(\mathsf{UHGraph})_*$ is given by the syntax of $\mathsf{Cospan}(\mathsf{Set})$ together with an extra "hyperedge" generator $\mathsf{e}_n \colon n \to 0$ for every $n \in \mathbb{N}$. This result is inspired by the similar one for cospans of directed graphs [RSW05].

It is well-known that the category $\mathsf{Cospan}(\mathsf{Set})$ of finite sets and cospans of functions between them has a convenient syntax given by the walking special Frobenius monoid [Lac04].

Proposition 3.24 [Lac04]**.** *The skeleton of the monoidal category* $\mathsf{Cospan}(\mathsf{Set})$ *is isomorphic to the prop* sFrob*, whose generators and axioms are in Figure 5.*

In order to obtain cospans of hypergraphs from cospans of sets, we need to add generators that behave like hyperedges: they have n inputs and these inputs can be permuted without any effect.

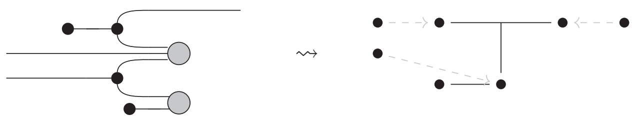
Definition 3.25. Define UHedge to be the prop generated by a "hyperedge" generator $\mathsf{e}_n \colon n \to 0$ for every $n \in \mathbb{N}$ such that permuting its inputs does not have any effect:

The syntax for cospans of graphs is defined as a coproduct of props.

Definition 3.26. Define the prop FGraph as a coproduct: $\mathsf{FGraph} := \mathsf{sFrob} + \mathsf{UHedge}$.

We will show that every morphism $g \colon n \to m$ in FGraph corresponds to a morphism in $\mathsf{Cospan}(\mathsf{UHGraph})_*$.

Example 3.27. The string diagram below corresponds to a hypergraph with two left sources, one right source and two hyperedges. The number of endpoints of each hyperedge is given by the arity of the corresponding generator in the string diagram. Two hyperedges are adjacent to the same vertex when they are connected by the Frobenius structure in the string diagram, and a hyperedge is adjacent to a source when it is connected to an input or output in the string diagram.



Proposition 3.28. *There is a symmetric monoidal functor* $\mathbf{S}\colon \mathsf{FGraph} \to \mathsf{Cospan}(\mathsf{UHGraph})_*$.

Proof. By definition, $\mathsf{FGraph} \coloneqq \mathsf{sFrob} + \mathsf{UHedge}$ is a coproduct. Therefore, it suffices to define two symmetric monoidal functors $\mathbf{S_1}\colon \mathsf{sFrob} \to \mathsf{Cospan}(\mathsf{UHGraph})_*$ and $\mathbf{S_2}\colon \mathsf{UHedge} \to \mathsf{Cospan}(\mathsf{UHGraph})_*$ for constructing the functor $\mathbf{S} \coloneqq [\mathbf{S_1}, \mathbf{S_2}]$.

The category of cospans of finite sets embeds into the category of cospans of undirected hypergraphs, and in particular $\mathsf{Cospan}(\mathsf{Set}) \hookrightarrow \mathsf{Cospan}(\mathsf{UHGraph})_*$. By Proposition 3.24, there is a functor $\mathsf{sFrob} \to \mathsf{Cospan}(\mathsf{Set})$, which gives us a functor $\mathbf{S_1}\colon \mathsf{sFrob} \to \mathsf{Cospan}(\mathsf{UHGraph})_*$.

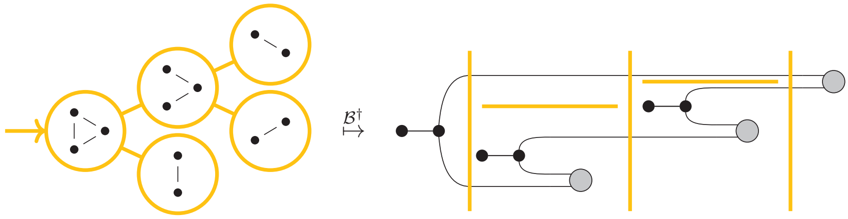
For the functor $\mathbf{S_2}$, we need to define it on the generators of UHedge and show that it preserves the equations. We define $\mathbf{S_2}(\mathsf{e}_n)$ to be the cospan of graphs $n \to (n, \{e\}) \leftarrow \emptyset$ given by $\mathbb{1}_n\colon n \to n$ and $¡_n\colon \emptyset \to n$. With this assignment, we can freely extend $\mathbf{S_2}$ to a monoidal functor $\mathsf{UHedge} \to \mathsf{Cospan}(\mathsf{UHGraph})_*$. In fact, it preserves the equations of UHedge because permuting the order of the endpoints of an undirected hyperedge has no effect by definition. □

In order to instantiate monoidal width in $\mathsf{Cospan}(\mathsf{UHGraph})_*$, we need to define an appropriate weight function.

Definition 3.29. Let $\mathcal{A}$ be all morphisms of $\mathsf{Cospan}(\mathsf{UHGraph})_*$. Define the *weight function* as follows. For an object X, $\mathsf{w}(X) \coloneqq |X|$. For a morphism $g \in \mathcal{A}$, $\mathsf{w}(g) \coloneqq |V|$, where V is the set of vertices of the apex of g, i.e. $g = X \to G \leftarrow Y$ and $G = (V, E)$.

3.5. **Tree width as monoidal width.** Here we show that monoidal width in the monoidal category $\mathsf{Cospan}(\mathsf{UHGraph})_*$, with the weight function given in Definition 3.29, is equivalent to tree width. We do this by bounding monoidal width by above with branch width +1 and by below with half of branch width (Theorem 3.34). We prove these bounds by defining maps from inductive branch decompositions to monoidal decompositions that preserve the width (Proposition 3.30), and vice versa (Proposition 3.33).

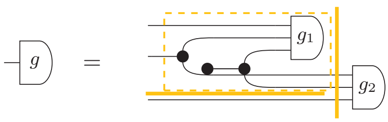
The idea behind the mapping from inductive branch decompositions to monoidal decompositions is to take a one-edge hypergraph for each leaf of the inductive branch decomposition and compose them following the structure of the decomposition tree. The 3-clique has a branch decomposition as shown on the left. The corresponding monoidal decomposition is shown on the right.



Proposition 3.30. *Let* $\Gamma = (G, X)$ *be a hypergraph with sources and* T *be an inductive branch decomposition of* Γ*. Let* $g := X \xrightarrow{\iota} G \leftarrow \emptyset$ *be the corresponding cospan. Then, there is a monoidal decomposition* $\mathcal{B}^\dagger(T) \in D_g$ *such that* $\mathsf{wd}(\mathcal{B}^\dagger(T)) \leq \max\{\mathsf{wd}(T) + 1, \gamma(G)\}$*.*

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

If $T = (T_1 \text{—} \Gamma \text{—} T_2)$, then, by definition of 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 $g_i = X_i \to G_i \leftarrow \emptyset$ be the cospan given by $\iota \colon X_i \to V_i$ and corresponding to Γ_i. Then, we can decompose g in terms of identities, the structure of $\mathsf{Cospan}(\mathsf{UHGraph})_*$, and its subgraphs g_1 and g_2:



By induction hypothesis, there are monoidal decompositions $\mathcal{B}^\dagger(T_i)$ of g_i whose width is bounded: $\mathsf{wd}(\mathcal{B}^\dagger(T_i)) \leq \max\{\mathsf{wd}(T_i) + 1, \gamma(G_i)\}$. By Lemma 2.8, there is a monoidal decomposition $\mathcal{C}(\mathcal{B}^\dagger(T_1))$ of the morphism in the above dashed box of bounded width: $\mathsf{wd}(\mathcal{C}(\mathcal{B}^\dagger(T_1))) \leq \max\{\mathsf{wd}(\mathcal{B}^\dagger(T_1)), |X_1| + 1\}$. Using this decomposition, we can define the monoidal decomposition given by the cuts in the figure above.

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

We can bound its width by applying Lemma 2.8, the induction hypothesis and the relevant definitions of width (Definition 3.11 and Definition 3.29).

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

□

The mapping $\mathcal{B}$ follows the same idea of the mapping $\mathcal{B}^\dagger$ but requires extra care: we need to keep track of which vertices are going to be identified in the final cospan. The function ϕ stores this information, thus it cannot identify two vertices that are not already in the boundary of the hypergraph. The proof of Proposition 3.33 proceeds by induction on the monoidal decomposition and constructs the corresponding branch decomposition. The inductive step relies on ϕ to identify which subgraphs of Γ correspond to the two subtrees in the monoidal decomposition, and, consequently, to define the corresponding branch decomposition.

Remark 3.31. Let $f\colon A \to C$ and $g\colon B \to C$ be two functions. The union of the images of f and g is the image of the coproduct map $[f, g]\colon A+B \to C$, i.e. $\mathsf{im}(f) \cup \mathsf{im}(g) = \mathsf{im}([f, g])$. The intersection of the images of f and g is the image of the pullback map $\langle f \wedge g\rangle\colon A \times_C B \to C$, i.e. $\mathsf{im}(f) \cap \mathsf{im}(g) = \mathsf{im}(\langle f \wedge g\rangle)$.

Remark 3.32. Let $f\colon A \to C$, $g\colon B \to C$ and $\phi\colon C \to V$ such that $\forall\ c \neq c' \in C\ \phi(c) = \phi(c') \Rightarrow c, c' \in \mathsf{im}(f)$. We have that $\mathsf{im}(\langle f;\phi \wedge g;\phi\rangle) \supseteq \mathsf{im}(\langle f \wedge g\rangle;\phi)$. Then, $\mathsf{im}(\langle f;\phi \wedge g;\phi\rangle) = \mathsf{im}(\langle f \wedge g\rangle\ ;\phi)$ because their difference is empty:

$$\begin{aligned}&\mathsf{im}(\langle f \wedge g\rangle\ ;\phi) \setminus \mathsf{im}(\langle f\ ;\phi \wedge g\ ;\phi\rangle)\\ &= \{v \in V : \exists a \in A\ \exists b \in B\ \phi(f(a)) = \phi(g(b)) \wedge f(a) \notin \mathsf{im}(g) \wedge g(b) \notin \mathsf{im}(f)\} = \emptyset\end{aligned}$$

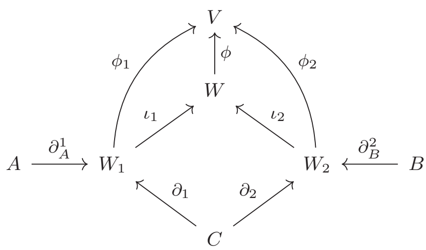
Proposition 3.33. *Let $h = A \overset{\partial_A}{\to} H \overset{\partial_B}{\leftarrow} B$ with $H = (W, F)$. Let $\phi\colon W \to V$ such that $\forall\ w \neq w' \in W\ \phi(w) = \phi(w') \Rightarrow w, w' \in \mathsf{im}(\partial_A) \cup \mathsf{im}(\partial_B)$ (glueing property). Let d be a monoidal decomposition of h. Let $\Gamma := ((\mathsf{im}(\phi), F), \mathsf{im}(\partial_A\ ;\phi) \cup \mathsf{im}(\partial_B\ ;\phi))$. Then, there is an inductive branch decomposition $\mathcal{B}(d)$ of Γ such that $\mathsf{wd}(\mathcal{B}(d)) \leq 2 \cdot \max\{\mathsf{wd}(d), |A|, |B|\}$.*

Proof. Proceed by induction on the decomposition tree d. If it is just a leaf, $d = (h)$ and H has no edges, $F = \emptyset$, then the corresponding inductive branch decomposition is empty, $\mathcal{B}(d) := ()$, and we can compute its width: $\mathsf{wd}(\mathcal{B}(d)) := 0 \leq 2 \cdot \max\{\mathsf{wd}(d), |A|, |B|\}$.

If the decomposition is just a leaf $d = (h)$ but H has exactly one edge, $F = \{e\}$, then the corresponding branch decomposition is just a leaf as well, $\mathcal{B}(d) := (\Gamma)$, and we can compute its width: $\mathsf{wd}(\mathcal{B}(d)) := |\mathsf{im}(\partial_A\ ;\phi) \cup \mathsf{im}(\partial_B\ ;\phi)| \leq |A| + |B| \leq 2 \cdot \max\{\mathsf{wd}(d), |A|, |B|\}$.

If the decomposition is just a leaf $d = (h)$ and H has more than one edge, $|F| > 1$, then we can let $\mathcal{B}(d)$ be any inductive branch decomposition of Γ. Its width is not greater than the number of vertices in Γ, thus we can bound its width $\mathsf{wd}(\mathcal{B}(d)) \leq |\mathsf{im}(\phi)| \leq 2 \cdot \max\{\mathsf{wd}(d), |A|, |B|\}$.

If $d = (d_1 \text{—}\, ;_C \text{—} d_2)$, then d_i is a monoidal decomposition of h_i with $h = h_1 ;_C h_2$. We can give the expressions of these morphisms: $h_1 = A \overset{\partial_A^1}{\to} H_1 \overset{\partial_1}{\leftarrow} C$ and $h_2 = C \overset{\partial_2}{\to} H_2 \overset{\partial_B^2}{\leftarrow} B$, with $H_i = (W_i, F_i)$, and obtain the following diagram, where $\iota_i\colon W_i \to W$ are the functions induced by the pushout and we define $\phi_i := \iota_i\ ;\phi$.



We show that ϕ_1 satisfies the glueing property in order to apply the induction hypothesis to ϕ_1 and H_1: let $w \neq w' \in W_1$ such that $\phi_1(w) = \phi_1(w')$. Then, $\iota_1(w) = \iota_1(w')$ or $\phi(\iota_1(w)) = \phi(\iota_1(w')) \wedge \iota_1(w) \neq \iota_1(w')$. Then, $w, w' \in \mathsf{im}(\partial_1)$ or $\iota_1(w), \iota_1(w') \in \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)$. Then, $w, w' \in \mathsf{im}(\partial_1)$ or $w, w' \in \mathsf{im}(\partial_A^1)$. Then, $w, w' \in \mathsf{im}(\partial_1) \cup \mathsf{im}(\partial_A^1)$. Similarly, we can show that ϕ_2 satisfies the same property. Then, we can apply the induction hypothesis to get an inductive branch decomposition $\mathcal{B}(d_1)$ of $\Gamma_1 = ((\mathsf{im}(\phi_1), F_1), \mathsf{im}(\partial_A^1 ; \phi_1) \cup \mathsf{im}(\partial_1 ; \phi_1))$ and an inductive branch decomposition $\mathcal{B}(d_2)$ of $\Gamma_2 = ((\mathsf{im}(\phi_2), F_2), \mathsf{im}(\partial_B^2 ; \phi_2) \cup \mathsf{im}(\partial_2 ; \phi_2))$ with bounded width: $\mathsf{wd}(\mathcal{B}(d_1)) \leq 2 \cdot \max\{\mathsf{wd}(d_1), |A|, |C|\}$ and $\mathsf{wd}(\mathcal{B}(d_2)) \leq 2 \cdot \max\{\mathsf{wd}(d_2), |B|, |C|\}$.

We check that we can define an inductive branch decomposition of Γ from $\mathcal{B}(d_1)$ and $\mathcal{B}(d_2)$.

- $F = F_1 \sqcup F_2$ because the pushout is along discrete hypergraphs.
- $\mathsf{im}(\phi) = \mathsf{im}(\phi_1) \cup \mathsf{im}(\phi_2)$ because $\mathsf{im}([\iota_1, \iota_2]) = W$ and $\mathsf{im}(\phi_1) \cup \mathsf{im}(\phi_2) = \mathsf{im}(\iota_1 ; \phi) \cup \mathsf{im}(\iota_2 ; \phi) = \mathsf{im}([\iota_1, \iota_2] ; \phi) = \mathsf{im}(\phi)$.
- $\mathsf{im}([\partial_A^1, \partial_1] ; \phi_1) = \mathsf{im}(\phi_1) \cap (\mathsf{im}(\phi_2) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi))$ because

$$
\begin{aligned}
&\mathsf{im}(\phi_1) \cap (\mathsf{im}(\phi_2) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)) \\
&= \quad \text{(by definition of } \phi_i) \\
&\mathsf{im}(\iota_1 ; \phi) \cap (\mathsf{im}(\iota_2 ; \phi) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)) \\
&= \quad \text{(because } \mathsf{im}(\partial_B) = \mathsf{im}(\partial_B^2 ; \iota_2) \subseteq \mathsf{im}(\iota_2)) \\
&\mathsf{im}(\iota_1 ; \phi) \cap (\mathsf{im}(\iota_2 ; \phi) \cup \mathsf{im}(\partial_A ; \phi)) \\
&= \quad \text{(by Remark 3.31)} \\
&\mathsf{im}(\iota_1 ; \phi) \cap \mathsf{im}([\iota_2, \partial_A] ; \phi) \\
&= \quad \text{(by Remark 3.31)} \\
&\mathsf{im}(\langle \iota_1 ; \phi \wedge [\iota_2, \partial_A] ; \phi \rangle) \\
&= \quad \text{(by Remark 3.32)} \\
&\mathsf{im}(\langle \iota_1 \wedge [\iota_2, \partial_A] \rangle ; \phi) \\
&= \quad \text{(because pullbacks commute with coproducts)} \\
&\mathsf{im}([\langle \iota_1 \wedge \iota_2 \rangle, \langle \iota_1 \wedge \partial_A \rangle] ; \phi) \\
&= \quad \text{(because } \partial_A = \partial_A^1 ; \iota_1) \\
&\mathsf{im}([\langle \iota_1 \wedge \iota_2 \rangle, \partial_A] ; \phi) \\
&= \quad \text{(because } \partial_1 ; \iota_1 = \partial_2 ; \iota_2 \text{ is the pushout map of } \partial_1 \text{ and } \partial_2) \\
&\mathsf{im}([\partial_1 ; \iota_1, \partial_A^1 ; \iota_1] ; \phi) \\
&= \quad \text{(by property of the coproduct)} \\
&\mathsf{im}([\partial_1, \partial_A^1] ; \phi_1)
\end{aligned}
$$

- $\mathsf{im}([\partial_2, \partial_B^2] ; \phi_2) = \mathsf{im}(\phi_2) \cap (\mathsf{im}(\phi_1) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi))$ similarly to the former point.

Then, $\mathcal{B}(d) \coloneqq (\mathcal{B}(d_1) — \Gamma — \mathcal{B}(d_2))$ is an inductive branch decomposition of Γ and

$$
\begin{aligned}
&\mathsf{wd}(\mathcal{B}(d)) \\
&\coloneqq \max\{\mathsf{wd}(\mathcal{B}(d_1)), |\mathsf{im}([\partial_A, \partial_B])|, \mathsf{wd}(\mathcal{B}(d_2))\} \\
&\leq \max\{2 \cdot \mathsf{wd}(d_1), 2 \cdot |A|, 2 \cdot |C|, |A| + |B|,
\end{aligned}
$$

$$2 \cdot \mathsf{wd}(d_2), 2 \cdot |B|\}$$
$$\leq 2 \cdot \max\{\mathsf{wd}(d_1), |A|, |C|, \mathsf{wd}(d_2), |B|\}$$
$$=: 2 \cdot \max\{\mathsf{wd}(d), |A|, |B|\}$$

If $d = (d_1 \text{—} \otimes \text{—} d_2)$, then d_i is a monoidal decomposition of h_i with $h = h_1 \otimes h_2$. Let $h_i = X_i \xrightarrow{\partial^i_X} H_i \xleftarrow{\partial^i_Y} Y_i$ with $H_i = F_i \overset{s,t}{\rightrightarrows} W_i$. Let $\iota_i \colon W_i \to W$ be the inclusions induced by the monoidal product. Define $\phi_i := \iota_i \, ; \phi$. We show that ϕ_1 satisfies the glueing property: Let $w \neq w' \in W_1$ such that $\phi_1(w) = \phi_1(w')$. Then, $\iota_1(w) = \iota_1(w')$ or $\phi(\iota_1(w)) = \phi(\iota_1(w')) \wedge \iota_1(w) \neq \iota_1(w')$. Then, $\iota_1(w), \iota_1(w') \in \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)$ because ι_i are injective. Then, $w, w' \in \mathsf{im}(\partial^1_A) \cup \mathsf{im}(\partial^1_B)$. Similarly, we can show that ϕ_2 satisfies the same property. Then, we can apply the induction hypothesis to get $\mathcal{B}(d_i)$ inductive branch decomposition of $\Gamma_i = ((\mathsf{im}(\phi_i), F_i), \mathsf{im}([\partial^i_A, \partial^i_B] ; \phi_i))$ such that $\mathsf{wd}(\mathcal{B}(d_i)) \leq 2 \cdot \max\{\mathsf{wd}(d_i), |A_i|, |B_i|\}$.

We check that we can define an inductive branch decomposition of Γ from $\mathcal{B}(d_1)$ and $\mathcal{B}(d_2)$.

- $F = F_1 \sqcup F_2$ because the monoidal product is given by the coproduct in Set.
- $\mathsf{im}(\phi) = \mathsf{im}(\phi_1) \cup \mathsf{im}(\phi_2)$ because $\mathsf{im}([\iota_1, \iota_2]) = W$ and $\mathsf{im}(\phi_1) \cup \mathsf{im}(\phi_2) = \mathsf{im}(\iota_1 ; \phi) \cup \mathsf{im}(\iota_2 ; \phi) = \mathsf{im}([\iota_1, \iota_2] ; \phi) = \mathsf{im}(\phi)$.
- $\mathsf{im}([\partial^1_A, \partial^1_B] ; \phi_1) = \mathsf{im}(\phi_1) \cap (\mathsf{im}(\phi_2) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi))$ because

$$\begin{aligned}
&\mathsf{im}(\phi_1) \cap (\mathsf{im}(\phi_2) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)) \\
&= \quad \text{(by definition of } \phi_i) \\
&\mathsf{im}(\iota_1 ; \phi) \cap (\mathsf{im}(\iota_2 ; \phi) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi)) \\
&= \quad \text{(by Remark 3.31 and property of the coproduct)} \\
&\mathsf{im}(\iota_1 ; \phi) \cap \mathsf{im}([\iota_2, [\partial_A, \partial_B]] ; \phi) \\
&= \quad \text{(by Remark 3.31)} \\
&\mathsf{im}(\langle \iota_1 ; \phi \wedge [\iota_2, [\partial_A, \partial_B]] ; \phi \rangle) \\
&= \quad \text{(by Remark 3.32)} \\
&\mathsf{im}(\langle \iota_1 \wedge [\iota_2, [\partial_A, \partial_B]] \rangle ; \phi) \\
&= \quad \text{(because pullbacks commute with coproducts)} \\
&\mathsf{im}([\langle \iota_1 \wedge \iota_2 \rangle, \langle \iota_1 \wedge [\partial_A, \partial_B] \rangle] ; \phi) \\
&= \quad \text{(because } \langle \iota_1 \wedge \iota_2 \rangle = \text{¡}) \\
&\mathsf{im}(\langle \iota_1 \wedge [\partial_A, \partial_B] \rangle ; \phi) \\
&= \quad \text{(because } \partial_A = \partial^1_A + \partial^2_A \text{ and } \partial_B = \partial^1_B + \partial^2_B) \\
&\mathsf{im}([\partial^1_A ; \iota_1, \partial^1_B ; \iota_1] ; \phi) \\
&= \quad \text{(by property of the coproduct)} \\
&\mathsf{im}([\partial^1_A, \partial^1_B] ; \phi_1)
\end{aligned}$$

- $\mathsf{im}([\partial^2_A, \partial^2_B] ; \phi_2) = \mathsf{im}(\phi_2) \cap (\mathsf{im}(\phi_1) \cup \mathsf{im}(\partial_A ; \phi) \cup \mathsf{im}(\partial_B ; \phi))$ similarly to the former point.

Then, $\mathcal{B}(d) := (\mathcal{B}(d_1) \text{—} \Gamma \text{—} \mathcal{B}(d_2))$ is an inductive branch decomposition of Γ and

$$\mathsf{wd}(\mathcal{B}(d))$$
$$:= \max\{\mathsf{wd}(\mathcal{B}(d_1)), |\mathsf{im}([\partial_A, \partial_B])|, \mathsf{wd}(\mathcal{B}(d_2))\}$$

$$\begin{aligned}&\leq \max\{2\cdot \mathsf{wd}(d_1), 2\cdot|A_1|, 2\cdot|B_1|, |A|+|B|,\\&\quad 2\cdot\mathsf{wd}(d_2), 2\cdot|A_2|, 2\cdot|B_2|\}\\&\leq 2\cdot\max\{\mathsf{wd}(d_1), |A|, \mathsf{wd}(d_2), |B|\}\\&=: 2\cdot\max\{\mathsf{wd}(d), |A|, |B|\}\end{aligned}$$

where we applied the induction hypothesis and Definition 3.29. □

Combining Theorem 3.9, Proposition 3.19, Proposition 3.30, and Proposition 3.33, we obtain the following.

Theorem 3.34. *Branch width is equivalent to monoidal width in* $\mathsf{Cospan}(\mathsf{UHGraph})_*$. *More precisely, let G be a hypergraph and $g = \emptyset \to G \leftarrow \emptyset$ be the corresponding morphism of* $\mathsf{Cospan}(\mathsf{UHGraph})_*$. *Then,* $\frac{1}{2}\cdot\mathsf{bwd}(G) \leq \mathsf{mwd}(g) \leq \mathsf{bwd}(G)+1$.

With Theorem 3.9, we obtain:

Corollary 3.35. *Tree width is equivalent to monoidal width in* $\mathsf{Cospan}(\mathsf{UHGraph})_*$.

4. Monoidal width in matrices

We have just seen that instantiating monoidal width in a monoidal category of graphs yields a measure that is equivalent to tree width. Now, we turn our attention to rank width, which is more linear algebraic in flavour as it relies on treating the connectivity of graphs by means of adjacency matrices. Thus, the monoidal category of matrices is a natural example to study first. We relate monoidal width in the category of matrices over the natural numbers, which we introduce in Section 4.1, to their rank (Section 4.2).

The rank of a matrix is the maximum number of its linearly independent rows (or, equivalently, columns). Conveniently, it can be characterised in terms of minimal factorisations.

Lemma 4.1 [PO99]**.** *Let* $A \in \mathsf{Mat}_{\mathbb{N}}(m,n)$ *be an m by n matrix with entries in the natural numbers. Then* $\mathsf{rk}(A) = \min\{k \in \mathbb{N} : \exists B \in \mathsf{Mat}_{\mathbb{N}}(k,n)\ \exists C \in \mathsf{Mat}_{\mathbb{N}}(m,k)\ A = C\cdot B\}$.

4.1. The prop of matrices.

The monoidal category $\mathsf{Mat}_{\mathbb{N}}$ of matrices with entries in the natural numbers is a prop whose morphisms from n to m are m by n matrices.

Definition 4.2. $\mathsf{Mat}_{\mathbb{N}}$ is the prop whose morphisms $n \to m$ are m by n matrices with entries in the natural numbers. Composition is the usual product of matrices and the monoidal product is the biproduct $A \otimes B := \left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & B\end{smallmatrix}\right)$.

Let us examine matrix decompositions enabled by this algebra. A matrix A can be written as a monoidal product $A = A_1 \otimes A_2$ iff the matrix has blocks A_1 and A_2, i.e. $A = \begin{pmatrix} A_1 & \mathbb{0} \\ \mathbb{0} & A_2\end{pmatrix}$. On the other hand, a composition is related to the rank: the statement of Lemma 4.1 can be read in the category $\mathsf{Mat}_{\mathbb{N}}$ as $\mathsf{rk}(A) = \min\{k \in \mathbb{N} : A = B\,;_k\, C\}$.

Theorem 4.3 [Zan15]**.** *Let* Bialg *be the prop whose generators and axioms are given in Figure 6. There is an isomorphism of categories* $\mathbf{Mat}\colon \mathsf{Bialg} \to \mathsf{Mat}_{\mathbb{N}}$.

Every morphism $f\colon n \to m$ in Bialg corresponds to a matrix $A = \mathbf{Mat}(f) \in \mathsf{Mat}_{\mathbb{N}}(m,n)$: we can read the (i,j)-entry of A off the diagram of f by counting the number of paths from the jth input to the ith output.

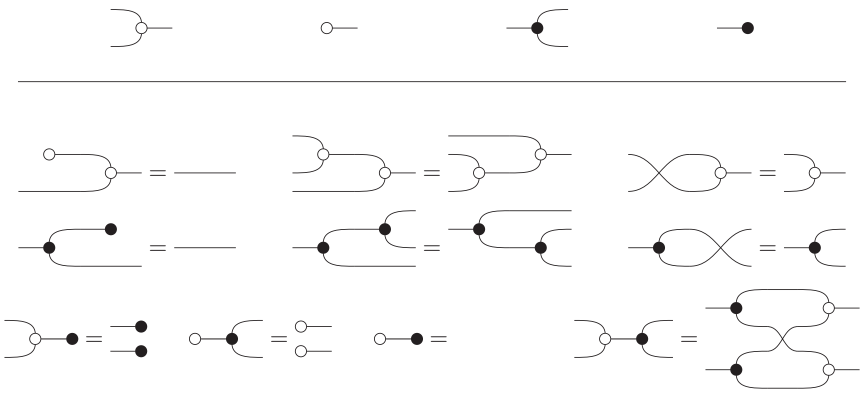
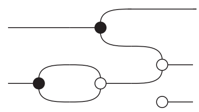


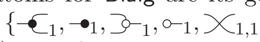
FIGURE 6. Generators and axioms of a bialgebra.

Example 4.4. The matrix $\begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 0 \end{pmatrix} \in \mathsf{Mat}_{\mathbb{N}}(3, 2)$ corresponds to



For matrices $A \in \mathsf{Mat}_{\mathbb{N}}(m, n)$, $B \in \mathsf{Mat}_{\mathbb{N}}(m, p)$ and $C \in \mathsf{Mat}_{\mathbb{N}}(l, n)$, we indicate with $(A \mid B) \in \mathsf{Mat}_{\mathbb{N}}(m, n + p)$ and with $\binom{A}{C} \in \mathsf{Mat}_{\mathbb{N}}(m + l, n)$ the matrices obtained by concatenating A with B horizontally or with C vertically.

In order to instantiate monoidal width in Bialg, we need to define an appropriate weight function: the natural choice for a prop is to assign weight n to compositions along the object n.

Definition 4.5. The atoms for Bialg are its generators (Figure 6) with the symmetry and identity on 1: $\mathcal{A} = \{$  $, \mathbb{1}_1\}$. The weight function $\mathsf{w} \colon \mathcal{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{Bialg}) \to \mathbb{N}$ has $\mathsf{w}(n) := n$, for any $n \in \mathbb{N}$, and $\mathsf{w}(g) := \max\{m, n\}$, for $g \colon n \to m \in \mathcal{A}$.

4.2. **Monoidal width of matrices.** We show that the monoidal width of a morphism in the category of matrices Bialg, with the weight function in Definition 4.5, is, up to 1, the maximum rank of its blocks. The overall strategy to prove this result is to first relate monoidal width directly with the rank (Proposition 4.8) and then to improve this bound by prioritising $\otimes$-nodes in a decomposition (Proposition 4.10). Combining these two results leads to Theorem 4.13. The shape of an optimal decomposition is given in Figure 7: a matrix $A = \begin{pmatrix} A_1 & \mathbb{0} & \cdots & \mathbb{0} \\ \mathbb{0} & A_2 & \cdots & \mathbb{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{0} & \mathbb{0} & \cdots & A_k \end{pmatrix}$ can be decomposed as $A = (M_1 \,;\, N_1) \otimes (M_2 \,;\, N_2) \otimes \cdots \otimes (M_k \,;\, N_k)$, where $A_j = M_j \,;\, N_j$ is a rank factorisation as in Lemma 4.1.

The characterisation of the rank of a matrix in Lemma 4.1 hints at some relationship between the monoidal width of a matrix and its rank. In fact, we have Proposition 4.8,

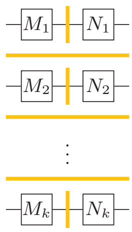


Figure 7. Generic shape of an optimal decomposition in Bialg.

which bounds the monoidal width of a matrix with its rank. In order to prove this result, we first need to bound the monoidal width of a matrix with its domain and codomain, which is done in Proposition 4.6.

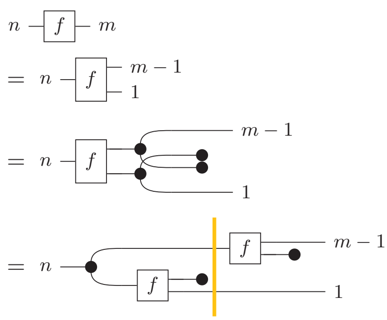
Proposition 4.6. *Let* P *be a cartesian and cocartesian prop. Suppose that* $\mathbb{1}_1, -\!\!\bullet\!\!\!<_1, >\!\!\circ\!\!-_1, -\!\!\bullet_1, \circ\!\!-_1 \in \mathcal{A}$ *and* $\mathsf{w}(\mathbb{1}_1) \leq 1$, $\mathsf{w}(-\!\!\bullet\!\!\!<_1) \leq 2$, $\mathsf{w}(>\!\!\circ\!\!-_1) \leq 2$, $\mathsf{w}(-\!\!\bullet_1) \leq 1$ *and* $\mathsf{w}(\circ\!\!-_1) \leq 1$. *Suppose that, for every* $g \colon 1 \to 1$, $\mathsf{mwd}(g) \leq 2$. *Let* $f \colon n \to m$ *be a morphism in* P. *Then* $\mathsf{mwd}(f) \leq \min\{m, n\} + 1$.

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, and we can compute its width, $\mathsf{mwd}(f) = \mathsf{mwd}(\bigotimes_m \circ\!\!-_1) \leq \mathsf{w}(\circ\!\!-_1) \leq 1 \leq 0 + 1$.
- If $m = 0$, then $f = -\!\!\bullet_n$ because 0 is terminal by hypothesis, and we can compute its width, $\mathsf{mwd}(f) = \mathsf{mwd}(\bigotimes_m -\!\!\bullet_1) \leq \mathsf{w}(-\!\!\bullet_1) \leq 1 \leq 0 + 1$.
- If $m = n = 1$, then $\mathsf{mwd}(f) \leq 2 \leq 1 + 1$ by hypothesis.

For the induction steps, suppose that the statement is true for any $f' \colon n' \to m'$ 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 $-\!\!\bullet\!\!\!<_{n+1}$ is uniform and morphisms are copiable because P is cartesian by hypothesis.



This corresponds to $f = -\!\!\bullet\!\!\!<_n ; (\mathbb{1}_n \otimes h_1) ;_{n+1} (h_2 \otimes \mathbb{1}_1)$, where $h_1 := f ; (-\!\!\bullet_{m-1} \otimes \mathbb{1}_1) \colon n \to 1$ and $h_2 := f ; (\mathbb{1}_{m-1} \otimes -\!\!\bullet_1) \colon n \to m - 1$.

Then, $\mathsf{mwd}(f) \leq \max\{\mathsf{mwd}(-\!\!\bullet\!\!\!<_n ; (\mathbb{1}_n \otimes h_1)), n + 1, \mathsf{mwd}(h_2 \otimes \mathbb{1}_1)\}$. 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 $h_1 \colon n \to 1$ and $1, n < k$.

For the second morphism, we apply the induction hypothesis because $h_2 \colon n \to m-1$ and $n, m-1 < k$.

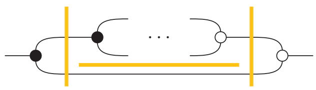
$$
\begin{aligned}
&\mathsf{mwd}(\text{-}\!\bullet\!\!\subset_n ; (\mathbb{1}_n \otimes h_1)) \\
&\leq \qquad \text{(by Lemma 2.8)} \\
&\max\{\mathsf{mwd}(h_1), n+1\} \\
&\leq \qquad \text{(by induction hypothesis)} \\
&\max\{\min\{n,1\}+1, n+1\} \\
&= \qquad \text{(because } 0 < n\text{)} \\
&n+1
\end{aligned}
\qquad\qquad
\begin{aligned}
&\mathsf{mwd}(h_2 \otimes \mathbb{1}_1) \\
&= \qquad \text{(by Definition 2.3)} \\
&\mathsf{mwd}(h_2) \\
&\leq \qquad \text{(by induction hypothesis)} \\
&\min\{n, m-1\}+1 \\
&= \qquad \text{(because } n \leq m-1\text{)} \\
&n+1
\end{aligned}
$$

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

(2) If $0 < m < n = k$, we can apply Item 1 to P^{op} with the same assumptions on the set of atoms because P^{op} is also cartesian and cocartesian. We obtain that $\mathsf{mwd}(f) \leq m+1 = \min\{m,n\}+1$ because $m < n$.

(3) If $0 < m = n = k$, f can be decomposed as in Item 1 and, instead of applying the induction hypothesis to bound $\mathsf{mwd}(h_1)$ and $\mathsf{mwd}(h_2)$, one applies Item 2. Then, $\mathsf{mwd}(f) \leq m+1 = \min\{m,n\}+1$ because $m = n$. □

We can apply the former result to Bialg and obtain Proposition 4.8 because the width of 1×1 matrices, which are numbers, is at most 2. This follows from the reasoning in Example 2.5 as we can write every natural number $k \colon 1 \to 1$ as the following composition:



Lemma 4.7. *Let $k \colon 1 \to 1$ in Bialg. Then, $\mathsf{mwd}(k) \leq 2$.*

Proposition 4.8. *Let $f \colon n \to m$ in Bialg. Then, $\mathsf{mwd} f \leq \mathsf{rk}(\mathbf{Mat} f) + 1$. Moreover, if f is not $\otimes$-decomposable, i.e. there are no f_1, f_2 both distinct from f s.t. $f = f_1 \otimes f_2$, then $\mathsf{rk}(\mathbf{Mat} f) \leq \mathsf{mwd} f$.*

Proof. We prove the second inequality. Let d be a monoidal decomposition of f. By hypothesis, f is non $\otimes$-decomposable. Then, there are two options.

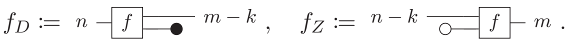
(1) If the decomposition is just a leaf, $d = (f)$, then f must be an atom. We can check the inequality for all the atoms: $\mathsf{w}(\times) = 2 \geq \mathsf{rk}(\mathbf{Mat} f) = 2$, $\mathsf{w}(\text{-}\!\bullet\!\!\subset_1) = \mathsf{w}(\supset\!\!\circ\!\text{-}_1) = 2 \geq \mathsf{rk}(\mathbf{Mat} f) = 1$ or $\mathsf{w}(\text{-}\!\bullet_1) = \mathsf{w}(\circ\!\text{-}_1) = 1 \geq \mathsf{rk}(\mathbf{Mat} f) = 0$. Then, $\mathsf{wd}(d) = \mathsf{w}(f) \geq \mathsf{rk}(\mathbf{Mat} f)$.

(2) If $d = (d_1 \text{—} ;_k \text{—} d_2)$, then there are $g \colon n \to k$ and $h \colon k \to m$ such that $f = g ; h$. By Lemma 4.1, $k \geq \mathsf{rk}(\mathbf{Mat} f)$. Then, $\mathsf{wd}(d) \geq k \geq \mathsf{rk}(\mathbf{Mat} f)$.

We prove the first inequality. By Lemma 4.1, there are $g \colon n \to r$ and $h \colon r \to m$ such that $f = g ; h$ with $r = \mathsf{rk}(\mathbf{Mat} f)$. Then, $r \leq m, n$ by definition of rank. By Lemma 4.7, we can apply Proposition 4.6 to obtain that $\mathsf{mwd}(g) \leq \min\{n, r\} + 1 = r+1$ and $\mathsf{mwd}(h) \leq \min\{m, r\} + 1 = r+1$. Then, $\mathsf{mwd}(f) \leq \max\{\mathsf{mwd}(g), r, \mathsf{mwd}(h)\} \leq r+1$. □

The bounds given by Proposition 4.8 can be improved when we have a $\otimes$-decomposition of a matrix, i.e. we can write $f = f_1 \otimes \ldots \otimes f_k$, to obtain Proposition 4.10. The latter relies on Lemma 4.9, which shows that discarding inputs or outputs cannot increase the monoidal width of a morphism in Bialg.

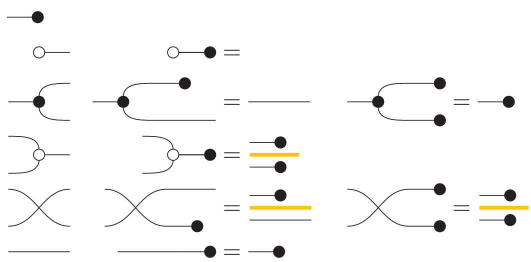
Lemma 4.9. *Let $f\colon n \to m$ in* Bialg *and $d \in D_f$. Let $f_D \coloneqq f \,;\, (\mathbb{1}_{m-k} \otimes -\!\bullet_k)$ and $f_Z \coloneqq (\mathbb{1}_{n-k'} \otimes \circ\!-_{k'}) \,;\, f$, with $k \leq m$ and $k' \leq n$.*



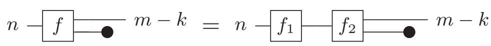
Then there are $\mathcal{D}(d) \in D_{f_D}$ and $\mathcal{Z}(d) \in D_{f_Z}$ such that $\mathsf{wd}(\mathcal{D}(d)) \leq \mathsf{wd}(d)$ and $\mathsf{wd}(\mathcal{Z}(d)) \leq \mathsf{wd}(d)$.

Proof. We show the inequality for f_D by induction on the decomposition d. The inequality for f_Z follows from the fact that Bialg coincides with its opposite category. If the decomposition has only one node, $d = (f)$, then f is an atom and we can check these cases by hand in the table below. The first column shows the possibilities for f, while the second and third columns show the decompositions of f_D for $k = 1$ and $k = 2$.

f $\quad k = 1 \quad k = 2$

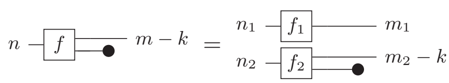


If the decomposition starts with a composition node, $d = (d_1 - ; - d_2)$, then $f = f_1 \,;\, f_2$, with d_i monoidal decomposition of f_i.

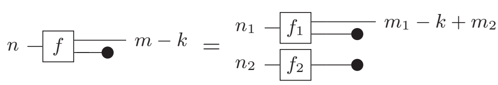


By induction hypothesis, there is a monoidal decomposition $\mathcal{D}(d_2)$ of $f_2 \,;\, (\mathbb{1}_{m-k} \otimes -\!\bullet_k)$ such that $\mathsf{wd}(\mathcal{D}(d_2)) \leq \mathsf{wd}(d_2)$. We use this decomposition to define a decomposition $\mathcal{D}(d) \coloneqq (d_1 - ; - \mathcal{D}(d_2))$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

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 $f_i\colon n_i \to m_i$. There are two possibilities: either $k \leq m_2$ or $k > m_2$. If $k \leq m_2$, then $f \,;\, (\mathbb{1}_{m-k} \otimes -\!\bullet_k) = f_1 \otimes (f_2 \,;\, (\mathbb{1}_{m_2-k} \otimes -\!\bullet_k))$.



By induction hypothesis, there is a monoidal decomposition $\mathcal{D}(d_2)$ of $f_2 \,;\, (\mathbb{1}_{m-k} \otimes -\!\bullet_k)$ such that $\mathsf{wd}(\mathcal{D}(d_2)) \leq \mathsf{wd}(d_2)$. Then, we can use this decomposition to define a decomposition $\mathcal{D}(d) \coloneqq (d_1 - \otimes - \mathcal{D}(d_2))$ of f_D. If $k > m_2$, then $f \,;\, (\mathbb{1}_{m-k} \otimes -\!\bullet_k) = (f_1 \,;\, (\mathbb{1}_{m_1-k+m_2} \otimes -\!\bullet_{k-m_2})) \otimes (f_2 \,;\, -\!\bullet_{m_2})$.

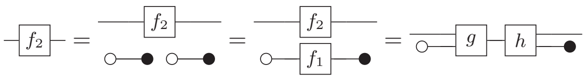


By induction hypothesis, there are monoidal decompositions $\mathcal{D}(d_i)$ of $f_1 ; (\mathbb{1}_{m_1-k+m_2} \otimes -\!\bullet_{k-m_2})$ and $f_2 \,; -\!\bullet_{m_2}$ such that $\mathsf{wd}(\mathcal{D}(d_i)) \leq \mathsf{wd}(d_i)$. Then, we can use these decompositions to define a monoidal decomposition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

Proposition 4.10. *Let $f : n \to m$ in* Bialg *and $d' = (d'_1 \text{—} ;_k \text{—} d'_2) \in D_f$. Suppose there are f_1 and f_2 such that $f = f_1 \otimes f_2$. Then, there is $d = (d_1 \text{—} \otimes \text{—} d_2) \in D_f$ such that $\mathsf{wd}(d) \leq \mathsf{wd}(d')$.*

Proof. By hypothesis, d' is a monoidal decomposition of f. Then, there are g and h such that $f_1 \otimes f_2 = f = g \,;h$. By Proposition 4.8, there are monoidal decompositions d_i of f_i with $\mathsf{wd}(d_i) \leq r_i + 1$, where $r_i := \mathsf{rk}(\mathbf{Mat} f_i)$. By properties of the rank, $r_1 + r_2 = \mathsf{rk}(\mathbf{Mat} f)$ and, by Lemma 4.1, $\mathsf{rk}(\mathbf{Mat} f) \leq k$.

There are two cases: either both ranks are non-zero, or at least one is zero. If $r_i > 0$, then $r_1 + r_2 \geq \max\{r_1, r_2\} + 1$. If there is $r_i = 0$, then $f_i = -\!\bullet \,;_0 \circ\!-$ and we may assume that $f_1 = -\!\bullet \,;_0 \circ\!-$. Then, we can express f_2 in terms of g and h.



By Lemma 4.9, $\mathsf{mwd}((\mathbb{1} \otimes \circ\!-) \,; g) \leq \mathsf{mwd}(g)$ and $\mathsf{mwd}(h \,; (\mathbb{1} \otimes -\!\bullet)) \leq \mathsf{mwd}(h)$. We compute the widths of the decompositions in these two cases.

Case $r_i > 0$	Case $r_1 = 0$
$\mathsf{wd}(d')$	$\mathsf{wd}(d')$
$= \max\{\mathsf{wd}(d'_1), k, \mathsf{wd}(d'_2)\}$	$= \max\{\mathsf{wd}(d'_1), k, \mathsf{wd}(d'_2)\}$
$\geq k$	$\geq \max\{\mathsf{mwd}(g), k, \mathsf{mwd}(h)\}$
$\geq \mathsf{rk}(\mathbf{Mat} f)$	$\geq \max\{\mathsf{mwd}((\mathbb{1} \otimes \circ\!-) \,; g), k, \mathsf{mwd}(h \,; (\mathbb{1} \otimes -\!\bullet))\}$
$= r_1 + r_2$	$\geq \mathsf{mwd}(f_2)$
$\geq \max\{r_1, r_2\} + 1$	$= \mathsf{wd}(d_2)$
$\geq \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2)\}$	$= \mathsf{wd}(d)$
$= \mathsf{wd}(d)$	

□

We summarise Proposition 4.10 and Proposition 4.8 in Corollary 4.11.

Corollary 4.11. *Let $f = f_1 \otimes \ldots \otimes f_k$ in* Bialg. *Then,* $\mathsf{mwd}(f) \leq \max_{i=1,\ldots,k} \mathsf{rk}(\mathbf{Mat}(f_i)) + 1$. *Moreover, if f_i are not $\otimes$-decomposable, then* $\max_{i=1,\ldots,k} \mathsf{rk}(\mathbf{Mat}(f_i)) \leq \mathsf{mwd} f$.

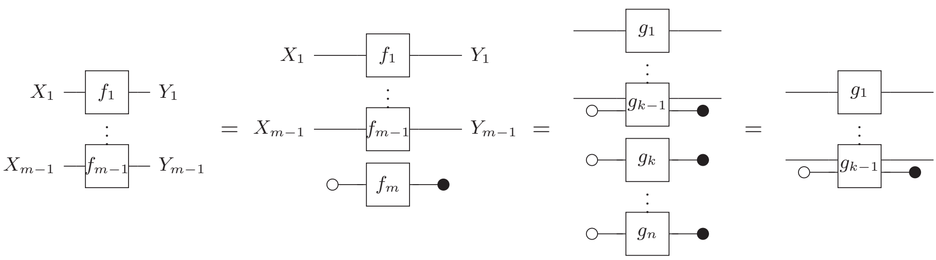
Proof. By Proposition 4.10 there is a decomposition of f of the form $d = (d_1 \text{—}\otimes\text{—} \cdots (d_{k-1} \text{—}\otimes \text{—} d_k))$, where we can choose d_i to be a minimal decomposition of f_i. Then, $\mathsf{mwd}(f) \leq \mathsf{wd}(d) = \max_{i=1,\ldots,k} \mathsf{wd}(d_i)$. By Proposition 4.8, $\mathsf{wd}(d_i) \leq r_i + 1$. Then, $\mathsf{mwd}(f) \leq \max\{r_1, \ldots, r_k\} + 1$. Moreover, if f_i are not $\otimes$-decomposable, Proposition 4.8 gives also a lower bound on their monoidal width: $\mathsf{rk}(\mathbf{Mat}(f_i)) \leq \mathsf{mwd} f_i$; and we obtain that $\max_{i=1,\ldots,k} \mathsf{rk}(\mathbf{Mat}(f_i)) \leq \mathsf{mwd} f$. □

The results so far show a way to construct efficient decompositions given a $\otimes$-decomposition of the matrix. However, we do not know whether $\otimes$-decompositions are unique. Proposition 4.12 shows that every morphism in Bialg has a unique $\otimes$-decomposition.

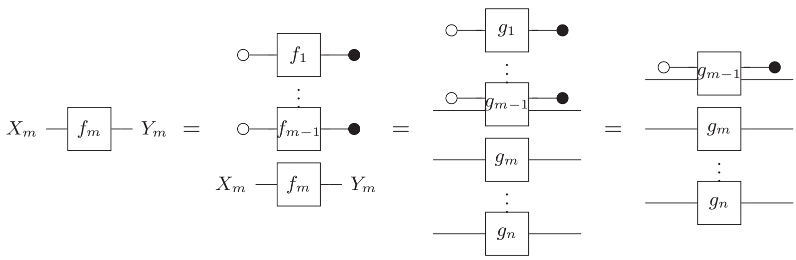
Proposition 4.12. *Let* C *be a monoidal category whose monoidal unit* 0 *is both initial and terminal, and whose objects are a unique factorisation monoid. Let f be a morphism in* C. *Then f has a unique $\otimes$-decomposition.*

Proof. Suppose $f = f_1 \otimes \cdots \otimes f_m = g_1 \otimes \cdots \otimes g_n$ with $f_i \colon X_i \to Y_i$ and $g_j \colon Z_j \to W_j$ non $\otimes$-decomposables. Suppose $m \leq n$ and proceed by induction on m. If $m = 0$, then $f = \mathbb{1}_0$ and $g_i = \mathbb{1}_0$ for every $i = 1, \ldots, n$ because 0 is initial and terminal.

Suppose that $\bar{f} \coloneqq f_1 \otimes \ldots \otimes f_{m-1}$ has a unique $\otimes$-decomposition. Let $A_1 \otimes \ldots \otimes A_\alpha$ and $B_1 \otimes \ldots \otimes B_\beta$ be the unique $\otimes$-decompositions of $X_1 \otimes \ldots \otimes X_m = Z_1 \otimes \ldots \otimes Z_n$ and $Y_1 \otimes \ldots \otimes Y_m = W_1 \otimes \ldots \otimes W_n$, respectively. Then, there are $x \leq \alpha$ and $y \leq \beta$ such that $A_1 \otimes \ldots \otimes A_x = X_1 \otimes \ldots \otimes X_{m-1}$ and $B_1 \otimes \ldots \otimes B_y = Y_1 \otimes \ldots \otimes Y_{m-1}$. Then, we can rewrite $\bar{f}$ in terms of g_is:



By induction hypothesis, $\bar{f}$ has a unique $\otimes$-decomposition, thus it must be that $k = m - 1$, for every $i < m - 1$ $f_i = g_i$ and $f_{m-1} = (\mathbb{1} \otimes \circ\!-) ; g_k ; (\mathbb{1} \otimes -\!\bullet)$. Then, we can express f_m in terms of $g_m, \ldots, g_n$:



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

Our main result in this section follows from Corollary 4.11 and Proposition 4.12, which can be applied to Bialg because 0 is both terminal and initial, and the objects, being a free monoid, are a unique factorisation monoid.

Theorem 4.13. *Let $f = f_1 \otimes \ldots \otimes f_k$ be a morphism in Bialg and its unique $\otimes$-decomposition given by Proposition 4.12, with $r_i = \mathsf{rk}(\mathbf{Mat}(f_i))$. Then* $\max\{r_1, \ldots, r_k\} \leq \mathsf{mwd}(f) \leq \max\{r_1, \ldots, r_k\} + 1$.

Note that the identity matrix has monoidal width 1 and twice the identity matrix has monoidal width 2, attaining both the upper and lower bounds for the monoidal width of a matrix.

5. A monoidal algebra for rank width

After having studied monoidal width in the monoidal category of matrices, we are ready to introduce the second monoidal category of "open graphs", which relies on matrices to encode the connectivity of graphs. In this setting, we capture rank width: we show that instantiating monoidal width in this monoidal category of graphs is equivalent to rank width.

After recalling rank width in Section 5.1, we define the intermediate notion of inductive rank decomposition in Section 5.2, and show its equivalence to that of rank decomposition. As for branch decompositions, adding this intermediate step allows a clearer presentation of the correspondence between rank decompositions and monoidal decompositions. Section 5.3 recalls the categorical algebra of graphs with boundaries [CS15, DLHS21]. Finally, Section 5.4 contains the main result of the present section, which relates inductive rank decompositions, and thus rank decompositions, with monoidal decompositions.

Rank decompositions were originally defined for undirected graphs [OS06]. This motivates us to consider graphs rather than hypergraphs as in Section 3. As mentioned in Definition 3.3, a finite undirected graph is a finite undirected hypergraph with hyperedge size 2. More explicitly,

Definition 5.1. A *graph* $G = (V, E)$ is given by a finite set of vertices V, a finite set of edges E and an adjacency function $\mathsf{ends}\colon E \to \wp_{\leq 2}(V)$, where $\wp_{\leq 2}(V)$ indicates the set of subsets of V with at most two elements. The same information recorded in the function ends can be encoded in an equivalence class of matrices, an *adjacency matrix* $[G]$: the sum of the entries (i, j) and (j, i) of this matrix records the number of edges between vertex i and vertex j; two adjacency matrices are equivalent when they encode the same graph, i.e. $[G] = [H]$ iff $G + G^\top = H + H^\top$.

5.1. Background: rank width.

Intuitively, rank width measures the amount of information needed to construct a graph by adding edges to a discrete graph. Constructing a clique requires little information: we add an edge between any two vertices. This is reflected in the fact that cliques have rank width 1.

Rank width relies on rank decompositions. In analogy with branch decompositions, a rank decomposition records in a tree a way of iteratively partitioning the vertices of a graph.

Definition 5.2 [OS06]**.** A *rank decomposition* (Y, r) of a graph G is given by a subcubic tree Y together with a bijection $r\colon \mathsf{leaves}(Y) \to \mathsf{vertices}(G)$.

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 2-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 beween 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.

Definition 5.3 (Order of an edge)**.** Let (Y, r) be a rank decomposition of a graph G. Let b be an edge of Y. The order of b is the rank of the matrix associated to it: $\mathsf{ord}(b) := \mathsf{rk}(X_b)$.

Note that the order of the two sets in the partition does not matter as the rank is invariant to transposition. The width of a rank decomposition is the maximum order of the edges of the tree and the rank width of a graph is the width of its cheapest decomposition.

Definition 5.4 (Rank width)**.** Given a rank decomposition (Y, r) of a graph G, define its width as $\mathsf{wd}(Y, r) \coloneqq \max_{b \in \mathsf{edges}(Y)} \mathsf{ord}(b)$. The *rank width* of G is given by the min-max formula:

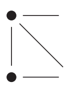
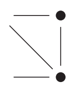
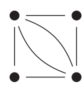
$$\mathsf{rwd}(G) \coloneqq \min_{(Y,r)} \mathsf{wd}(Y, r).$$

5.2. **Graphs with dangling edges and inductive definition.** We introduce graphs with dangling edges and inductive rank decomposition of them. These decompositions are an intermediate notion between rank decompositions and monoidal decompositions. Similarly to the definition of inductive branch decomposition (Section 3.2), they add to rank decompositions the algebraic flavour of monoidal decompositions by using the inductive data type of binary trees to encode a decomposition.

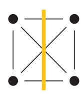
Intuitively, a graph with dangling edges is a graph equipped with some extra edges that connect some vertices in the graph to some boundary ports. This allows us to combine graphs with dangling edges by connecting some of their dangling edges. Thus, the equivalence between rank decompositions and inductive rank decompositions formalises the intuition that a rank decomposition encodes a way of dividing a graph into smaller subgraphs by "cutting" along some edges.

Definition 5.5. A *graph with dangling edges* $\Gamma = ([G], B)$ is given by an adjacency matrix $G \in \mathsf{Mat}_{\mathbb{N}}(k, k)$ that records the connectivity of the graph and a matrix $B \in \mathsf{Mat}_{\mathbb{N}}(k, n)$ that records the "dangling edges" connected to n boundary ports. We will sometimes write $G \in \mathsf{adjacency}(\Gamma)$ and $B = \mathsf{sources}(\Gamma)$.

Example 5.6. Two graphs with the same ports, as illustrated below, can be "glued" together:

 glued with  gives 

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.



$$\mathsf{rk}\left(\begin{smallmatrix} 1 & 1 \\ 1 & 1 \end{smallmatrix}\right) = 1$$

Decompositions are elements of a tree data type, with nodes carrying subgraphs Γ' of the ambient graph Γ. In the following Γ' ranges over the non-empty subgraphs of Γ: $T_\Gamma \ ::= \ (\Gamma') \ | \ (T_\Gamma—\Gamma'—T_\Gamma)$. Given $T \in T_\Gamma$, the label function λ takes a decomposition and returns the graph with dangling edges at the root: $\lambda(T_1—\Gamma—T_2) \coloneqq \Gamma$ and $\lambda((\Gamma)) \coloneqq \Gamma$.

The conditions in the definition of inductive rank decomposition ensure that, by glueing Γ_1 and Γ_2 together, we get Γ back.

Definition 5.7. Let $\Gamma = ([G], B)$ be a graph with dangling edges, where $G \in \mathsf{Mat}_{\mathbb{N}}(k, k)$ and $B \in \mathsf{Mat}_{\mathbb{N}}(k, n)$. An *inductive rank decomposition* of Γ is $T \in T_\Gamma$ where either: Γ is empty and $T = ()$; or Γ has one vertex and $T = (\Gamma)$; or $T = (T_1—\Gamma—T_2)$ and $T_i \in T_{\Gamma_i}$ are inductive rank decompositions of subgraphs $\Gamma_i = ([G_i], B_i)$ of Γ such that:

- The vertices are partitioned in two, $[G] = \left[\begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}\right]$;

- 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 $B = \left(\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right)$.

We will sometimes write $\Gamma_i = \lambda(T_i)$, $G_i = \mathsf{adjacency}(\Gamma_i)$ and $B_i = \mathsf{sources}(\Gamma_i)$. We can always assume that the rows of G and B are ordered like the leaves of T so that we can actually split B horizontally to get A_1 and A_2.

Remark 5.8. The perspective on rank width and branch width given by their inductive definitions emphasises an operational difference between them: a branch decompositon gives a recipe to construct a graph from its one-edge subgraphs by identifying some of their vertices; on the other hand, a rank decomposition gives a recipe to construct a graph from its one-vertex components by connecting some of their "dangling" edges.

Definition 5.9. Let $T = (T_1 — \Gamma — T_2)$ be an inductive rank decomposition of $\Gamma = ([G], B)$, with T_i possibly both empty. Define the *width* of T inductively: if T is empty, $\mathsf{wd}(()) \coloneqq 0$; otherwise, $\mathsf{wd}(T) \coloneqq \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2), \mathsf{rk}(B)\}$. Expanding this expression, we obtain

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

The *inductive rank width* of Γ is defined by the min-max formula $\mathsf{irwd}(\Gamma) \coloneqq \min_T \mathsf{wd}(T)$.

We show that the inductive rank width of $\Gamma = ([G], B)$ is the same as the rank width of G, up to the rank of the boundary matrix B.

Before proving the upper bound for inductive rank width, we need a technical lemma that relates the width of a graph with that of its subgraphs and allows us to compute it "globally".

Lemma 5.10. *Let T be an inductive rank decomposition of $\Gamma = ([G], B)$. Let T' be a full subtree of T and $\Gamma' \coloneqq \lambda(T')$ with $\Gamma' = ([G'], B')$. The adjacency matrix of Γ can be written as* $[G] = \left[\begin{pmatrix} G_L & C_L & C \\ \mathbb{0} & G' & C_R \\ \mathbb{0} & \mathbb{0} & G_R \end{pmatrix}\right]$ *and its boundary as* $B = \begin{pmatrix} A_L \\ A' \\ A_R \end{pmatrix}$. *Then,* $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$.

Proof. Proceed by induction on the decomposition tree T. If it is just a leaf, $T = (\Gamma)$, then Γ 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 the definition of inductive rank decomposition, $\lambda(T_i) = \Gamma_i = ([G_i], B_i)$ with $[G] = \left[\begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}\right]$, $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$, $B_1 = (A_1 \mid C)$ and $B_2 = (A_2 \mid C^\top)$. Suppose that $T' \leq T_1$. Then, we can write $[G_1] = \left[\begin{pmatrix} G_L & C_L & D' \\ \mathbb{0} & G' & D_R \\ \mathbb{0} & \mathbb{0} & H_R \end{pmatrix}\right]$, $A_1 = \begin{pmatrix} A_L \\ A' \\ F_R \end{pmatrix}$ 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, $\mathsf{rk}(B') = \mathsf{rk}(A' \mid E' \mid C_L^\top \mid D_R)$. The rank is invariant to permuting the order of columns, thus $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid D_R \mid E') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$. We proceed analogously if $T' \leq T_2$. □

The above result allows us to relate the width of rank decompositions, which is computed "globally", to the width of inductive rank decompositions, which is computed "locally", with the following bound.

Proposition 5.11. *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 Γ such that* $\mathsf{wd}(\mathcal{I}(Y, r)) \leq \mathsf{wd}(Y, r) + \mathsf{rk}(B)$.

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 Γ, $\mathcal{I}(Y, r) \coloneqq (\Gamma)$. We compute its width by definition: $\mathsf{wd}(\mathcal{I}(Y, r)) \coloneqq \mathsf{rk}(B) \leq \mathsf{wd}(Y, r) + \mathsf{rk}(B)$.

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 $V_i \coloneqq r(\mathsf{leaves}(Y_i))$ be the set of vertices associated to Y_i and $G_i \coloneqq G[V_i]$ 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 Γ. Let T' be a full subtree of T corresponding to $\Gamma' = ([G'], B')$. By Lemma 5.10, we can compute the rank of its boundary matrix $\mathsf{rk}(B') = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$, where A', C_L and C_R are defined as in the statement of Lemma 5.10. The matrix A' contains some of the rows of B, then its rank is bounded by the rank of B and we obtain $\mathsf{rk}(B') \leq \mathsf{rk}(B) + \mathsf{rk}(C_L^\top \mid C_R)$. 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, are the edges that determine $\mathsf{ord}(b')$. This means that the rank of this matrix is the order of the edge b': $\mathsf{rk}(C_L^\top \mid C_R) = \mathsf{ord}(b')$. With these observations, we can compute the width of T.

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

□

Proposition 5.12. *Let T be an inductive rank decomposition of $\Gamma = ([G], B)$ with $G \in \mathsf{Mat}_{\mathbb{N}}(k, k)$ and $B \in \mathsf{Mat}_{\mathbb{N}}(k, n)$. Then, there is a rank decomposition $\mathcal{I}^\dagger(T)$ of G such that $\mathsf{wd}(\mathcal{I}^\dagger(T)) \leq \mathsf{wd}(T)$.*

Proof. A binary tree is, in particular, a subcubic tree. Then, the rank decomposition corresponding to an inductive rank decomposition T can be defined by its underlying unlabelled tree Y. The corresponding bijection $r \colon \mathsf{leaves}(Y) \to \mathsf{vertices}(G)$ between the leaves of Y and the vertices of G can be defined by the labels of the leaves in T: the label of a leaf l of T is a subgraph of Γ with one vertex v_l and these subgraphs need to give Γ when composed together. Then, the leaves of T, which are the leaves of Y, are in bijection with the vertices of G: there is a bijection $r \colon \mathsf{leaves}(Y) \to \mathsf{vertices}(G)$ such that $r(l) \coloneqq v_l$. Then, (Y, r) is a branch decomposition of G and we can define $\mathcal{I}^\dagger(T) \coloneqq (Y, r)$.

By construction, the edges of Y are the same as the 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 $\{v, v_b\} = \mathsf{ends}(b)$ 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 Γ identified by T_b. We can express the adjacency and boundary matrices of Γ in terms of those of Γ_b:

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

By Lemma 5.10, the boundary rank of Γ_b can be computed by $\mathsf{rk}(B_b) = \mathsf{rk}(A' \mid C_L^\top \mid C_R)$. By definition, the order of the edge b is $\mathsf{ord}(b) := \mathsf{rk}(C_L^\top \mid C_R)$, and we can bound it with the boundary rank of Γ_b: $\mathsf{rk}(B_b) \geq \mathsf{ord}(b)$. These observations allow us to bound the width of the rank decomposition Y that corresponds to T.

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

□

Combining Proposition 5.11 and Proposition 5.12 we obtain:

Proposition 5.13. *Inductive rank width is equivalent to rank width.*

5.3. **A prop of graphs.** Here we recall the algebra of graphs with boundaries and its diagrammatic syntax [DLHS21]. Graphs with boundaries are graphs together with some extra "dangling" edges that connect the graph to the left and right boundaries. They compose by connecting edges that share a common boundary. All the information about connectivity is handled with matrices.

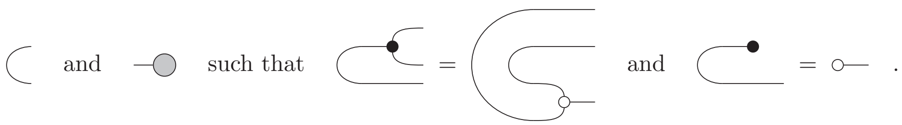
Remark 5.14. The categorical algebra of graphs with boundaries is a natural choice for capturing rank width because it emphasises the operation of splitting a graph into parts that share some *edges*. This contrasts with the algebra of cospans of graphs (Section 3.3), in which graphs are split into subgraphs that share some *vertices*. The difference in the operation that is emphasised by these two algebras reflects the difference between rank width and tree or branch width pointed out in Remark 5.8.

Definition 5.15 [DLHS21]**.** A *graph with boundaries* $g \colon n \to m$ is a tuple $g = ([G], L, R, P, [F])$ of an adjacency matrix $[G]$ of a graph on k vertices, with $G \in \mathsf{Mat}_{\mathbb{N}}(k, k)$; matrices $L \in \mathsf{Mat}_{\mathbb{N}}(k, n)$ and $R \in \mathsf{Mat}_{\mathbb{N}}(k, m)$ that record the connectivity of the vertices with the left and right boundary; a matrix $P \in \mathsf{Mat}_{\mathbb{N}}(m, n)$ that records the passing wires from the left boundary to the right one; and a matrix $F \in \mathsf{Mat}_{\mathbb{N}}(m, m)$ that records the wires from the right boundary to itself. Graphs with boundaries are taken up to an equivalence making the order of the vertices immaterial. Let $g, g' \colon n \to m$ on k vertices, with $g = ([G], L, R, P, [F])$ and $g' = ([G'], L', R', P, [F])$. The graphs g and g' are considered equal iff there is a permutation matrix $\sigma \in \mathsf{Mat}_{\mathbb{N}}(k, k)$ such that $g' = ([\sigma G \sigma^\top], \sigma L, \sigma R, P, [F])$.

Graphs with boundaries can be composed sequentially and in parallel [DLHS21], forming a symmetric monoidal category MGraph.

The prop Grph provides a convenient syntax for graphs with boundaries. It is obtained by adding a cup and a vertex generators to the prop of matrices Bialg (Figure 6).

Definition 5.16 [CS15]**.** The prop of graphs Grph is obtained by adding to Bialg the generators $\cup : 0 \to 2$ and $\mathsf{v} : 1 \to 0$ with the equations below.



These equations mean, in particular, that the cup transposes matrices (Figure 8, left) and that we can express the equivalence relation of adjacency matrices as in Definition 5.1: $[G] = [H]$ iff $G + G^\top = H + H^\top$ (Figure 8, right).

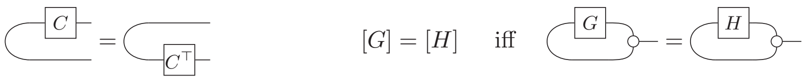
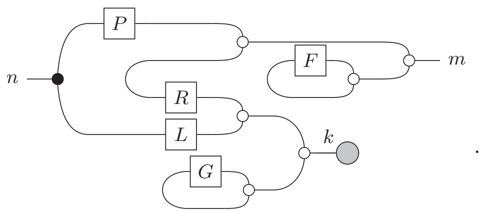


FIGURE 8. Adding the cup.

Proposition 5.17 [DLHS21, Theorem 23]**.** *The prop of graphs* Grph *is isomorphic to the prop* MGraph*.*

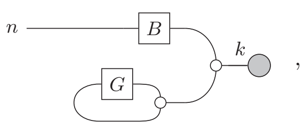
Proposition 5.17 means that the morphisms in Grph can be written in the following normal form



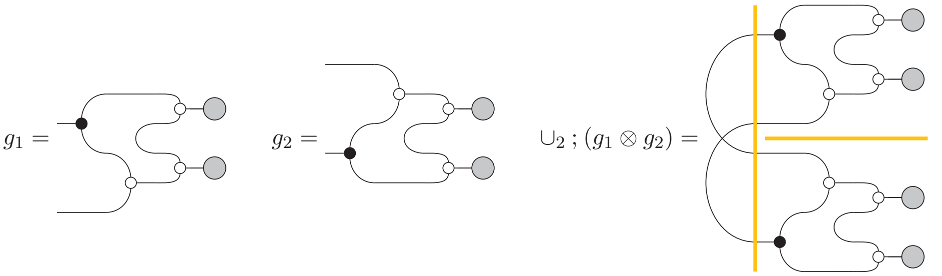
The prop Grph is more expressive than graphs with dangling edges (Definition 5.5): its morphisms can have edges between the boundaries as well. In fact, graphs with dangling edges can be seen as morphisms $n \to 0$ in Grph.

Example 5.18. A graph with dangling edges $\Gamma = ([G], B)$ can be represented as a morphism in Grph

$$g = ([G], B, ¡, !, [(\ )]) =$$



where $! : n \to 0$ and $¡ : 0 \to k$ are the unique maps to and from the terminal and initial object 0. We can now formalise the intuition of glueing graphs with dangling edges as explained in Example 5.6. The two graphs there correspond to g_1 and g_2 below left and middle. Their glueing is obtained by precomposing their monoidal product with a cup, i.e. $\cup_2 ; (g_1 \otimes g_2)$, as shown below right.



Definition 5.19. Let the set of *atomic morphisms* $\mathcal{A}$ be the set of all the morphisms of Grph. The *weight function* $\mathsf{w} \colon \mathcal{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{Grph}) \to \mathbb{N}$ is defined, on objects n, as $\mathsf{w}(n) := n$; and, on morphisms $g \in \mathcal{A}$, as $\mathsf{w}(g) := k$, where k is the number of vertices of g.

Note that, the monoidal width of g is bounded by the number k of its vertices, thus we could take as atoms all the morphisms with at most one vertex and the results would not change.

5.4. **Rank width as monoidal width.** We show that monoidal width in the prop Grph, with the weight function given in Definition 5.19, is equivalent to rank width. We do this by bounding monoidal width by above with twice rank width and by below with half of rank width (Theorem 5.26). We prove these bounds by defining maps from inductive rank decompositions to monoidal decompositions that preserve the width (Proposition 5.23), and vice versa (Proposition 5.25).

The upper bound (Proposition 5.23) is established by associating to each inductive rank decomposition a suitable monoidal decomposition. This mapping is defined inductively, given the inductive nature of both these structures. Given an inductive rank decomposition of a graph Γ, we can construct a decomposition of its corresponding morphism g as shown by the first equality in Figure 9. However, this decomposition is not optimal as it cuts along

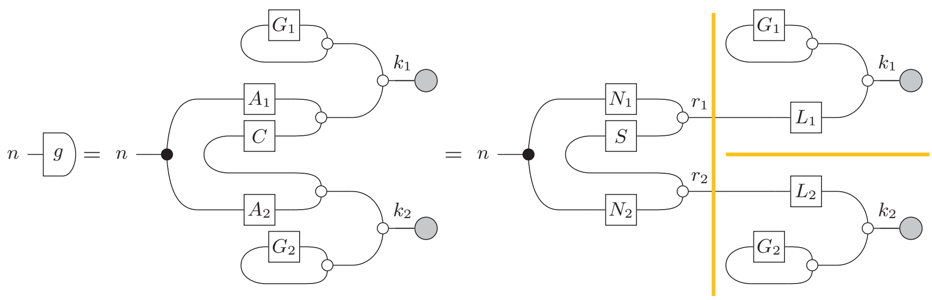
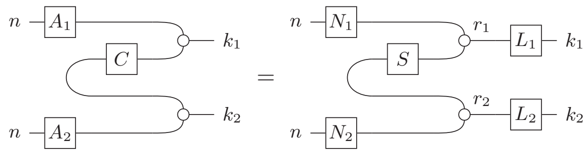


FIGURE 9. First step of a monoidal decomposition given by an inductive rank decomposition

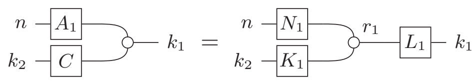
the number of vertices $k_1 + k_2$. But we can do better thanks to Lemma 5.21, which shows that we can cut along the ranks, $r_1 = \mathsf{rk}(A_1 \mid C)$ and $r_2 = \mathsf{rk}(A_2 \mid C^\top)$, of the boundaries of the induced subgraphs to obtain the second equality in Figure 9. First, recall some facts about ranks.

Remark 5.20. By Lemma 4.1, the rank of a composition of two matrices is bounded by their ranks: $\mathsf{rk}(A \cdot B) \leq \min\{\mathsf{rk}(A), \mathsf{rk}(B)\}$. If, moreover, B has full rank, then $\mathsf{rk}(A \cdot B) = \mathsf{rk}(A)$.

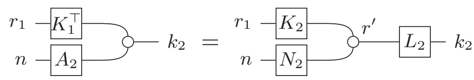
Lemma 5.21. *Let $A_i \in \mathsf{Mat}_{\mathbb{N}}(k_i, n)$, for $i = 1, 2$, and $C \in \mathsf{Mat}_{\mathbb{N}}(k_1, k_2)$. Then, there are rank decompositions of $(A_1 \mid C)$ and $(A_2 \mid C^\top)$ of the form $(A_1 \mid C) = L_1 \cdot (N_1 \mid S \cdot L_2^\top)$, and $(A_2 \mid C^\top) = L_2 \cdot (N_2 \mid S^\top \cdot L_1^\top)$. This ensures that we can decompose the diagram below on the left-hand-side as the one on the right-hand-side, where $r_1 = \mathsf{rk}(A_1 \mid C)$ and $r_2 = \mathsf{rk}(A_2 \mid C^\top)$.*



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



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

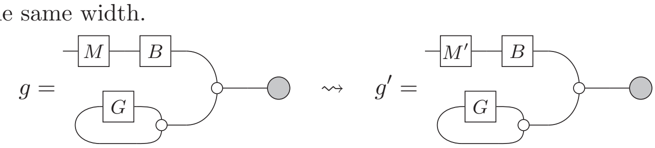


with $L_2 \in \mathsf{Mat}_{\mathbb{N}}(k_2, r')$, $N_2 \in \mathsf{Mat}_{\mathbb{N}}(r', n)$ and $K_2 \in \mathsf{Mat}_{\mathbb{N}}(r', k_1)$. We show that $r' = r_2$. By the first factorisation, we obtain that $C = L_1 \cdot K_1$, and

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

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

Once we have performed the cuts in Figure 9 on the right, we have changed the boundaries of the induced subgraphs. This means that we cannot apply the inductive hypothesis right away, but we need to transform first the inductive rank decompositions of the old subgraphs into decompositions of the new ones, as shown in Lemma 5.22. More explicitly, when M has full rank, if we have an inductive rank decomposition of $\Gamma = ([G], B \cdot M)$, which corresponds to g below left, we can obtain one of $\Gamma' = ([G], B \cdot M')$, which corresponds to g' below right, of at most the same width.



Lemma 5.22. *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 $\mathsf{wd}(T) \leq \mathsf{wd}(T')$ and such that T and T' have the same underlying tree structure. If, moreover, M' has full rank, then $\mathsf{wd}(T) = \mathsf{wd}(T')$.*

Proof. Proceed by induction on the decomposition tree T. If the tree T is just a leaf with label Γ, then we define the corresponding tree to be just a leaf with label Γ': $T' := (\Gamma')$. Clearly, T and T' have the same underlying tree structure. By Remark 5.20 and the fact that M has full rank, we can relate their widths: $\mathsf{wd}(T') := \mathsf{rk}(B \cdot M') \leq \mathsf{rk}(B) = \mathsf{rk}(B \cdot M) =: \mathsf{wd}(T)$. If, moreover, M' has full rank, the inequality becomes an equality and $\mathsf{wd}(T') = \mathsf{wd}(T)$.

If $T = (T_1 — \Gamma — T_2)$, then the adjacency and boundary matrices of Γ 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: $G = \begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}$, $B \cdot M = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \cdot M = \begin{pmatrix} A_1 \cdot M \\ A_2 \cdot M \end{pmatrix}$, 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 Γ_i can also be expressed as a composition with a full-rank matrix: $D_1 = (A_1 \cdot M \mid C) = (A_1 \mid C) \cdot \begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix}$ and $D_2 = (A_2 \cdot M \mid C^\top) = (A_2 \mid C^\top) \cdot \begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix}$. The matrices $\begin{pmatrix} M & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_i} \end{pmatrix}$ 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 $\Gamma_1' = ([G_1], B_1 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix})$ and $\Gamma_2' = ([G_2], B_2 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix})$ with the same underlying tree structure as T_1 and T_2, respectively. Moreover, their width is bounded, $\mathsf{wd}(T_i') \leq \mathsf{wd}(T_i)$, and if, additionally, M' has full rank, $\mathsf{wd}(T_i') = \mathsf{wd}(T_i)$. Then, we can use these decompositions to define an inductive rank decomposition $T' := (T_1' — \Gamma' — T_2')$ of Γ' because its adjacency and boundary matrices can be expressed in terms of those of Γ_i' as in the definition of inductive rank decomposition: $G = \begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}$, $B_1 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_2} \end{pmatrix} = (A_1 \cdot M' \mid C)$ and $B_2 \cdot \begin{pmatrix} M' & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_{k_1} \end{pmatrix} = (A_2 \cdot M' \mid C^\top)$. Applying the induction hypothesis and Remark 5.20, we compute the width of this decomposition.

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

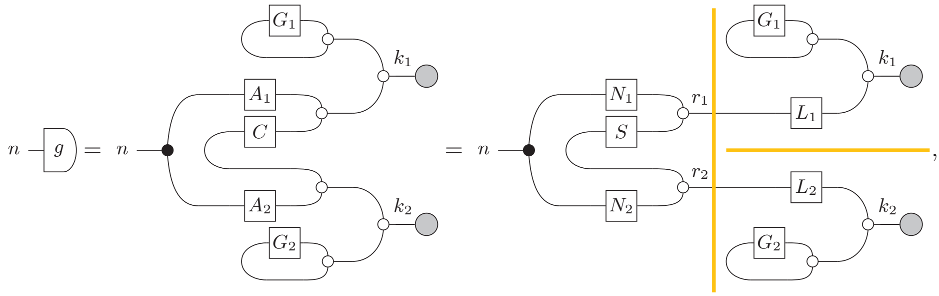
If, moreover, M' has full rank, the inequality becomes an equality and $\mathsf{wd}(T') = \mathsf{wd}(T)$. □

With the above ingredients, we can show that rank width bounds monoidal width from above.

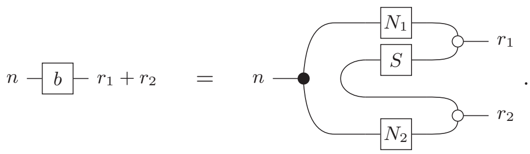
Proposition 5.23. *Let $\Gamma = ([G], B)$ be a graph with dangling edges and $g \colon n \to 0$ be the morphism in Grph corresponding to Γ. Let T be an inductive rank decomposition of Γ. Then, there is a monoidal decomposition $\mathcal{R}^\dagger(T)$ of g such that $\mathsf{wd}(\mathcal{R}^\dagger(T)) \leq 2 \cdot \mathsf{wd}(T)$.*

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

If $T = (T_1—\Gamma—T_2)$, then we can relate the adjacency and boundary matrices of Γ to those of $\Gamma_i := \lambda(T_i) = ([G_i], B_i)$, by definition of inductive rank decomposition: $G = \left(\begin{smallmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{smallmatrix}\right)$, $B = \left(\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right)$, $B_1 = (A_1 \mid C)$ and $B_2 = (A_2 \mid C^\top)$. By Lemma 5.21, 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



with $r_i = \mathsf{rk}(B_i)$. Then, $B_i = L_i \cdot M_i$ with M_i that has full rank r_i. By taking $M' = \mathbb{1}$ in Lemma 5.22, 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 $\mathsf{wd}(T_i) = \mathsf{wd}(T_i')$. Let $g_i : r_i \to 0$ be the morphisms in Grph corresponding to Γ_i' and let $b: n \to r_1 + r_2$ be defined as



By induction hypothesis, there are monoidal decompositions $\mathcal{R}^\dagger(T_1')$ and $\mathcal{R}^\dagger(T_2')$ of g_1 and g_2 of bounded width: $\mathsf{wd}(\mathcal{R}^\dagger(T_i')) \leq 2 \cdot \mathsf{wd}(T_i') = 2 \cdot \mathsf{wd}(T_i)$. Then, $g = b \,;_{r_1+r_2} (g_1 \otimes g_2)$ and $\mathcal{R}^\dagger(T) := (b— ;_{r_1+r_2} —(\mathcal{R}^\dagger(T_1')— \otimes —\mathcal{R}^\dagger(T_2')))$ is a monoidal decomposition of g. Its width can be computed.

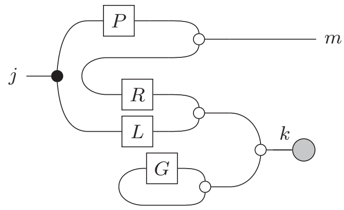
$$\begin{aligned}
&\mathsf{wd}(\mathcal{R}^\dagger(T)) \\
&:= \max\{\mathsf{w}(b), \mathsf{w}(r_1 + r_2), \mathsf{wd}(\mathcal{R}^\dagger(T_1')), \mathsf{wd}(\mathcal{R}^\dagger(T_2'))\} \\
&\leq \max\{\mathsf{w}(b), \mathsf{w}(r_1 + r_2), 2 \cdot \mathsf{wd}(T_1'), 2 \cdot \mathsf{wd}(T_2')\} \\
&=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

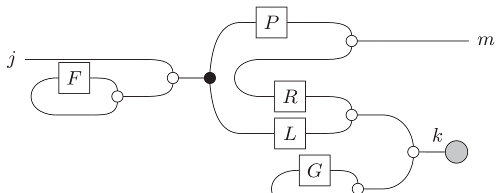
□

Proving the lower bound is similarly involved and follows a similar proof structure. From a monoidal decomposition we construct inductively an inductive rank decomposition of bounded width. The inductive step relative to composition nodes is the most involved and needs two additional lemmas, which allow us to transform inductive rank decompositions of the induced subgraphs into ones of two subgraphs that satisfy the conditions of Definition 5.7.

Applying the inductive hypothesis gives us an inductive rank decomposition of $\Gamma = ([G], (L \mid R))$, which is associated to g below left, and we need to construct one of $\Gamma' :=$

$(\left[G + L \cdot F \cdot L^\top\right], (L \mid R + L \cdot (F + F^\top) \cdot P^\top))$, which is associated to $f\,;g$ below right, of at most the same width.

$g =$ 

$f\,;g =$ 

Lemma 5.24. *Let T be an inductive rank decomposition of $\Gamma = ([G], (L \mid R))$, with $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$, $L \in \mathsf{Mat}_{\mathbb{N}}(k,j)$ and $R \in \mathsf{Mat}_{\mathbb{N}}(k,m)$. Let $F \in \mathsf{Mat}_{\mathbb{N}}(j,j)$, $P \in \mathsf{Mat}_{\mathbb{N}}(m,j)$ and define $\Gamma' := (\left[G + L \cdot F \cdot L^\top\right], (L \mid R + L \cdot (F + F^\top) \cdot P^\top))$. Then, there is an inductive rank decomposition T' of Γ' such that $\mathsf{wd}(T') \leq \mathsf{wd}(T)$.*

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

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

If $T = (T_1 \text{—} \Gamma \text{—} T_2)$, 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 Γ_i, and we can relate the adjacency and boundary matrices of Γ to those of Γ_1 and Γ_2, by definition of inductive rank decomposition: $[G] = \left[\begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}\right]$ 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 Γ' in terms of the same components: $\left[G + L \cdot F \cdot L^\top\right] = \left[\begin{pmatrix} G_1 + L_1 \cdot F \cdot L_1^\top & C + L_1 \cdot (F + F^\top) \cdot L_2^\top \\ \mathbb{0} & G_2 + L_2 \cdot F \cdot L_2^\top \end{pmatrix}\right]$ 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 Γ' and apply the induction hypothesis to them.

$$\begin{aligned} \Gamma_1' :=& (\left[G_1 + L_1 \cdot F \cdot L_1^\top\right], (L_1 \mid R_1 + L_1 \cdot (F + F^\top) \cdot P^\top \mid C + L_1 \cdot (F + F^\top) \cdot L_2^\top)) \\ =& (\left[G_1 + L_1 \cdot F \cdot L_1^\top\right], (L_1 \mid (R_1 \mid C) + L_1 \cdot (F + F^\top) \cdot (P^\top \mid L_2^\top))) \end{aligned}$$

and

$$\begin{aligned} \Gamma_2' :=& (\left[G_2 + L_2 \cdot F \cdot L_2^\top\right], (L_2 \mid R_2 + L_2 \cdot (F + F^\top) \cdot P^\top \mid C^\top + L_2 \cdot (F + F^\top) \cdot L_1^\top)) \\ =& (\left[G_2 + L_2 \cdot F \cdot L_2^\top\right], (L_2 \mid (R_2 \mid C^\top) + L_2 \cdot (F + F^\top) \cdot (P^\top \mid L_1^\top))) \end{aligned}$$

By induction, we have inductive rank decompositions T_i' of Γ_i' such that $\mathsf{wd}(T_i') \leq \mathsf{wd}(T_i)$. We defined Γ_i' so that $T' := (T_1'$—Γ'—$T_2')$ would be an inductive rank decomposition of Γ'. We can bound its width as desired.

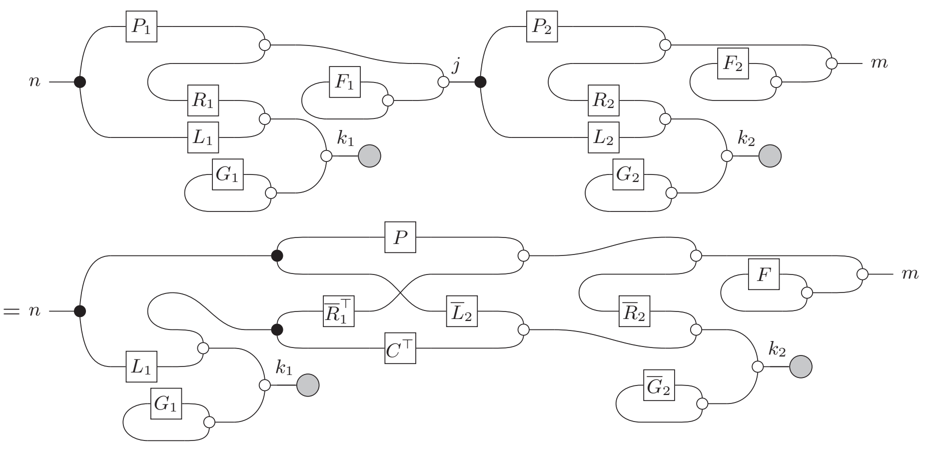
$$
\begin{aligned}
&\mathsf{wd}(T') \\
&:= \max\{\mathsf{wd}(T_1'), \mathsf{wd}(T_2'), \mathsf{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top)\} \\
&\leq \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2), \mathsf{rk}(L \mid R + L \cdot (F + F^\top) \cdot P^\top)\} \\
&\leq \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2),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

In order to obtain the subgraphs of the desired shape we need to add some extra connections to the boundaries. This can be done thanks to Lemma 5.22, by taking $M = \mathbb{1}$. We are finally able to prove the lower bound for monoidal width.

Proposition 5.25. *Let $g = ([G], L, R, P, [F])$ in* Grph *and $d \in D_g$. Let $\Gamma = ([G], (L \mid R))$. Then, there is an inductive rank decomposition $\mathcal{R}(d)$ of Γ s.t. $\mathsf{wd}(\mathcal{R}(d)) \leq 2 \cdot \max\{\mathsf{wd}(d), \mathsf{rk}(L), \mathsf{rk}(R)\}$.*

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, $\mathsf{wd}(d) := k$. Pick any inductive rank decomposition of Γ and define $\mathcal{R}(d) := T$. Surely, $\mathsf{wd}(T) \leq k =: \mathsf{wd}(d)$

If $d = (d_1$— $;_j$ —$d_2)$, then g is the composition of two morphisms: $g = g_1 \,;\, g_2$, 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 $[G] = \left[\begin{pmatrix} \overline{G}_1 & C \\ \mathbb{0} & \overline{G}_2 \end{pmatrix}\right]$ and $B = (L \mid R) = \begin{pmatrix} \overline{L}_1 & \overline{R}_1 \\ \overline{L}_2 & \overline{R}_2 \end{pmatrix}$. Then, we have that $\overline{G}_1 = G_1$, $\overline{L}_1 = L_1$, $P = P_2 \cdot P_1$, $C = R_1 \cdot L_2^\top$, $\overline{R}_1 = R_1 \cdot P_2^\top$, $\overline{L}_2 = L_2 \cdot P_1$, $\overline{R}_2 = R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top$, $\overline{G}_2 = G_2 + L_2 \cdot F_1 \cdot L_2^\top$, and $F = F_2 + P_2 \cdot F_1 \cdot P_2^\top$. This corresponds to the following diagrammatic rewriting using the equations of Grph.



We define $\overline{B}_1 := (\overline{L}_1 \mid \overline{R}_1 \mid C)$ and $\overline{B}_2 := (\overline{L}_2 \mid \overline{R}_2 \mid C^\top)$. In order to build an inductive rank decomposition of Γ, we need rank decompositions of $\overline{\Gamma}_i = ([\overline{G}_i], \overline{B}_i)$. We obtain these in three steps. Firstly, we apply induction to obtain inductive rank decompositions $\mathcal{R}(d_i)$ of $\Gamma_i = ([G_i], (L_i \mid R_i))$ such that $\mathsf{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\mathsf{wd}(d_i), \mathsf{rk}(L_i), \mathsf{rk}(R_i)\}$. Secondly, we apply

Lemma 5.24 to obtain an inductive rank decomposition T_2' of $\Gamma_2' = (\left[G_2 + L_2 \cdot F_1 \cdot L_2^\top\right], (L_2 \mid R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top))$ such that $\mathsf{wd}(T_2') \leq \mathsf{wd}(\mathcal{R}(d_2))$. Lastly, we observe that $(\overline{R}_1 \mid C) = R_1 \cdot (P_2^\top \mid L_2^\top)$ and $(\overline{L}_2 \mid C^\top) = L_2 \cdot (P_1 \mid R_1^\top)$. Then we obtain that $\overline{B}_1 = (L_1 \mid R_1) \cdot \begin{pmatrix} \mathbb{1}_n & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & P_2^\top & L_2^\top \end{pmatrix}$ and $\overline{B}_2 = (L_2 \mid R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top) \cdot \begin{pmatrix} P_1 & \mathbb{0} & R_1^\top \\ \mathbb{0} & \mathbb{1}_m & \mathbb{0} \end{pmatrix}$, and we can apply Lemma 5.22, with $M = \mathbb{1}$, to get inductive rank decompositions T_i of $\overline{\Gamma}_i$ such that $\mathsf{wd}(T_1) \leq \mathsf{wd}(\mathcal{R}(d_1))$ and $\mathsf{wd}(T_2) \leq \mathsf{wd}(T_2') \leq \mathsf{wd}(\mathcal{R}(d_2))$. If $k_1, k_2 > 0$, then we define $\mathcal{R}(d) \coloneqq (T_1 — \Gamma — T_2)$, which is an inductive rank decomposition of Γ because $\overline{\Gamma}_i$ satisfy the two conditions in Definition 5.7. If $k_1 = 0$, then $\Gamma = \overline{\Gamma}_2$ and we can define $\mathcal{R}(d) \coloneqq T_2$. Similarly, if $k_2 = 0$, then $\Gamma = \overline{\Gamma}_1$ and we can define $\mathcal{R}(d) \coloneqq T_1$. In any case, we can compute the width of $\mathcal{R}(d)$ (if $k_i = 0$ then $T_i = ()$ and $\mathsf{wd}(T_i) = 0$) using the inductive hypothesis, Lemma 5.24, Lemma 5.22, the fact that $\mathsf{rk}(L) \geq \mathsf{rk}(L_1)$, $\mathsf{rk}(R) \geq \mathsf{rk}(R_2)$ and $j \geq \mathsf{rk}(R_1), \mathsf{rk}(L_2)$ because $R_1 \colon j \to k_1$ and $L_2 \colon j \to k_2$.

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

If $d = (d_1 — \otimes — d_2)$, then g is the monoidal product of two morphisms: $g = g_1 \otimes g_2$, with $g_i = ([G_i], L_i, R_i, P_i, [F_i]) \colon n_i \to m_i$. By exlicitly computing the monoidal product, we obtain that $[G] = \left[\begin{pmatrix} G_1 & \mathbb{0} \\ \mathbb{0} & G_2 \end{pmatrix}\right]$, $L = \begin{pmatrix} L_1 & \mathbb{0} \\ \mathbb{0} & L_2 \end{pmatrix}$, $R = \begin{pmatrix} R_1 & \mathbb{0} \\ \mathbb{0} & R_2 \end{pmatrix}$, $P = \begin{pmatrix} P_1 & \mathbb{0} \\ \mathbb{0} & P_2 \end{pmatrix}$ and $F = \begin{pmatrix} F_1 & \mathbb{0} \\ \mathbb{0} & F_2 \end{pmatrix}$. By induction, we have inductive rank decompositions $\mathcal{R}(d_i)$ of $\Gamma_i \coloneqq ([G_i], B_i)$, where $B_i = (L_i \mid R_i)$, of bounded width: $\mathsf{wd}(\mathcal{R}(d_i)) \leq 2 \cdot \max\{\mathsf{wd}(d_i), \mathsf{rk}(L_i), \mathsf{rk}(R_i)\}$. Let $\overline{B}_1 \coloneqq (L_1 \mid \mathbb{0}_{n_2} \mid R_1 \mid \mathbb{0}_{m_2} \mid \mathbb{0}_{k_2}) = B_1 \cdot \begin{pmatrix} \mathbb{1}_{n_1} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{m_1} & \mathbb{0} & \mathbb{0} \end{pmatrix}$ and $\overline{B}_2 \coloneqq (\mathbb{0}_{n_1} \mid L_2 \mid \mathbb{0}_{m_1} \mid R_2 \mid \mathbb{0}_{k_1}) = B_2 \cdot \begin{pmatrix} \mathbb{0} & \mathbb{1}_{n_2} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{m_2} & \mathbb{0} \end{pmatrix}$. By taking $M = \mathbb{1}$ in Lemma 5.22, we can obtain inductive rank decompositions T_i of $\overline{\Gamma}_i \coloneqq ([G_i], \overline{B}_i)$ such that $\mathsf{wd}(T_i) \leq \mathsf{wd}(\mathcal{R}(d_i))$. If $k_1, k_2 > 0$, then we define $\mathcal{R}(d) \coloneqq (T_1 — \Gamma — T_2)$, which is an inductive rank decomposition of Γ because $\overline{\Gamma}_i$ satisfy the two conditions in Definition 5.7. If $k_1 = 0$, then $\Gamma = \overline{\Gamma}_2$ and we can define $\mathcal{R}(d) \coloneqq T_2$. Similarly, if $k_2 = 0$, then $\Gamma = \overline{\Gamma}_1$ and we can define $\mathcal{R}(d) \coloneqq T_1$. In any case, we can compute the width of $\mathcal{R}(d)$ (if $k_i = 0$ then $T_i = ()$ and $\mathsf{wd}(T_i) = 0$) using the inductive hypothesis and Lemma 5.22.

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

$$\leq 2 \cdot \max\{\mathsf{wd}(d_1), \mathsf{rk}(L_1), \mathsf{rk}(R_1), \mathsf{wd}(d_2), \mathsf{rk}(L_2), \mathsf{rk}(R_2), \mathsf{rk}(L), \mathsf{rk}(R)\}$$
$$\leq 2 \cdot \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2), \mathsf{rk}(L), \mathsf{rk}(R)\}$$
$$=: 2 \cdot \max\{\mathsf{wd}(d), \mathsf{rk}(L), \mathsf{rk}(R)\} \qquad \square$$

From Proposition 5.23, Proposition 5.25 and Proposition 5.13, we obtain the main result of this section.

Theorem 5.26. *Let G be a graph and let $g = ([G], ¡, ¡, (\ ), [(\ )])$ be the corresponding morphism in* Grph. *Then,* $\frac{1}{2} \cdot \mathsf{rwd}(G) \leq \mathsf{mwd}(g) \leq 2 \cdot \mathsf{rwd}(G)$.

6. Conclusion and future work

We defined monoidal width for measuring the complexity of morphisms in monoidal categories. The concrete examples that we aimed to capture are tree width and rank width. In fact, we have shown that, by choosing suitable categorical algebras, monoidal width is equivalent to these widths. We have also related monoidal width to the rank of matrices over the natural numbers.

Our future goal is to leverage the generality of monoidal categories to study other examples outside the graph theory literature. In the same way Courcelle's theorem gives fixed-parameter tractability of a class of problems on graphs with parameter tree width or rank width, we aim to obtain fixed-parameter tractability of a class of problems on morphisms of monoidal categories with parameter monoidal width. This result would rely on Feferman-Vaught-Mostowski type theorems specific to the operations of a particular monoidal category C or particular class of monoidal categories, which would ensure that the problems at hand respect the compositional structure of these categories.

Conjecture. Computing a compositional problem on the set of morphisms $\mathsf{C}_k(X, Y)$ with k-bounded monoidal width with a compositional algorithm is linear in w. Explicitly, computing the solution on $f \in \mathsf{C}_k(X, Y)$ takes $\mathcal{O}(c(k) \cdot \mathsf{w}(f))$, for some more than exponential function $c \colon \mathbb{N} \to \mathbb{N}$.

References

[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)*, pages 1–12. IEEE, 2017. doi:10.1109/LICS.2017.8005129.

[AM21] Samson Abramsky and Dan Marsden. Comonadic semantics for guarded fragments. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470594.

[AS21] Samson Abramsky and Nihil Shah. Relating structure and power: Comonadic semantics for computational resources. *Journal of Logic and Computation*, 31(6):1390–1428, 2021. doi:10.1093/logcom/exab048.

[BB73] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *Journal of Combinatorial Theory, Series A*, 14(2):137–148, 1973. doi:10.1016/0097-3165(73)90016-2.

[BBFK11] Christoph Blume, HJ Sander Bruggink, Martin Friedrich, and Barbara König. Treewidth, pathwidth and cospan decompositions. *Electronic Communications of the EASST*, 41, 2011. doi:10.14279/tuj.eceasst.41.643.

[BC87] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(1):83–127, 1987. doi:10.1007/BF01692060.

[BK08] Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

[BK21] Benjamin Merlin Bumpus and Zoltan A Kocsis. Spined categories: generalizing tree-width beyond graphs, 2021. arXiv:2104.01841.

[BKM23] Benjamin Merlin Bumpus, Zoltan Kocsis, and Jade Edenstar Master. Structured decompositions: Structural and algorithmic compositionality, 2023. arXiv:2207.06091.

[Bod92] Hans L Bodlaender. A tourist guide through treewidth. *Technical report*, 1992.

[BS21] Guillaume Boisseau and Paweł Sobociński. String diagrammatic electrical circuit theory. In Kohei Kishida, editor, Proceedings of the Fourth International Conference on *Applied Category Theory,* Cambridge, United Kingdom, 12-16th July 2021, volume 372 of *Electronic Proceedings in Theoretical Computer Science*, pages 178–191. Open Publishing Association, 2021. doi:10.4204/EPTCS.372.13.

[BSZ21] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A survey of compositional signal flow theory. In Michael Goedicke, Erich J. Neuhold, and Kai Rannenberg, editors, *Advancing Research in Information and Communication Technology - IFIP's Exciting First 60+ Years, Views from the Technical Committees and Working Groups*, volume 600 of *IFIP Advances in Information and Communication Technology*, pages 29–56. Springer, 2021. doi:10.1007/978-3-030-81701-5_2.

[Bum21] Benjamin Merlin Bumpus. *Generalizing graph decompositions*. PhD thesis, University of Glasgow, 2021.

[CD21] Adam Ó Conghaile and Anuj Dawar. Game comonads & generalised quantifiers. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:17, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2021.16.

[CJ19] Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019. doi:10.1017/S0960129518000488.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. doi:10.1017/9781316219317.

[CK22] Cole Comfort and Aleks Kissinger. A graphical calculus for lagrangian relations. In Kohei Kishida, editor, Proceedings of the Fourth International Conference on *Applied Category Theory,* Cambridge, United Kingdom, 12-16th July 2021, volume 372 of *Electronic Proceedings in Theoretical Computer Science*, pages 338–351. Open Publishing Association, 2022. doi:10.4204/EPTCS.372.24.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs I: Recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

[Cou92] Bruno Courcelle. The monadic second-order logic of graphs III: Tree-decompositions, minors and complexity issues. *RAIRO-Theoretical Informatics and Applications*, 26(3):257–286, 1992. doi:10.1051/ita/1992260302571.

[CS15] Apiwat Chantawibul and Paweł Sobociński. Towards compositional graph theory. *Electronic Notes in Theoretical Computer Science*, 319:121–136, 2015. doi:10.1016/j.entcs.2015.12.009.

[DKPvdW20] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the ZX-calculus. *Quantum*, 4:279, 2020. doi:10.22331/q-2020-06-04-279.

[DLHS21] Elena Di Lavore, Jules Hedges, and Paweł Sobociński. Compositional modelling of network games. In Christel Baier and Jean Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*, volume 183 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 30:1–30:24, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CSL.2021.30.

[DLS21] Elena Di Lavore and Paweł Sobociński. Monoidal Width: Unifying Tree Width, Path Width and Branch Width, 2021. arXiv:2202.07582.

[DLS22] Elena Di Lavore and Paweł Sobociński. Monoidal Width: Capturing Rank Width, to appear in ACT 2022. arXiv:2202.07582.

[Fon15] Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.

[Fri20] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020.

[FS18] Brendan Fong and David I Spivak. Seven sketches in compositionality: An invitation to applied category theory, 2018. arXiv:1803.05316.

[GH97] Fabio Gadducci and Reiko Heckel. An inductive view of graph transformation. In Francesco Parisi Presicce, editor, *International Workshop on Algebraic Development Techniques*, pages 223–237. Springer, 1997. doi:10.1007/3-540-64299-4_36.

[GHWZ18] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481. ACM, 2018. doi:10.1145/3209108.3209165.

[GJL17] Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic semantics for digital circuits. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.24.

[Hal76] Rudolf Halin. S-functions for graphs. *Journal of Geometry*, 8(1):171–186, 1976. doi:10.1007/BF01917434.

[JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

[Lac04] Stephen Lack. Composing props. *Theory and Applications of Categories*, 13(9):147–163, 2004.

[Mac78] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.

[Mas22] Jade Master. How to compose shortest paths, 2022. arXiv:2205.15306.

[MS22] Yoàv 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 (LICS)*, pages 1–11. ACM, 2022. doi:10.1145/3531130.3533335.

[OS06] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

[PO99] R Piziak and PL Odell. Full rank factorization of matrices. *Mathematics magazine*, 72(3):193–201, 1999. doi:10.1080/0025570X.1999.11996730.

[PRS88] Pavel Pudlák, Vojtěch Rödl, and Petr Savickỳ. Graph complexity. *Acta Informatica*, 25(5):515–535, 1988. doi:10.1007/bf00279952.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-n.

[RSS14] Julian Rathke, Paweł Sobociński, and Owen Stephens. Compositional reachability in Petri nets. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *International Workshop on Reachability Problems, Lecture Notes in Computer Science*, pages 230–243. Springer, 2014. doi:10.1007/978-3-319-11439-2_18.

[RSW05] Robert Rosebrugh, Nicoletta Sabadini, and Robert FC Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15(6):164–177, 2005.

[Zan15] Fabio Zanasi. *Interacting Hopf Algebras - The Theory of Linear Systems*. PhD thesis, École Normale Supérieure de Lyon, 2015.

Monoidal Width: Capturing Rank Width

Elena Di Lavore
Tallinn University of Technology

Paweł Sobociński
Tallinn University of Technology

Monoidal width was recently introduced by the authors as a measure of the complexity of decomposing morphisms in monoidal categories. We have shown that in a monoidal category of cospans of graphs, monoidal width and its variants can be used to capture tree width, path width and branch width. In this paper we study monoidal width in a category of matrices, and in an extension to a *different* monoidal category of open graphs, where the connectivity information is handled with matrix algebra and graphs are composed along edges instead of vertices. We show that here monoidal width captures rank width: a measure of graph complexity that has received much attention in recent years.

1 Introduction

Many applications of category theory rely on monoidal categories as algebras of processes [26, 15, 28, 18, 10, 25, 11, 17, 23, 27]. Morphisms are compound processes, defined as parallel and sequential compositions of simpler process components. The compositional nature of this modelling allows a compositional computation of the underlying semantics. But how efficient is this computation? Given two processes f and g, we can compute their semantics separately. However, computing the semantics of their sequential composition $f\,;g$ often requires an additional cost [36]. Indeed, the semantics of sequential composition often means resource sharing or synchronisation along the common boundary. This in turn carries a computational burden, dependent on the size of the boundary. On the other hand, computing the semantics of a parallel composition $f \otimes f'$ typically does not involve any resource sharing, as indicated by the wiring of the string diagrams, and thus typically does not require significant additional computational resources. Taking this into account, the choice of the *recipe* for a morphism in terms of parallel and sequential compositions influences the cost of computing its semantics. As shown in Figure 1, where vertical cuts represent sequential compositions and horizontal cuts represent parallel compositions, the same morphism can be defined in different ways with possibly different computational costs. Given a morphism, it is thefore desirable to find the least costly recipe of *decomposing* it in terms

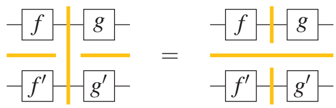


Figure 1: Two monoidal decompositions of the same morphism, the right one being the cheapest.

of more primitive components. We can rephrase the original question: what is the most efficient way to decompose a morphism in a monoidal category?

The authors recently proposed *monoidal width* [22] as a way of assigning a natural number to a morphism of a monoidal category, representing – roughly speaking – the cost of its most efficient decomposition. In turn, this is related to the cost of computing the semantics of this morphism.

Computing efficient decompositions is not a new problem. The graph theory literature abounds [6, 29, 38, 37, 39, 33, 20, 2, 3, 16] with notions of complexity of graphs that ultimately measure the difficulty

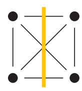
J. Master & M. Lewis (Eds.): Fifth International Conference on Applied Category Theory (ACT 2022).
EPTCS 380, 2023, pp. 268–283, doi:10.4204/EPTCS.380.16

of decomposing a graph into smaller components by cutting along the vertices or the edges of the graph. Measures such as tree width [6, 29, 38], path width [37], branch width [39], clique width [20] and rank width [33] are motivated by algorithmic considerations. Probably the best known among several results that etablish links with algorithms [8, 9, 19], the following shows the importance of tree width.

Theorem (Courcelle [19])**.** *Every property expressible in the monadic second order logic of graphs can be tested in linear time on graphs with bounded tree width.*

The different notions of complexity for graphs vastly differ in low-level "implementation details" but they all share a similar underlying idea: that of defining decompositions and suitably measuring their efficiency. One of our contributions is to exhibit monoidal width as a unifying framework for graph measures based on a notion of decomposition. In fact, by choosing a suitable algebra of composition for graphs — i.e. choosing the right monoidal cateory — we recover some of these known measures as particular instances of monoidal width. We have previously captured [22] tree width, path width and branch width by instantiating monoidal width and two variants in a category of cospans of graphs.

In this paper we focus on rank width [33] – a relatively recent development that has attracted significant attention in the graph theory community. A rank decomposition is a recipe for decomposing a graph into its single-vertex subgraphs by cutting along its edges. The cost of a cut is the rank of the adjacency matrix that represents it, as shown in Figure 2. A useful intuition for rank width is that it is a kind of "Kolmogorov complexity" for graphs. For example, although the family of cliques has unbounded tree width, the connectivity of cliques is quite simple to describe: and, in fact, all cliques have rank width 1.



$$\mathsf{rank}\begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix} = 1$$

Figure 2: A cut and its matrix in a rank decomposition.

To capture rank width as an instance of monoidal width, rather than taking cospans, we work in a different monoidal category of graphs. First introduced in [14], it was recently used [21] as a syntax for network games. This approach to computing with "open graphs" is more linear algebraic, building modularly on the theory of bialgebra, well known to be closely related to matrix algebra [41]. Indeed, the connectivity of graphs is handled with adjacency matrices, and boundary connections are matrices.

Related work. This manuscript, although self-contained, complements our previous work [22], where we considered tree width, path width and branch width as instances of monoidal width.

Previous syntactical approaches to graph widths are the work of Pudlák, Rödl and Savickỳ [35] and the work of Bauderon and Courcelle [5]. Their works consider different notions of graph decompositions, which lead to different notions of graph complexity. In particular, in [5], the cost of a decomposition is measured by counting *shared names*, which is clearly closely related to penalising sequential composition as in monoidal width. Nevertheless, these approaches are specific to particular, concrete notions of graphs, whereas our work concerns the more general algebraic framework of monoidal categories.

Recent abstract approaches focus on other graph widths. The work of Blume et. al. [7], characterises tree and path decompositions in terms of colimits. Abramsky et. al. [24] give a coalgebraic characterization of tree width of relational structures (and graphs in particular). Bumpus and Kocsis [13] also generalise tree width to the categorical setting, although their approach is far removed from ours.

Synopsis. Monoidal width is recalled in Section 2. In Section 3, we study the monoidal width of matrices. Section 4 recalls rank width and gives an equivalent recursive definition of it that will be useful as an intermediate step towards our main result, which is presented in Section 5.

Preliminaries. We use string diagrams [30, 40]: sequential and parallel compositions of f and g are drawn as in Figure 3, left and middle, respectively. Much of the bureaucracy, e.g. the interchange law $(f\,;g)\otimes(f';g') = (f\otimes f');(g\otimes g')$, disappears (Figure 3, right). *Props* [32, 31] are important examples of

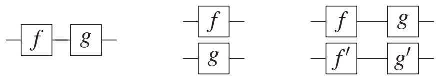


Figure 3: String diagrammatic notation.

monoidal categories. They are symmetric strict monoidal, with natural numbers as objects, and addition as monoidal product on objects. Roughly speaking, morphisms can be thought of as processes, and the objects (natural numbers) keep track of the number of inputs or outputs of a process.

2 Monoidal width

This section recalls the concept of monoidal width from [22]. Monoidal width records the cost of the most efficient way one can decompose a morphism into its atomic components, thus capturing—roughly speaking—its intrinsic structural complexity. A decomposition is a binary tree whose internal nodes are labelled with compositions or monoidal products, and whose leaves are labelled with atomic morphisms.

Definition 2.1 (Monoidal decomposition [22])**.** Let C be a monoidal category and $\mathscr{A}$ be a subset of its morphisms referred to as *atomic*. The set D_f of *monoidal decompositions* of $f\colon A\to B$ in C is defined:

$$
\begin{array}{rcll}
D_f & ::= & (f) & \text{if } f\in\mathscr{A}\\
 & \mid & (d_1,\otimes,d_2) & \text{if } d_1\in D_{f_1},\, d_2\in D_{f_2} \text{ and } f = f_1\otimes f_2\\
 & \mid & (d_1,;_X,d_2) & \text{if } d_1\in D_{f_1\colon A\to X},\, d_2\in D_{f_2\colon X\to B} \text{ and } f = f_1\,;f_2
\end{array}
$$

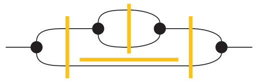
The cost of a decomposition depends on the operations and atoms present: each operation and each atomic morphism is associated with a cost, which we call weight. Roughly speaking, sequential composition is priced according to the size of the object the composition occurs over, while monoidal products are free. Finally, the weight of an atom is the application-specific cost of computing its semantics.

Definition 2.2 (Weight function [22])**.** Let C be a monoidal category and let $\mathscr{A}$ be a set of atoms for C. A weight function for $(\mathsf{C},\mathscr{A})$ is a function $\mathsf{w}\colon \mathscr{A}\cup\{\otimes\}\cup\mathsf{Obj}(\mathsf{C})\to\mathbb{N}$ such that

- $\mathsf{w}(X\otimes Y) = \mathsf{w}(X)+\mathsf{w}(Y)$,
- $\mathsf{w}(\otimes) = 0$.

Example 2.3. *Let* $\Delta_1 : 1\to 2$ *and* $\nabla_1 : 2\to 1$ *be the diagonal and codiagonal morphisms in a cartesian and cocartesian prop*[1] *s.t.* $\mathsf{w}(\Delta_1) = \mathsf{w}(\nabla_1) = 2$. *The following figure represents the monoidal decomposition of* $\Delta\,;(\Delta\otimes\text{—});(\nabla\otimes\text{—});\nabla$ *given by* $(\Delta,;_2,(((\Delta,;_2,\nabla),\otimes,\text{—}),;_2,\nabla))$.

[1] In a *cartesian prop* the $\otimes$ satisfies the universal property of products. Dually, in a *cocartesian prop*, the $\otimes$ satisfies the universal property of the coproduct.



The width of a decomposition is the cost of the most expensive node in the decomposition tree.

Definition 2.4 (Width of a monoidal decomposition [22])**.** Let w be a weight function for $(\mathsf{C},\mathscr{A})$. Let f be in C and $d \in D_f$. The width of d is defined recursively as follows:

$$\begin{aligned} \mathsf{wd}(d) &:= \mathsf{w}(f) && \text{if } d = (f) \\ & \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2)\} && \text{if } d = (d_1, \otimes, d_2) \\ & \max\{\mathsf{wd}(d_1), \mathsf{w}(X), \mathsf{wd}(d_2)\} && \text{if } d = (d_1, ;_X, d_2) \end{aligned}$$

As sketched in Example 2.3, decompositions can be seen as labelled trees (S,μ) where S is a tree and $\mu : \mathsf{vertices}(S) \to \mathscr{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{C})$ is a labelling function. With this we can restate the width as:

$$\mathsf{wd}(d) = \mathsf{wd}(S,\mu) := \max_{v \in \mathsf{vertices}(S)} \mathsf{w}(\mu(v))$$

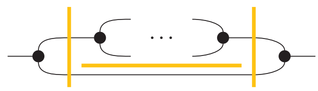
which may be familiar to those aquainted with graph widths.

Monoidal width is simply the width of the cheapest decomposition.

Definition 2.5 (Monoidal width [22])**.** Let w be a weight function for $(\mathsf{C},\mathscr{A})$ and f be in C. Then the *monoidal width* of f is $\mathsf{mwd}(f) := \min_{d \in D_f} \mathsf{wd}(d)$.

Example 2.6. *With the data of Example 2.3, define a family of morphisms $n \colon 1 \to 1$ inductively:*

- $1 := \text{—}_1$;
- $2 := \prec ;_2 \succ$;
- $n+1 := \prec ;_2 (n \otimes 1) ;_2 \succ$ *for* $n \geq 2$.



Each n has a monoidal decomposition of width n: the root node is the composition along the n wires in the middle. However, $\mathsf{mwd}(n) = 2$ *for any n, with an optimal decomposition shown above.*

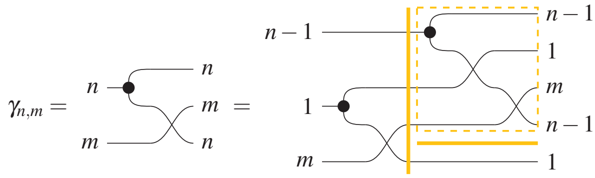
2.1 The width of copying

Before we begin with the original technical contributions of this paper in Section 3, we need to recall a technical result from [22] about decomposing copy morphisms. We consider symmetric monoidal categories equipped with such morphisms and show that copying n wires costs at most $n+1$.

Definition 2.7 (Copying)**.** Let X be a symmetric monoidal category with symmetries given by $\times_{X,Y}$. We say that X has *coherent copying* if there is a class of objects $\mathscr{C}_{\mathsf{X}} \subseteq \mathsf{Obj}(\mathsf{X})$, satisfying $X,Y \in \mathscr{C}_{\mathsf{X}}$ iff $X \otimes Y \in \mathscr{C}_{\mathsf{X}}$, such that every X in $\mathscr{C}_{\mathsf{X}}$ is endowed with a morphism $\prec_X \colon X \to X \otimes X$. Moreover, $\prec_{X \otimes Y} = (\prec_X \otimes \prec_Y) ; (\text{—}_X \otimes \times_{X,Y} \otimes \text{—}_Y)$ for every $X,Y \in \mathscr{C}_{\mathsf{X}}$.

An example is any cartesian prop with $\prec_n \colon n \to n+n$ given by the cartesian structure. We take $\prec_X$, the symmetries $\times_{X,Y}$ and the identities —_X as atomic for all objects X,Y, i.e. the set of atomic morphisms is $\mathscr{A} = \{\prec_X, \times_{X,Y}, \text{—}_X \ : \ X,Y \in \mathscr{C}_{\mathsf{X}}\}$. The weight function is $\mathsf{w}(\prec_X) := 2 \cdot \mathsf{w}(X)$, $\mathsf{w}(\times_{X,Y}) := \mathsf{w}(X) + \mathsf{w}(Y)$ and $\mathsf{w}(\text{—}_X) := \mathsf{w}(X)$. In a prop, we take $\mathsf{w}(n) := n$. Note that $\mathsf{w}(\prec_{X \otimes Y}) = 2 \cdot \mathsf{w}(X \otimes Y) = 2 \cdot (\mathsf{w}(X) + \mathsf{w}(Y))$, but utilising coherence we can do better, as illustrated below.

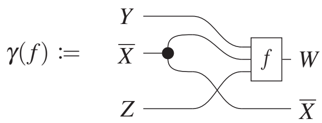
Example 2.8. *Let C be a prop with coherent copying and consider $\prec_n \colon n \to 2n$. Let $\gamma_{n,m} := (\prec_n \otimes \text{—}_m) ; (\text{—}_n \otimes \times_{n,m}) \colon n+m \to n+m+n$. We can decompose $\gamma_{n,m}$ in terms of $\gamma_{n-1,m+1}$ (in the dashed box), $\prec_1$ and $\times_{1,1}$ by cutting along at most $n+1+m$ wires:*



This allows us to decompose $-\!\!\bullet\!\!<_n = \gamma_{n,0}$ *cutting along at most* $n+1$ *wires. In particular,* $\mathsf{mwd}(-\!\!\bullet\!\!<_n) \leq n+1$.

The following result is a technical generalisation of the argument presented in Example 2.8.

Lemma 2.9 ([22]). *Let* X *be a symmetric monoidal category with coherent copying. Suppose that* $\mathscr{A}$ *contains* $-\!\!\bullet\!\!<_X$ *for* $X \in \mathscr{C}_{\mathsf{X}}$*, and* $\times_{X,Y}$ *and* $-_X$ *for* $X \in \mathsf{Obj}(\mathsf{X})$*. Let* $\overline{X} := X_1 \otimes \cdots \otimes X_n$*,* $f \colon Y \otimes \overline{X} \otimes Z \to W$ *and let* $d \in D_f$*. Let* $\gamma(f) := (-_Y \otimes -\!\!\bullet\!\!<_{\overline{X}} \otimes -_Z) ; (-_{Y \otimes \overline{X}} \otimes \times_{\overline{X},Z}) ; (f \otimes -_{\overline{X}})$.



There is a decomposition $\mathscr{C}(d)$ *of* $\gamma(f)$ *of bounded width:*

$$\mathsf{wd}(\mathscr{C}(d)) \leq \max\{\mathsf{wd}(d), \mathsf{w}(Y) + \mathsf{w}(Z) + (n+1) \cdot \max_{i=1,\ldots,n} \mathsf{w}(X_i)\}.$$

3 Monoidal width in matrices

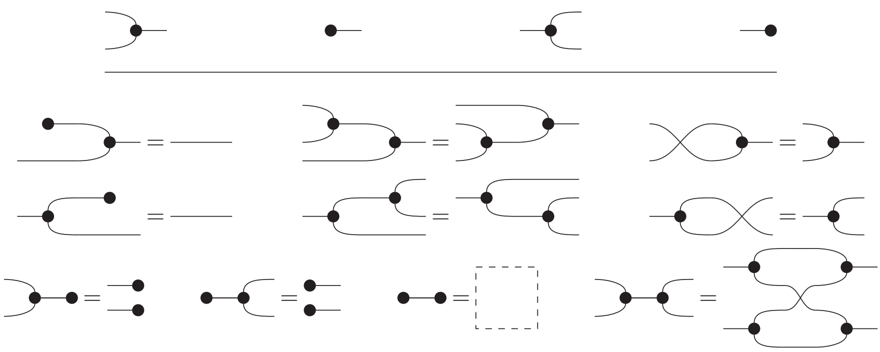


Figure 4: Bialgebra axioms

Given the ubiquity of matrix algebra, matrices are an obvious case study. Theorem 3.12 shows that the monoidal width of a matrix is, up to 1, the maximum of the ranks of its blocks.

Consider the monoidal category $\mathsf{Mat}_{\mathbb{N}}$ of matrices with entries in the natural numbers. The objects are natural numbers and morphisms from n to m are m by n matrices. Composition is the usual product of matrices and the monoidal product is the biproduct: $A \otimes B := \left(\begin{smallmatrix} A & 0 \\ 0 & B \end{smallmatrix}\right)$. Let us examine matrix decompositions enabled by this algebra. A matrix A can be written as a monoidal product $A = A_1 \otimes A_2$ iff the matrix has blocks A_1 and A_2, i.e. $A = \begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$. On the other hand, a composition is related to the rank.

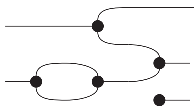
Lemma 3.1 ([34])**.** *Let* $A \colon n \to m$ *in* $\mathsf{Mat}_{\mathbb{N}}$*. Then* $\min\{k \in \mathbb{N} : A = B \,;_k C\} = \mathsf{rank}(A)$.

We first introduce a convenient syntax for matrices.

Proposition 3.2 ([41])**.** *The category* $\mathsf{Mat}_{\mathbb{N}}$ *is isomorphic to the prop* Bialg*, generated by* -●< $: 1 \to 2$, -● $: 1 \to 0$, >●- $: 2 \to 1$ *and* ●- $: 0 \to 1$ *and quotiented by bialgebra axioms (Figure 4).*

For the uninitiated reader, let us briefly explain this correspondence. Every morphism $f \colon n \to m$ in Bialg corresponds to a matrix $A = \mathsf{Mat}(f) \in \mathsf{Mat}_{\mathbb{N}}(m,n)$: we can read the (i, j)-entry of A off the diagram of f by counting the number of paths from the jth input to the ith output.

Example 3.3. *The matrix* $\begin{pmatrix} 1 & 0 \\ 1 & 2 \\ 0 & 0 \end{pmatrix} \in \mathsf{Mat}_{\mathbb{N}}(3,2)$ *corresponds to*



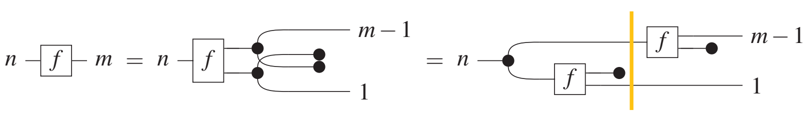
Definition 3.4. The atomic morphisms $\mathscr{A}$ are the generators of Bialg, with the symmetry and identity on 1: $\mathscr{A} = \{$-●<, -●, >●-, ●-, ×, $—_1\}$. The weight $\mathsf{w} \colon \mathscr{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{Bialg}) \to \mathbb{N}$ has $\mathsf{w}(n) := n$, for any $n \in \mathbb{N}$, and $\mathsf{w}(g) := \max\{m,n\}$, for $g \colon n \to m \in \mathscr{A}$.

3.1 Monoidal width in Bialg

The characterisation of the rank of a matrix in Lemma 3.1 hints at some relationship between the monoidal width of a matrix and its rank. In fact, we have Proposition 3.7, which bounds the monoidal width of a matrix with its rank. In order to prove this result, we first need to bound the monoidal width of a matrix with its domain and codomain, which is done in Proposition 3.5.

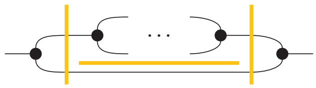
Proposition 3.5. *Let* P *be a cartesian and cocartesian prop. Suppose that* $—_1$, -●<$_1$, >●-$_1$, -●$_1$, ●-$_1 \in \mathscr{A}$ *and* $\mathsf{w}(—_1) \leq 1$, $\mathsf{w}($-●<$_1) \leq 2$, $\mathsf{w}($>●-$_1) \leq 2$, $\mathsf{w}($-●$_1) \leq 1$ *and* $\mathsf{w}($●-$_1) \leq 1$*. Suppose that, for every* $g \colon 1 \to 1$, $\mathsf{mwd}(g) \leq 2$*. Let* $f \colon n \to m$ *be a morphism in* P*. Then* $\mathsf{mwd}(f) \leq \min\{m,n\} + 1$.

Proof sketch. The proof proceeds by induction on $\max\{m,n\}$. The base cases are easily checked. The inductive step relies on the fact that, applying Lemma 2.9, if $n < m$, we can decompose f as shown below by cutting at most $n+1$ wires or, if $m < n$, in the symmetric way by cutting at most $m+1$ wires.



□

We can apply the former result to Bialg and obtain Proposition 3.7 because the width of 1×1 matrices, which are numbers, is at most 2. This follows from the reasoning in Example 2.6 as we can write every natural number $k \colon 1 \to 1$ as the following composition:



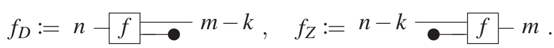
Lemma 3.6. *Let* $k\colon 1 \to 1$ *in* Bialg. *Then,* $\mathsf{mwd}(k) \leq 2$.

Proposition 3.7. *Let* $f\colon n \to m$ *in* Bialg. *Then,* $\mathsf{mwd} f \leq \mathsf{rank}(\mathsf{Mat} f) + 1$. *Moreover, if* f *is not* $\otimes$*-decomposable, i.e. there are no* f_1, f_2 *both distinct from* f *s.t.* $f = f_1 \otimes f_2$*, then* $\mathsf{rank}(\mathsf{Mat} f) \leq \mathsf{mwd} f$.

Proof sketch. This result follows from Lemma 3.1 and Proposition 3.5, which we can apply thanks to Lemma 3.6. □

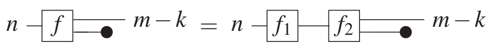
The bounds given by Proposition 3.7 can be improven when we have a $\otimes$-decomposition of a matrix, i.e. we can write $f = f_1 \otimes \ldots \otimes f_k$, to obtain Proposition 3.9. The latter relies on Lemma 3.8, which shows that discarding inputs or outputs cannot increase the monoidal width of a morphism in Bialg.

Lemma 3.8. *Let* $f\colon n \to m$ *in* Bialg *and* $d \in D_f$. *Let* $f_D := f\,;(\text{—}_{m-k} \otimes \text{-}\bullet_k)$ *and* $f_Z := (\text{—}_{n-k'} \otimes \bullet\text{-}_{k'})\,; f$*, where* $\text{-}\bullet_k\colon k \to 0$ *is the discard morphism with* $k \leq m$ *and* $\bullet\text{-}_{k'}\colon 0 \to k$ *is the zero morphism with* $k' \leq n$.



Then there are $\mathscr{D}(d) \in D_{f_D}$ *and* $\mathscr{Z}(d) \in D_{f_Z}$ *such that* $\mathsf{wd}(\mathscr{D}(d)) \leq \mathsf{wd}(d)$ *and* $\mathsf{wd}(\mathscr{Z}(d)) \leq \mathsf{wd}(d)$.

Proof sketch. By induction. The base cases are easy. If $f = f_1\,; f_2$, use the inductive hypothesis on f_2.



The $f = f_1 \otimes f_2$ case is similar. □

Proposition 3.9. *Let* $f\colon n \to m$ *in* Bialg *and* $d' = (d'_1, ;_k, d'_2) \in D_f$. *Suppose there are* f_1 *and* f_2 *such that* $f = f_1 \otimes f_2$. *Then, there is* $d = (d_1, \otimes, d_2) \in D_f$ *such that* $\mathsf{wd}(d) \leq \mathsf{wd}(d')$.

Proof sketch. By Lemma 3.1, $\mathsf{rank}(\mathsf{Mat} f_1) + \mathsf{rank}(\mathsf{Mat} f_2) = \mathsf{rank}(\mathsf{Mat}(f_1 \otimes f_2)) \leq k$ and, by Proposition 3.7, there is a monoidal decomposition d_i of f_i such that $\mathsf{wd}(d_i) \leq \mathsf{rank}(\mathsf{Mat} f_i) + 1$. Then, $\mathsf{wd}(d) := \mathsf{wd}((d_1, \otimes, d_2)) \leq \max\{\mathsf{rank}(\mathsf{Mat} f_1), \mathsf{rank}(\mathsf{Mat} f_2)\} + 1 \leq \mathsf{rank}(\mathsf{Mat} f_1) + \mathsf{rank}(\mathsf{Mat} f_2)$ whenever $\mathsf{rank}(\mathsf{Mat} f_1), \mathsf{rank}(\mathsf{Mat} f_2) > 0$. We apply Lemma 3.8 to obtain the same result if $\mathsf{rank}(\mathsf{Mat} f_1) = 0$ or $\mathsf{rank}(\mathsf{Mat} f_2) = 0$. □

We summarise Proposition 3.9 and Proposition 3.7 in Corollary 3.10.

Corollary 3.10. *Let* $f = f_1 \otimes \ldots \otimes f_k$ *in* Bialg. *Then,* $\mathsf{mwd}(f) \leq \max_{i=1,\ldots,k} \mathsf{rank}(\mathsf{Mat}(f_i)) + 1$. *Moreover, if* f_i *are not* $\otimes$*-decomposable, then* $\max_{i=1,\ldots,k} \mathsf{rank}(\mathsf{Mat}(f_i)) \leq \mathsf{mwd} f$.

Proof. By Proposition 3.9 there is a decomposition of f of the form $d = (d_1, \otimes, \cdots (d_{k-1}, \otimes, d_k))$, where we can choose d_i to be a minimal decomposition of f_i. Then, $\mathsf{mwd}(f) \leq \mathsf{wd}(d) = \max_{i=1,\ldots,k} \mathsf{wd}(d_i)$. By Proposition 3.7, $\mathsf{wd}(d_i) \leq r_i + 1$. Then, $\mathsf{mwd}(f) \leq \max\{r_1, \ldots, r_k\} + 1$. Moreover, if f_i are not $\otimes$-decomposable, Proposition 3.7 gives also a lower bound on their monoidal width: $\mathsf{rank}(\mathsf{Mat}(f_i)) \leq \mathsf{mwd} f_i$; and we obtain that $\max_{i=1,\ldots,k} \mathsf{rank}(\mathsf{Mat}(f_i)) \leq \mathsf{mwd} f$. □

The results so far show a way to construct efficient decompositions given a $\otimes$-decomposition of the matrix. However, we do not know whether $\otimes$-decompositions are unique. Proposition 3.11 shows that every morphism in Bialg has a unique $\otimes$-decomposition.

Proposition 3.11. *Let* C *be a monoidal category whose monoidal unit* 0 *is both initial and terminal, and whose objects are a unique factorization monoid. Let* f *be a morphism in* C*. Then* f *has a unique* $\otimes$*-decomposition.*

Our main result in this section follows from Corollary 3.10 and Proposition 3.11, which can be applied to Bialg because 0 is both terminal and initial, and the objects, being a free monoid, are a unique factorization monoid.

Theorem 3.12. *Let* $f = f_1 \otimes \ldots \otimes f_k$ *be a morphism in* Bialg *and its unique* $\otimes$*-decomposition given by Proposition 3.11, with* $r_i = \mathsf{rank}(\mathsf{Mat}(f_i))$*. Then* $\max\{r_1,\ldots,r_k\} \leq \mathsf{mwd}(f) \leq \max\{r_1,\ldots,r_k\}+1$.

Proof. This result is obtained by applying Corollary 3.10 to the $\otimes$-decomposition given by Proposition 3.11, which can be applied because, in Bialg, 0 is both terminal and initial, and the objects, being a free monoid, are a unique factorization monoid. □

Note that the identity matrix has monoidal width 1 and twice the identity matrix has monoidal width 2, attaining both the upper and lower bounds for the monoidal width of a matrix.

4 Graphs and rank width

Here we recall rank width [33] for undirected graphs.

Definition 4.1. An *undirected graph* $G = (V, E, \mathsf{ends})$ is given by a set of edges E, a set of vertices V and a function $\mathsf{ends}\colon E \to \wp_{\leq 2}(V)$ that gives the endpoints of each edge. We consider graphs *up to isomorphism*, or *abstract graphs*, thus the set of vertices can be fully characterised by its cardinality. An abstract graph can be equivalently given by an adjacency matrix $[G]$, where $G \in \mathsf{Mat}_{\mathbb{N}}(n,n)$ and n is the number of vertices. The equivalence class of adjacency matrices is defined by the equivalence relation

$$G \sim H \quad \text{iff} \quad G + G^{\top} = H + H^{\top}.$$

We will refer to abstract undirected graphs as simply graphs.

Definition 4.2. A *path* in a graph G is a sequence of edges $(e_1,\ldots,e_k)$ together with a sequence of distinct vertices $(v_1,\ldots,v_{k+1})$ of G such that, for every $i = 1,\ldots,k$, $\mathsf{ends}(e_i) = \{v_i, v_{i+1}\}$. A *tree* is a graph such that there is a unique path between any two of its vertices. Two vertices v and w in a graph G are *neighbours* if G has an edge between them. The *leaves* of a tree are those vertices with at most one neighbour. A *subcubic tree* is a tree where each vertex has between one and three neighbours.

A rank decomposition for a graph G is a tree whose leaves are labelled with the vertices of G.

Definition 4.3 ([33]). A *rank decomposition* (Y, r) of a graph G is given by a subcubic tree Y together with a bijection $r\colon \mathsf{leaves}(Y) \to \mathsf{vertices}(G)$.

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 beween 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.

Definition 4.4 (Order of an edge)**.** Let (Y, r) be a rank decomposition of a graph G. Let b be an edge of Y. The order of b is the rank of the matrix associated to it: $\mathsf{ord}(b) \coloneqq \mathsf{rank}(X_b)$.

Note that the order of the two sets in the partition does not matter as the rank is invariant to transposition. The width of a rank decomposition is the maximum order of the edges of the tree and the rank width of a graph is the width of its cheapest decomposition.

Definition 4.5 (Rank width)**.** Given a rank decomposition (Y,r) of a graph G, define its width as $\mathsf{wd}(Y,r) := \max_{b\in\mathsf{edges}(Y)} \mathsf{ord}(b)$. The *rank width* of G is given by the min-max formula:

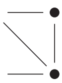
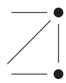
$$\mathsf{rwd}(G) := \min_{(Y,r)} \mathsf{wd}(Y,r).$$

4.1 Graphs with dangling edges

As intermediate step between rank decompositions and monoidal decompositions, we introduce recursive rank decompositions of *graphs with dangling edges* and we prove that they give a notion of width that is equivalent to rank width. Similar recursive characterisations were done for tree decompositions in [4] and for path and branch decompositions in [22]. We first need a notion of graph that is equipped with some "open" edges along which it can be glued with other graphs.

Definition 4.6. A *graph with dangling edges* $\Gamma = ([G], B)$ is given by an adjacency matrix $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ that records the connectivity of the graph and a matrix $B \in \mathsf{Mat}_{\mathbb{N}}(k,n)$ that records the "dangling edges" connected to n boundary ports. We will sometimes write $G \in \mathsf{adjacency}(\Gamma)$ and $B = \mathsf{boundary}(\Gamma)$.

Example 4.7. *Two graphs with the same ports, as illustrated below, can be "glued" together:*

 *glued with*  *gives* 

Decompositions are elements of a tree data type, with nodes carrying subgraphs Γ' of the ambient graph Γ. In the following Γ' ranges over the non-empty subgraphs of Γ: $T_\Gamma ::= (\Gamma') \mid (T_\Gamma, \Gamma', T_\Gamma)$. Given $T \in T_\Gamma$, the label function λ takes a decomposition and returns the graph with dangling edges at the root: $\lambda(T_1, \Gamma, T_2) := \Gamma$ and $\lambda(\Gamma) := \Gamma$.

Definition 4.8 (Recursive rank decomposition)**.** Let $\Gamma = ([G], B)$ be a graph with dangling edges, where $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ and $B \in \mathsf{Mat}_{\mathbb{N}}(k,n)$. A recursive rank decomposition of Γ is $T \in T_\Gamma$ where either: Γ has at most one vertex and $T = (\Gamma)$; or $T = (T_1, \Gamma, T_2)$ and $T_i \in T_{\Gamma_i}$ are recursive rank decompositions of subgraphs $\Gamma_i = ([G_i], B_i)$ of Γ such that:

- The vertices are partitioned in two, $[G] = \left[\begin{pmatrix} G_1 & C \\ \mathbb{0} & G_2 \end{pmatrix}\right]$;
- 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 $B = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix}$.

As with before, the *recursive* rank width of a graph is the width of its cheapest decomposition.

Definition 4.9. Let T be a recursive rank decomposition of $\Gamma = ([G], B)$. Define the width of T recursively: if $T = (\Gamma)$, $\mathsf{wd}(T) := \mathsf{rank}(B)$, and, if $T = (T_1, \Gamma, T_2)$, $\mathsf{wd}(T) := \max\{\mathsf{wd}(T_1), \mathsf{wd}(T_2), \mathsf{rank}(B)\}$ Expanding this expression, we obtain $\mathsf{wd}(T) = \max_{T' \text{ subtree of } T} \mathsf{rank}(\mathsf{boundary}(\lambda(T')))$. The *recursive rank width* of Γ is defined by the min-max formula $\mathsf{rrwd}(\Gamma) := \min_T \mathsf{wd}(T)$.

We show that recursive rank width is the same as rank width, up to the rank of the boundary of the graph.

Proposition 4.10. *Let $\Gamma = ([G], B)$ be a graph with dangling edges and (Y,r) be a rank decomposition of G. Then, there is a recursive rank decomposition $\mathscr{I}(Y,r)$ of Γ s.t. $\mathsf{wd}(\mathscr{I}(Y,r)) \leq \mathsf{wd}(Y,r) + \mathsf{rank}(B)$.*

Before proving the lower bound for recursive rank width, we need a technical lemma that relates the width of a graph with that of its subgraphs.

Lemma 4.11. *Let T be a recursive rank decomposition of $\Gamma = ([G], B)$. Let T' be a subtree of T and $\Gamma' := \lambda(T')$ with $\Gamma' = ([G'], B')$. The adjacency matrix of Γ can be written as $[G] = \left[\begin{pmatrix} G_L & C_L & C \\ \mathbb{0} & G' & C_R \\ \mathbb{0} & \mathbb{0} & G_R \end{pmatrix}\right]$ and its boundary as $B = \begin{pmatrix} A_L \\ A' \\ A_R \end{pmatrix}$. Then,* $\mathsf{rank}(B') = \mathsf{rank}(A' \mid C_L^\top \mid C_R)$.

Proposition 4.12. *Let T be a recursive rank decomposition of $\Gamma = ([G], B)$ with $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ and $B \in \mathsf{Mat}_{\mathbb{N}}(k,n)$. Then, there is a rank decomposition $\mathscr{I}^\dagger(T)$ of G such that* $\mathsf{wd}(\mathscr{I}^\dagger(T)) \leq \mathsf{wd}(T)$.

From Proposition 4.12 and Proposition 4.10 we conclude the following result.

Theorem 4.13. *Let $\Gamma = ([G], B)$. Then,* $\mathsf{rwd}(G) \leq \mathsf{rrwd}(\Gamma) \leq \mathsf{rwd}(G) + \mathsf{rank}(B)$.

5 Monoidal width and rank width

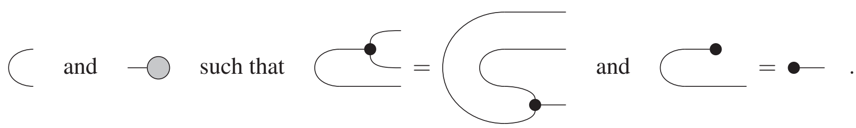
This section contains our main results. We prove that monoidal width in the prop of graphs Grph [14] corresponds to rank width, up to a constant multiplicative factor of 2.

We start by introducing the algebra of graphs with boundaries and its diagrammatic syntax [21]. A graph with boundaries is a graph together with two matrices L and R that record the connectivity of the vertices with the left and right boundary, a matrix P that records the passing wires from the left boundary to the right one and a matrix F that records the wires from the right boundary to itself.

Definition 5.1 ([21])**.** A *graph with boundaries* $g \colon n \to m$ is defined as $g = ([G], L, R, P, [F])$, where $[G]$ is the adjacency matrix of a graph on k vertices, with $G \in \mathsf{Mat}_{\mathbb{N}}(k,k)$; $L \in \mathsf{Mat}_{\mathbb{N}}(k,n)$, $R \in \mathsf{Mat}_{\mathbb{N}}(k,m)$, $P \in \mathsf{Mat}_{\mathbb{N}}(m,n)$ and $F \in \mathsf{Mat}_{\mathbb{N}}(m,m)$ recording connectivity information as explained above. Graphs with boundaries are taken up to an equivalence making the order of the vertices immaterial. Let $g, g' \colon n \to m$ on k vertices, with $g = ([G], L, R, P, [F])$ and $g' = ([G'], L', R', P, [F])$. The graphs g and g' are considered equal iff there is a permutation matrix $\sigma \in \mathsf{Mat}_{\mathbb{N}}(k,k)$ such that $g' = ([\sigma G \sigma^\top], \sigma L, \sigma R, P, [F])$.

Graphs with boundaries can be composed sequentially and in parallel [21], forming a symmetric monoidal category BGraph. The prop Grph provides a convenient syntax for graphs with boundaries. It is obtained by adding a cup and a vertex generators to the prop of matrices Bialg (Figure 4).

Definition 5.2 ([14])**.** The prop of graphs Grph is obtained by adding to Bialg the generators $\cup \colon 0 \to 2$ and $v \colon 1 \to 0$ with the equations below.

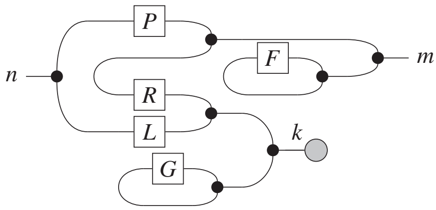


These equations mean, in particular, that the cup transposes matrices (Figure 5, left) and that we can express the equivalence relation of adjacency matrices: $G \sim H$ iff $G + G^\top = H + H^\top$ (Figure 5, right).

Proposition 5.3 ([21], Theorem 23)**.** *The prop of graphs* Grph *is isomorphic to the prop* BGraph.

Figure 5: Adding the cup.

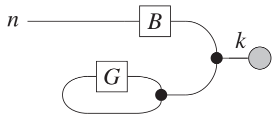
Proposition 5.3 means that the morphisms in Grph can be written in the following normal form



The prop Grph is more expressive than graphs with dangling edges (Definition 4.6): its morphisms can have edges between the boundaries as well. In fact, graphs with dangling edges can be seen as morphisms $n \to 0$ in Grph.

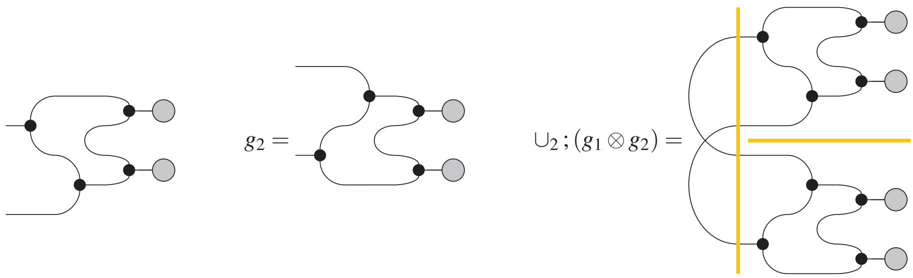
Example 5.4. *A graph with dangling edges* $\Gamma = ([G], B)$ *can be represented as a morphism in* Grph

$$g = ([G], B, ¡, !, [(\,)]) =$$



We can now formalise the intuition of glueing graphs with dangling edges as explained in Example 4.7. The two graphs there correspond to g_1 *and* g_2 *below left and middle. Their glueing is obtained by precomposing their monoidal product with a cup, i.e.* $\cup_2 ; (g_1 \otimes g_2)$*, as shown below right.*

$g_1 =$ $g_2 =$ $\cup_2 ; (g_1 \otimes g_2) =$



5.1 Rank width in open graphs

The technical content of our main result (Theorem 5.12) is split in two: an upper and a lower bound.

As in the prop of matrices Bialg, the cost of composing along n wires is n. All morphisms in Grph are chosen as atomic. One could restrict this to those with at most one vertex without affecting the results.

Definition 5.5. Let the set of *atomic morphisms* $\mathscr{A}$ be the set of all the morphisms of Grph. The *weight function* $\mathsf{w} \colon \mathscr{A} \cup \{\otimes\} \cup \mathsf{Obj}(\mathsf{Grph}) \to \mathbb{N}$ is defined, on objects n, as $\mathsf{w}(n) := n$; and, on morphisms $g \in \mathscr{A}$, as $\mathsf{w}(g) := k$, where k is the number of vertices of g.

Note that the monoidal width of g is bounded by the number of its vertices.

The upper bound (Proposition 5.8) is established by associating to each recursive rank decomposition a suitable monoidal decomposition. This mapping is defined inductively, given the inductive nature of both these structures. Given a recursive rank decomposition of a graph Γ, we can construct a decomposition of its corresponding morphism g as shown by the first equality in Figure 6. However, this

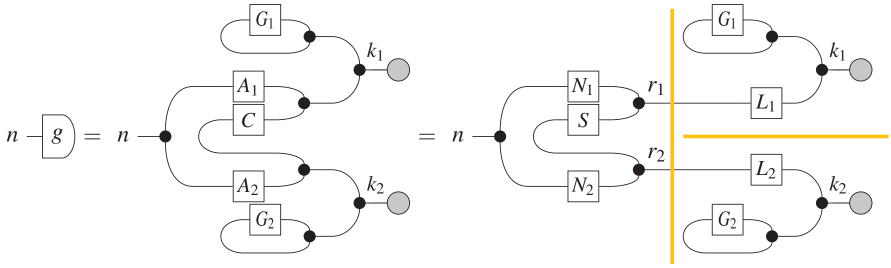
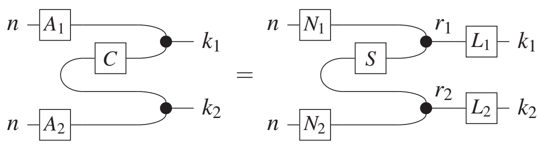


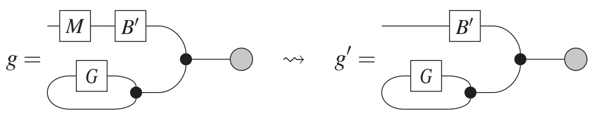
Figure 6: First step of a monoidal decomposition given by a recursive rank decomposition

decomposition is not optimal as it cuts along the number of vertices $k_1 + k_2$. But we can do better thanks to Lemma 5.6, which shows that we can cut along the ranks, $r_1 = \mathsf{rank}(A_1 \mid C)$ and $r_2 = \mathsf{rank}(A_2 \mid C^\top)$, of the boundaries of the induced subgraphs to obtain the second equality in Figure 6.



Lemma 5.6. *Let $A_i \in \mathsf{Mat}_{\mathbb{N}}(k_i, n)$, for $i = 1,2$, and $C \in \mathsf{Mat}_{\mathbb{N}}(k_1, k_2)$. 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 $(A_2 \mid C^\top) = L_2 \cdot (N_2 \mid S^\top \cdot L_1^\top)$.*

Once we have performed the cuts in Figure 6 on the right, we have changed the boundaries of the induced subgraphs. This means that we cannot apply the inductive hypothesis right away, but we need to transform first the recursive rank decompositions of the old subgraphs into decompositions of the new ones, as shown in Lemma 5.7. More explicitly, when M has full rank, if we have a recursive 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 5.7. *Let T be a recursive rank decomposition of $\Gamma = ([G], B)$ and $B = B' \cdot M$, with M that has full rank. Then, there is a recursive rank decomposition T' of $\Gamma' = ([G], B')$ such that $\mathsf{wd}(T) = \mathsf{wd}(T')$ and such that T and T' have the same underlying tree structure.*

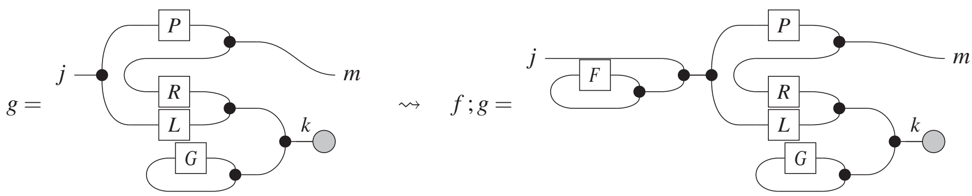
With the above ingredients, we can show that rank width bounds monoidal width from above.

Proposition 5.8. *Let* $\Gamma = ([G], B)$ *be a graph with dangling edges and* $g\colon n \to 0$ *be the morphism in* Grph *corresponding to* Γ. *Let* T *be a recursive rank decomposition of* Γ. *Then, there is a monoidal decomposition* $\mathscr{R}^\dagger(T)$ *of* g *such that* $\mathsf{wd}(\mathscr{R}^\dagger(T)) \leq 2\cdot\mathsf{wd}(T)$.

Proof sketch. The proof proceeds by induction on T. The base cases are easily checked and the inductive step relies on the decomposition of g in Figure 6, which we can write thanks to Lemma 5.6. Applying the inductive hypothesis and Lemma 5.7, the width of this decomposition can be bounded by $\max\{r_1 + r_2, 2\cdot\mathsf{wd}(T_1), 2\cdot\mathsf{wd}(T_2)\} \leq 2\cdot\mathsf{wd}(T)$, where $T = (T_1, \Gamma, T_2)$. □

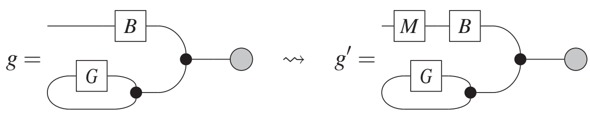
Proving the lower bound is similarly involved and follows a similar proof structure. From a monoidal decomposition we construct inductively a recursive rank decomposition of bounded width. The inductive step relative to composition nodes is the most involved and needs two additional lemmas, which allow us to transform recursive rank decompositions of the induced subgraphs into ones of two subgraphs that satisfy the conditions of Definition 4.8.

Applying the inductive hypothesis gives us a recursive rank decomposition of $\Gamma = ([G], (L \mid R))$, which is associated to g below left, and we need to construct one of $\Gamma' := \left(\left[G + L\cdot F\cdot L^\top\right], (L \mid R + L\cdot(F + F^\top)\cdot P^\top)\right)$, which is associated to $f\,;g$ below right, of at most the same width.



Lemma 5.9. *Let* T *be a recursive rank decomposition of* $\Gamma = ([G], (L \mid R))$. *Let* $F \in \mathsf{Mat}_{\mathbb{N}}(j, j)$, $P \in \mathsf{Mat}_{\mathbb{N}}(m, j)$ *and define* $\Gamma' := \left(\left[G + L\cdot F\cdot L^\top\right], (L \mid R + L\cdot(F + F^\top)\cdot P^\top)\right)$. *Then, there is a recursive rank decomposition* T' *of* Γ' *of bounded width:* $\mathsf{wd}(T') \leq \mathsf{wd}(T)$.

In order to obtain the subgraphs of the desired shape we need to add some extra connections to the boundaries. We have a recursive rank decomposition of $\Gamma = ([G], B)$, which corresponds to g below left, and we need one of $\Gamma' = ([G], B\cdot M)$, which corresponds to g' below right, of at most the same width.



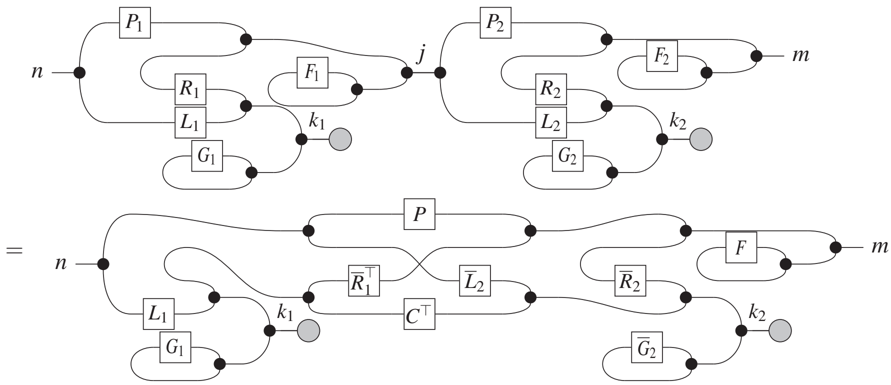
The following result and its proof are very similar to Lemma 5.7.

Lemma 5.10. *Let* T *be a recursive rank decomposition of* $\Gamma = ([G], B)$ *and let* $B' = B\cdot M$. *Then, there is a recursive rank decomposition* T' *of* $\Gamma' = ([G], B')$ *such that* $\mathsf{wd}(T') \leq \mathsf{wd}(T)$ *and such that* T *and* T' *have the same underlying tree structure. Moreover, if* M *has full rank, then* $\mathsf{wd}(T') = \mathsf{wd}(T)$.

Proposition 5.11. *Let* $g = ([G], L, R, P, [F])$ *in* Grph *and* $d \in D_g$. *Let* $\Gamma = ([G], (L \mid R))$. *Then, there is a recursive rank decomposition* $\mathscr{R}(d)$ *of* Γ *s.t.* $\mathsf{wd}(\mathscr{R}(d)) \leq 2\cdot\max\{\mathsf{wd}(d), \mathsf{rank}(L), \mathsf{rank}(R)\}$.

Proof sketch. The proof proceeds by induction on d. The base case is easily checked, while the inductive steps are a bit more involved. If $d = (d_1, ;_j, d_2)$, then there are $g_i = ([G_i], L_i, R_i, P_i, [F_i])$ such that $g =$

$g_1 ; g_2$ and we can write g as follows.



In order to build a recursive rank decomposition of Γ, we need recursive rank decompositions of $\overline{\Gamma}_i = (\left[\overline{G}_i\right], \overline{B}_i)$, but we can obtain recursive rank decompositions of $\Gamma_i = ([G_i], (L_i \mid R_i))$ by applying only induction. Thanks to Lemma 5.9, we obtain a recursive rank decomposition of $\Gamma_2' = (\left[G_2 + L_2 \cdot F_1 \cdot L_2^\top\right], (L_2 \mid R_2 + L_2 \cdot (F_1 + F_1^\top) \cdot P_2^\top))$. Lastly, we apply Lemma 5.10 to get recursive rank decompositions T_i of $\overline{\Gamma}_i$. Thanks to these, we can bound the width of $T := (T_1, \Gamma, T_2)$:

$$\mathsf{wd}(T) \leq 2 \cdot \max\{\mathsf{wd}(d_1), \mathsf{wd}(d_2), j, \mathsf{rank}(L), \mathsf{rank}(R)\} =: 2 \cdot \max\{\mathsf{wd}(d), \mathsf{rank}(L), \mathsf{rank}(R)\}.$$

If $d = (d_1, \otimes, d_2)$, we proceed similarly. □

From Proposition 5.8, Proposition 5.11 and Theorem 4.13, we obtain our main result.

Theorem 5.12. *Let G be a graph and let $g = ([G], ¡, ¡, (\,), [(\,)])$ be the corresponding morphism of* Grph. *Then,* $\frac{1}{2} \cdot \mathsf{rwd}(G) \leq \mathsf{mwd}(g) \leq 2 \cdot \mathsf{rwd}(G)$.

6 Conclusions and future work

We have shown that monoidal width, in a suitable category of graphs composable along "open" edges, yields rank width; a well-known measure from the graph theory literature.

Our goal with this line of research is to develop a generic, abstract "decomposition theory". We will study other graph widths like clique width [20] and twin width [12], as well as go beyond graphs: e.g. by focussing on tree width for hypergraphs and relational structures [1], branch width for matroids and widths for directed graphs. A part of "decomposition theory" means going beyond width as a mere number – in fact we believe that in each case the identification of a suitable monoidal category as an *algebra* of open graph structures is itself a worthwhile contribution. Indeed, having such an algebra means that a decomposition, rather than an ad hoc concept-specific construction, becomes more of a mathematical object in its own right. Such compositional algebras will add to the quiver of compositional structures of applied category theory; for example serving as syntax for more sophisticated applications [21].

Acknowledgements. Elena Di Lavore and Paweł Sobociński were supported by the European Union through the ESF funded Estonian IT Academy research measure (2014-2020.4.05.19-0001). This work was also supported by the Estonian Research Council grant PRG1210.

References

[1] 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)*, pages 1–12. IEEE, 2017. doi:10.1103/PhysRevLett.65.3373.

[2] D. Adolphson and T. C. Hu. Optimal linear ordering. *SIAM Journal on Applied Mathematics*, 25(3):403–423, 1973. doi:10.1137/0123021.

[3] Dorit Aharonov, Zeph Landau, and Johann Makowsky. The quantum fft can be classically simulated, 2006. arXiv:quant-ph/0611156.

[4] Stefan Arnborg, Bruno Courcelle, Andrzej Proskurowski, and Detlef Seese. An algebraic theory of graph reduction. *Journal of the ACM (JACM)*, 40(5):1134–1164, 1993. doi:10.1016/0095-8956(91)90061-N.

[5] Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical Systems Theory*, 20(1):83–127, 1987. doi:10.1007/BF01692060.

[6] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973. doi:10.1016/0097-3165(73)90016-2.

[7] Christoph Blume, HJ Sander Bruggink, Martin Friedrich, and Barbara König. Treewidth, pathwidth and cospan decompositions. *Electronic Communications of the EASST*, 41, 2011.

[8] Hans L Bodlaender. A tourist guide through treewidth. *Technical report*, 1992.

[9] Hans L Bodlaender and Arie MCA Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51(3):255–269, 2008. doi:10.1093/comjnl/bxm037.

[10] Guillaume Boisseau and Paweł Sobociński. String diagrammatic electrical circuit theory. *CoRR*, abs/2106.07763, 2021. URL: https://arxiv.org/abs/2106.07763, arXiv:2106.07763.

[11] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A survey of compositional signal flow theory. In Michael Goedicke, Erich J. Neuhold, and Kai Rannenberg, editors, *Advancing Research in Information and Communication Technology - IFIP's Exciting First 60+ Years, Views from the Technical Committees and Working Groups*, volume 600 of *IFIP Advances in Information and Communication Technology*, pages 29–56. Springer, 2021. doi:10.1007/978-3-030-81701-5_2.

[12] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. Twin-width i: tractable fo model checking. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 601–612. IEEE, 2020. doi:10.1109/FOCS.2015.63.

[13] Benjamin Merlin Bumpus and Zoltan A Kocsis. Spined categories: generalizing tree-width beyond graphs. *arXiv preprint arXiv:2104.01841*, 2021.

[14] Apiwat Chantawibul and Paweł Sobociński. Towards compositional graph theory. *Electronic Notes in Theoretical Computer Science*, 319:121–136, 2015. doi:10.1016/j.entcs.2015.12.009.

[15] Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019. arXiv:1709.00322, doi:10.1017/S0960129518000488.

[16] Maria Chudnovsky and Paul Seymour. A well-quasi-order for tournaments. *Journal of Combinatorial Theory, Series B*, 101(1):47–53, 2011. doi:10.1016/j.jctb.2010.10.003.

[17] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes - A first course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017. doi:10.1017/9781316219317.

[18] Cole Comfort and Aleks Kissinger. A graphical calculus for lagrangian relations. *CoRR*, abs/2105.06244, 2021. URL: https://arxiv.org/abs/2105.06244, arXiv:2105.06244.

[19] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and computation*, 85(1):12–75, 1990. doi:10.1016/0890-5401(90)90043-H.

[20] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

[21] Elena Di Lavore, Jules Hedges, and Paweł Sobociński. Compositional modelling of network games. In *29th EACSL Annual Conference on Computer Science Logic (CSL 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[22] Elena Di Lavore and Paweł Sobociński. Monoidal Width: Unifying Tree Width, Path Width and Branch Width, 2022. arXiv:2202.07582.

[23] Ross Duncan, Aleks Kissinger, Simon Perdrix, and John van de Wetering. Graph-theoretic simplification of quantum circuits with the zx-calculus. *Quantum*, 4:279, 2020. doi:10.22331/q-2020-06-04-279.

[24] Tomás Feder and Moshe Y Vardi. The computational structure of monotone monadic snp and constraint satisfaction: A study through datalog and group theory. *SIAM Journal on Computing*, 28(1):57–104, 1998. doi:10.1016/0196-6774(86)90023-4.

[25] Brendan Fong and David I Spivak. Seven sketches in compositionality: An invitation to applied category theory, 2018. arXiv:1803.05316.

[26] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. doi:10.1016/j.aim.2020.107239.

[27] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481. ACM, 2018. doi:10.1145/3209108.3209165.

[28] Dan R. Ghica, Achim Jung, and Aliaume Lopez. Diagrammatic semantics for digital circuits. In Valentin Goranko and Mads Dam, editors, *26th EACSL Annual Conference on Computer Science Logic, CSL 2017, August 20-24, 2017, Stockholm, Sweden*, volume 82 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CSL.2017.24.

[29] Rudolf Halin. S-functions for graphs. *Journal of geometry*, 8(1-2):171–186, 1976. doi:10.1007/BF01917434.

[30] André Joyal and Ross Street. The geometry of tensor calculus, i. *Advances in mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

[31] Stephen Lack. Composing PROPs. *Theor. App. Categories*, 13(9):147–163, 2004.

[32] Saunders Mac Lane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71:40–106, 1965. doi:10.2307/1993828.

[33] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *Journal of Combinatorial Theory, Series B*, 96(4):514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

[34] R Piziak and PL Odell. Full rank factorization of matrices. *Mathematics magazine*, 72(3):193–201, 1999. doi:10.1080/0025570X.1999.11996730.

[35] Pavel Pudlák, Vojtěch Rödl, and Petr Savickỳ. Graph complexity. *Acta Informatica*, 25(5):515–535, 1988.

[36] Julian Rathke, Paweł Sobociński, and Owen Stephens. Compositional reachability in petri nets. In *International Workshop on Reachability Problems*, pages 230–243. Springer, 2014. doi:10.1007/978-3-319-07734-5_9.

[37] Neil Robertson and Paul D. Seymour. Graph minors. I. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983. doi:10.1016/0095-8956(83)90079-5.

[38] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

[39] Neil Robertson and Paul D. Seymour. Graph minors. X. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153–190, 1991. doi:10.1016/0095-8956(91)90061-N.

[40] Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. doi:10.1090/conm/134/1187296.

[41] Fabio Zanasi. *Interacting Hopf Algebras - The Theory of Linear Systems*. PhD thesis, École Normale Supérieure de Lyon, 2015.

Compositional Modelling of Network Games

Elena Di Lavore
Department of Software Science, Tallinn University of Technology, Estonia
elena.di@taltech.ee

Jules Hedges
Department of Computer and Information Sciences, University of Strathclyde, Glasgow, UK
jules.hedges@strath.ac.uk

Paweł Sobociński
Department of Software Science, Tallinn University of Technology, Estonia
pawel.sobocinski@taltech.ee

Abstract

The analysis of games played on graph-like structures is of increasing importance due to the prevalence of social networks, both virtual and physical, in our daily life. As well as being relevant in computer science, mathematical analysis and computer simulations of such distributed games are vital methodologies in economics, politics and epidemiology, amongst other fields. Our contribution is to give compositional semantics of a family of such games as a well-behaved mapping, a strict monoidal functor, from a category of open graphs (syntax) to a category of open games (semantics). As well as introducing the theoretical framework, we identify some applications of compositionality.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases game theory, category theory, network games, open games, open graphs, compositionality

Digital Object Identifier 10.4230/LIPIcs.CSL.2021.30

Funding *Elena Di Lavore* and *Paweł Sobociński* are supported by the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

1 Introduction

Compositionality concerns finding homomorphic mappings

$$\mathbf{Syntax} \to \mathbf{Semantics}. \tag{1}$$

This important concept originated in formal logic [20, 21], and is at the centre of formal semantics of programming languages [23]. In recent years, there have been several *2-dimensional* examples [6, 4, 1], where both **Syntax** and **Semantics** are symmetric monoidal categories. Usually **Syntax** is freely generated from a (monoidal) signature, possibly modulo equations. This opens up the possibility of recursive definitions and proofs by structural induction, familiar from our experience with ordinary, 1-dimensional syntax.

In this paper, we consider an instance of (1) that is—at first sight—quite different from the usual concerns of programming and logic: network games [5], also known as graphical games. Network games involve agents that play concurrently, and share information based on an underlying, ambient network topology. Indeed, the utility of each player typically depends on the structure of the network. An interesting application is social networks [10], but they also feature in economics [11], politics [22] and epidemiology [16], amongst other fields. (These games should not be confused with classes of dynamic games played on graphs, such as parity games and pursuit games, which are not within the scope of this paper.)

29th EACSL Annual Conference on Computer Science Logic (CSL 2021).
Editors: Christel Baier and Jean Goubault-Larrecq; Article No. 30; pp. 30:1–30:24
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In formal accounts of network games, graphs represent network topologies. Players are identified with graph vertices, and their utility is influenced only by their choices and those of their immediate neighbours. Network games are thus "games on graphs". An example is the *majority game*: players "win" when they agree with the majority of their neighbours.

In what way do such games fit into the conceptual framework of (1)? Our main contribution is the framing of certain network games as monoidal functors from a suitable category of *open graphs* **Grph** [7], our **Syntax**, to the category of open games **Game** [13], our **Semantics**. Given a network game $\mathcal{N}$ (e.g. the majority game), such games are functors

$$F_{\mathcal{N}} \colon \mathbf{Grph} \to \mathbf{Game} \tag{2}$$

that, for any *closed* graph $\Gamma \in \mathbf{Grph}$, yield the game $F_{\mathcal{N}}(\Gamma)$, which is the game $\mathcal{N}$ played on Γ. However, compositionality means that such games are actually "glued together" from simpler, *open* games. In fact, $F_{\mathcal{N}}$ maps each vertex of Γ to an open game called the *utility-maximising player*, and the connectivity of Γ is mapped, following the rules of $\mathcal{N}$, to structure in **Game**.

Our contribution thus makes the intuitively obvious idea that the data of network games is dependent on their network topology precise. Concrete descriptions of network games, given a fixed topology, are often quite involved: our approach means that they can be derived in a principled way from basic building blocks. In some cases, the compositional description can also help in the mathematical analysis of games. For example, in the case of the majority game, the right decomposition of a network topology Γ as an expression in **Grph** can yield a recipe for the Nash equilibrium of $F_{\mathcal{N}}(\Gamma)$ in **Game** in terms of the equilibria of the open games obtained via $F_{\mathcal{N}}$ from the open graphs in the decomposition. As it happens when solving optimization problems, a compositional analysis of the equilibria is possible only when the game has optimal substructure, which is the case for the majority game (but is not the case in general). Nevertheless, compositional modelling is valuable for the understanding of the structure of the system. It allows, for example, to modify a part of a system while keeping the analysis done for the rest of the system, as we show in Example 31.

Technically, we proceed as follows. We introduce *monoid network games* (Definition 7) that make common structure of all of our motivating examples explicit, and that we believe cover the majority of network games studied in the literature. Roughly speaking, monoid network games are parametrised wrt *(i)* a monoid that aggregates information from neighbours and *(ii)* functions that govern how that information is propagated in the network. While we are able to model all network games, the structure of monoid network games allows us to characterise them as functors in a generic fashion.

Our category of open graphs **Grph** (Definition 18) is an extension of the approach of [7], from undirected graphs to undirected *multi*graphs. Multigraphs allow us to model games on networks where some links are stronger than others, cf. Example 30. Our **Grph** is different from other notions of "open graph" in the literature, e.g. via cospans [9], in that it is centred on the use of *adjacency matrices*, which are commonly used in graph theory to encode connectivity. Adjacency matrices give an explicit presentation of the graphs that allows an explicit description of the games played on them. Moreover, the emphasis on the matrix algebra means that **Grph** has the structure of commutative bialgebra—equivalent to the algebra of ordinary $\mathbb{N}$ matrices [15, 24]—but also additional structure that captures the algebraic content of adjacency matrices. Given that **Grph** has a presentation in terms of generators and equations, to obtain (2) it suffices to define it on the generators and check that **Grph**-equations are respected in **Game**. This is our main result, Theorem 27.

In addition to the presentation of **Grph** in terms of generators and equations, we characterise it as another category (Theorem 23) that makes clear its status as a category of "open graphs". The result can be understood as a kind of normal form for the morphisms of **Grph**, useful to describe concrete instantiations of $F_{\mathcal{N}}$ for arbitrary open graphs (Theorem 29).

Our work is a first step towards a more principled way of defining games parametrised by graphs. We would like to remark that the methodology that we present to define games on networks is more general than the particular instance worked out in this paper. Indeed, future work will extend both the notions of graphs (e.g. by considering directed graphs), as well as the kinds of games played on them (e.g. stochastic games, repeated games). While we do identify some applications, we believe that compositional reasoning is severely under-rated in traditional game theory, and that its adoption will lead to both more flexible modelling frameworks, as well as more scalable mathematical analyses.

Structure of the paper

We introduce our running examples in §2 and unify them under the umbrella of monoid network games. Next, we recall the basics of open games in §3 and identify the building blocks needed for (2). In §4 we introduce the category **Grph** of open, undirected multigraphs, and give a combinatorial characterisation, which is useful in applications. The construction of $F_{\mathcal{N}}$ is in §5, and several applications of our compositional framework are given in §6.

2 Network games

In this section we introduce motivating examples for our compositional framework and introduce a notion of game called the *monoid network game* that unifies them.

Network games [5, 14] are parametric wrt a network topology, usually represented by a graph. Players are the vertices, and the possible connections between the players are represented by the edges. Moreover, each player's payoff is affected only by the choices of its immediate *neighbours* on the graph. We use undirected multigraphs to model network topologies.

▶ **Definition 1.** *An undirected multigraph is $G = (V_G, E_G)$, where V_G is the set of vertices and E_G is a sym. multi-relation on V_G: a function $E_G \colon V_G \times V_G \to \mathbb{N}$ st $E_G(v_i, v_j) = E_G(v_j, v_i)$.*

A common way of capturing the connectivity of a graph is via *adjacency matrices*, which play an important role in graph theory. They are also crucial for our compositional account.

Assuming an ordering on the set of vertices of a graph, square matrices A with entries from $\mathbb{N}$ can record connections between vertex i and j in A_{ij}: a 0-entry signifies no edge, and non-zero entries count the connections. Ordinary matrices are too concrete to uniquely represent connectivity since edges between i and j can be recorded in the (i, j)th entry or the (j, i)th entry. One could use symmetric matrices or triangular matrices. For us, it is better to equate matrices that encode the same connectivity: $A \sim A'$ iff $A + A^T = A' + A'^T$.

▶ **Definition 2.** *An* adjacency matrix *is an equivalence class $[A]$ of matrices with entries in the natural numbers. The equivalence relation is given by*

$$A \sim A' \iff A + A^T = A' + A'^T.$$

A finite multigraph can also be defined as $(k_G, [A])$ where $k_G \in \mathbb{N}$ and $[A]$ a $k_G \times k_G$ adjacency matrix. Let $\mathbf{G}(n)$ be the set of multigraphs with n vertices, enumerated as $v_1, \ldots, v_n$.

▶ **Definition 3** (Network game). *An n-player* network game $\mathcal{N}$ *consists of, for each player $1 \leq i \leq n$, a set of choices X_i and a payoff $u_i : \mathbf{G}(n) \times \prod_{j=1}^{n} X_j \to \mathbb{R}$, such that each player's payoff is affected only by its own and its neighbours' choices: for each $G \in \mathbf{G}(n)$, each player i, each $j \neq i$ such that $(v_i, v_j) \notin E_G$, each $\underline{x}_{-j} \in \prod_{k \neq j}^{n} X_k$, and each $x_j, x'_j \in X_j$*

$$u_i(G, x_j, \underline{x}_{-j}) = u_i(G, x'_j, \underline{x}_{-j})$$

(The notation x_{-j}, standard in game theory, means a tuple with the jth element missing.)

The set of strategies *is $\prod_{i=1}^{n} X_i$ and its elements $\underline{x} \in \prod_{i=1}^{n} X_i$ are* strategy profiles*.*

The best response*, for a graph $G \in \mathbf{G}(n)$, is a relation $\mathbb{B}_{\mathcal{N}}$ on the set of strategies, defined by*

$$(\underline{x}, \underline{x}') \in \mathbb{B}_{\mathcal{N}} \Leftrightarrow \forall 1 \leq i \leq n.\ \forall y_i \in X_i.\ u_i(G, \underline{x}[i \mapsto x'_i]) \geq u_i(G, \underline{x}[i \mapsto y_i])$$

A Nash equilibrium*, for $G \in \mathbf{G}(n)$, is a strategy profile $\underline{x}$ s.t. for each player $1 \leq i \leq n$, $u_i(G, \underline{x}) \geq u_i(G, \underline{x}[i \mapsto x'_i])$ for each $x'_i \in X_i$. It is a fix-point of the best response relation.*

We now recall three important examples of network games.

▶ **Example 4** (Majority game). Each player has two choices, $X_i = \{Y, N\}$. A player receives a utility of 1 if its choice is the majority choice of its neighbours, and 0 otherwise, i.e.

$$u_i(G, \underline{x}) = \begin{cases} 1 & \text{if } |\{v_j \mid (v_i, v_j) \in E_G \text{ and } x_i = x_j\}| \geq |\{v_j \mid (v_i, v_j) \in E_G \text{ and } x_i \neq x_j\}| \\ 0 & \text{otherwise.} \end{cases}$$

Nash equilibria are strategy profiles where players take the majority choice of their neighbours.

▶ **Example 5** (Best-shot public goods game). Each player has two choices, $X_i = \{Y, N\}$, interpreted as investing or not investing in a public good. The investor bears a cost $0 < c < 1$, and gives a utility of 1 to themselves and every neighbour. The players are already partially satisfied with the current situation and assign a utility of $1 - c + \epsilon$, with $0 < \epsilon < c$, to the situation where neither the player nor its neighbours invest. The utility functions thus are:

$$u_i(G, \underline{x}) = \begin{cases} 1 - c & \text{if } x_i = Y \\ 1 & \text{if } x_i = N \text{ and } x_j = Y \text{ for some } (v_i, v_j) \in E_G \\ 1 - c + \epsilon & \text{otherwise.} \end{cases}$$

The Nash equilibrium is when no player invests, an example of a 'tragedy of the commons'.

▶ **Example 6** (Weakest-link public goods game). Each player's choice is an investment, valued in $\mathbb{R}_+$. The cost to the player given by an increasing cost function $c : \mathbb{R}_+ \to \mathbb{R}_+$ where $c(0) = 0$, and utility is the *minimum* level of investment of the player and all neighbours:

$$u_i(G, \underline{x}) = \min_{j=i \text{ or } (v_i, v_j) \in E_G} x_j - c(x_i).$$

A necessary condition for Nash equilibrium is that no player invests more than its neighbours.

In Examples 4, 5 and 6 every player has the same set of choices, and the utility depends in a uniform way on neighbours' choices. We collect these, and other examples in the literature, under the umbrella of *monoid network games.* Most examples in the literature can be collected in two classes [5, ch. 5], namely games on networks with constrained continuous actions or with binary actions. Provided that weights are natural numbers, the latter can be expressed as monoid network games. To express the former as monoid network games, we need to additionally ask that the parameters appearing in the utility functions of the players be constant. However, we can still express games of this class with different parameters for different players by composing different monoid network games. This is shown in Example 32.

Network games that do not fall into this category can be nevertheless expressed in a compositional way as illustrated in Fig. 1. If a game can be described in the form of a a *monoid* network game, we can say more: such games are a monoidal functor from the category of syntax to the category of semantics. The details are in Section 5.

To the best of our knowledge, the following has not previously appeared in the literature.

▶ **Definition 7** (Monoid network game). *A* monoid network game *is* $\mathcal{N} = (X, M, f, g)$ *where:*

- X *is the set of choices for each player*
- $M = (M, \oplus, e)$ *is a commutative monoid*
- $f : X \to M$ *and* $g : X \times M \to \mathbb{R}$ *are functions such that each utility function has the form*

$$u_i(G, \underline{x}) = g\left(x_i, \bigoplus_{(v_i,v_j)\in E_G} f(x_j)\right).$$

Examples 4, 5, 6 are indeed examples of monoid network games:

- The majority game (Example 4) has the monoid $(\mathbb{N}, +, 0) \times (\mathbb{N}, +, 0)$, counting the Y and N 'votes'. Define $f : \{Y, N\} \to \mathbb{N}^2$ by $f(Y) = (1, 0)$ and $f(N) = (0, 1)$, and $g : \{Y, N\} \times \mathbb{N}^2 \to \mathbb{R}$ is:

$$g(x, (n_1, n_2)) = \begin{cases} 1 & \text{if } x = Y \text{ and } n_1 \geq n_2 \\ 1 & \text{if } x = N \text{ and } n_1 \leq n_2 \\ 0 & \text{otherwise.} \end{cases}$$

- The best-shot public goods game (Example 5) is a monoid network game with the monoid $\mathbf{Bool} = (\{Y, N\}, \vee, N)$, where $\vee$ is logical or, $f : \mathbf{Bool} \to \mathbf{Bool}$ is the identity, and $g : \mathbf{Bool} \times \mathbf{Bool} \to \mathbb{R}$:

$$g(x, y) = \begin{cases} 1 - c & \text{if } x = Y \\ 1 & \text{if } x = N \text{ and } y = Y \\ 1 - c + \epsilon & \text{if } x = N \text{ and } y = N \end{cases}$$

- The weakest-link public goods game (Example 6) has the monoid $\mathbb{R}_+^\infty = (\{\mathbb{R}_+ \cup \{\infty\}, \min, \infty)$, f the embedding $\mathbb{R}_+ \hookrightarrow \mathbb{R}_+^\infty$, and $g : \mathbb{R}_+ \times \mathbb{R}_+^\infty \to \mathbb{R}$ is $g(x, y) = \min(x, y) - c(x)$.

3 Open games

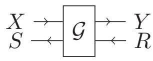
Open games were introduced in [13] as a compositional approach to game theory.

▶ **Definition 8** (Open game). *Let* X, Y, R, S, Σ *be sets. An open game* $\mathcal{G} \colon \left(\begin{smallmatrix} X \\ S \end{smallmatrix}\right) \xrightarrow{\Sigma} \left(\begin{smallmatrix} Y \\ R \end{smallmatrix}\right)$ *has:*

(i) $\mathbb{P}_\mathcal{G} \colon \Sigma \times X \to Y$*, called play function*
(ii) $\mathbb{C}_\mathcal{G} \colon \Sigma \times X \times R \to S$*, called coplay function*
(iii) $\mathbb{B}_\mathcal{G} \colon X \times (Y \to R) \to \mathcal{P}(\Sigma^2)$*, called best response function.*

Roughly speaking, an open game is a process that *(i)* given a *strategy* and *observation*, decides a *move*, and *(ii)* given a strategy, observation, and a *utility*, returns a *coutility* to the environment. Coutility is not a concept of classical game theory, but it enables compositionality by incorporating the fact that players reason about the future consequences of their actions. Finally, *(iii)*, the best response function, which, given a context for the game returns a relation on the set of strategies. A strategy σ is related to another strategy σ' if the latter is a best response to the former.

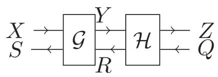
An open game is a process that receives observations (X) *"from the past"*, and the utility (R) *"from the future"*. It outputs moves (Y) covariantly and coutility (S) contravariantly.

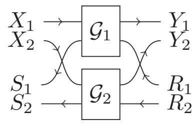


Open games are morphisms in a symmetric monoidal category **Game**. In order to formally define composition and monoidal product of games, it is useful to rephrase the definition in terms of lenses [19]. The detailed definitions are given in [13].

▶ **Definition 9 (Game).** **Game** *is the symmetric monoidal category with pairs of sets* $\binom{X}{S}$ *as objects and (equivalence classes of) open games* $\mathcal{G}\colon \binom{X}{S} \overset{\Sigma}{\nrightarrow} \binom{Y}{R}$ *as morphisms.*

We give some intuitions. Composition, shown below left, is sequential play: $\mathcal{H} \cdot \mathcal{G}$ is thought of as $\mathcal{H}$ happening after $\mathcal{G}$, observing the moves of $\mathcal{G}$ and feeding back its coutility as $\mathcal{G}$'s utility. The monoidal product of open games represents two games played independently. The games are placed side by side with no connections, as shown below right.





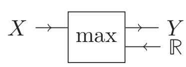
Classical games are *scalars* in **Game**, i.e. open games $\binom{1}{1} \nrightarrow \binom{1}{1}$. The fix-points of the best response functions of scalars in **Game** are the Nash equilibria of the games they represent.

Next we define specific open games used in our compositional account of network games. The first is the Utility Maximising Player, modelling typical players of classical game theory.

▶ **Definition 10** (Utility Maximising Player)**.** *Let X and Y be sets and* $\mathsf{argmax}\colon \mathbb{R}^Y \to \mathcal{P}(Y)$ *take a function $\kappa : Y \to \mathbb{R}$ to the subset of Y where κ is maximised. Define $\mathcal{D}$ to be:*

$$\mathcal{D}\colon \binom{X}{1} \overset{Y^X}{\nrightarrow} \binom{Y}{\mathbb{R}}$$

$$\begin{cases} \mathbb{P}_{\mathcal{D}}(f,x) = f(x) \\ \mathbb{C}_{\mathcal{D}}(f,x,r) = * \\ \mathbb{B}_{\mathcal{D}}(x,\kappa) = \{(y,y') \in Y^X \times Y^X : y'(x) \in \mathsf{argmax}(\kappa)\} \end{cases}$$

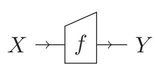


The category of sets and functions **Set** embeds into **Game** in two ways. In our account of network games, these embeddings encode how neighbours influence each other's utilities.

▶ **Definition 11.** *Let X, Y be sets and $f\colon X \to Y$ a function. Its covariant lifting is defined:*

$$f^*\colon \binom{X}{1} \overset{1}{\nrightarrow} \binom{Y}{1}$$

$$\begin{cases} \mathbb{P}_{f^*}(*,x) = f(x) \\ \mathbb{C}_{f^*}(*,x,*) = * \\ \mathbb{B}_{f^*}(x,*) = \{(*,*)\} \end{cases}$$



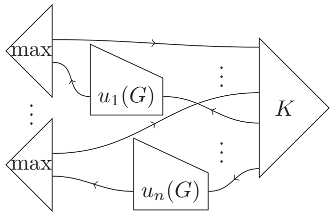
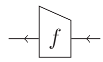


Figure 1 Open game representing a network game $\mathcal{N}$ played on a multigraph G

Similarly, its contravariant lifting is the following:

$$f_*\colon \ \left(\begin{smallmatrix}1\\Y\end{smallmatrix}\right) \xrightarrow{1} \left(\begin{smallmatrix}1\\X\end{smallmatrix}\right)$$

$$\begin{cases}\mathbb{P}_{f_*}(*,*) = * \\ \mathbb{C}_{f_*}(*,*,x) = f(x) \\ \mathbb{B}_{f_*}(*,x) = \{(*,*)\}\end{cases}$$



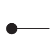
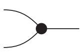
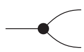
To obtain Examples 4, 5 and 6 as scalars in **Game**, players are taken to be utility-maximising players. The connectivity of the multigraph G determines their utility functions as contravariant liftings $u_i(G)$, while the context K sends back the choices of all players:

$$K\colon \ \left(\begin{smallmatrix}X^n\\X^n\times\cdots\times X^n\end{smallmatrix}\right) \xrightarrow{1} \left(\begin{smallmatrix}1\\1\end{smallmatrix}\right)$$
$$\mathbb{C}_K(\underline{x}) = (\underline{x},\ldots,\underline{x}).$$

The respective games are then obtained as the composition illustrated in Fig. 1. In this way, we obtain a compositional description of any network game. If a game can be described in the form of a a *monoid* network game, we can say more: such games are a monoidal functor from **Grph**, defined in the next section, to **Game**. The details are in Section 5.

4 Open graphs

We extend the compositional approach to graph theory of [7] from simple graphs to undirected multigraphs, identifying a "syntax" of network games as the arrows of a prop[1] **Grph**, generated from a monoidal signature and equations. We also provide a characterisation of **Grph** that explains its arrows as "open graphs". Differently from other approaches [3, 9], **Grph** uses adjacency matrices (Definition 2). Indeed, the presentation includes generators

 $: 0 \to 1,$  $: 2 \to 1,$  $: 1 \to 0,$  $: 1 \to 2$ (BIALG)

and the equations of Fig. 2. The prop **B** generated by this data is isomorphic [15, 24] to the prop of matrices with entries from $\mathbb{N}$, with composition being matrix multiplication. To convert between the two, think of the matrix as recording the numbers of paths: indeed, the (i,j)th entry in the matrix is the number of paths from the ith left port to the jth right port.

[1] A prop [17, 15] is a symmetric strict monoidal category where the objects are $\mathbb{N}$, and $m \otimes n := m + n$.

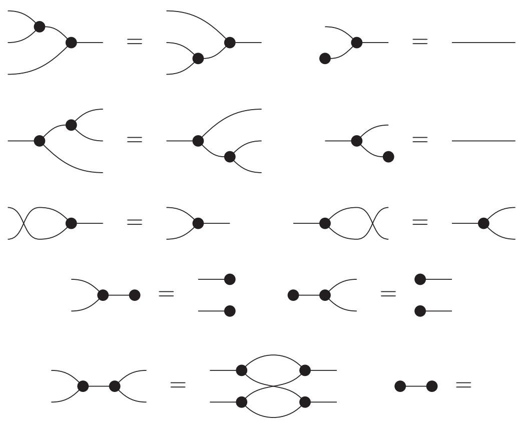


Figure 2 Commutative bialgebra equations, yielding prop **B**.

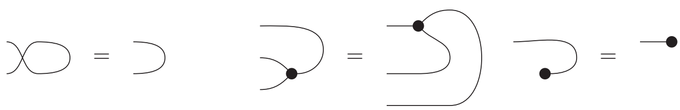
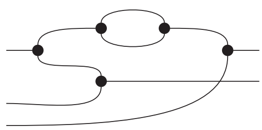


Figure 3 Equations of $\cup$, which together with the equations of Fig. 2 yield prop **BU**.

▶ **Example 12.** The following string diagram in **B** corresponds to the 3×2 matrix $\left(\begin{smallmatrix} 2 & 1 \\ 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$.



Next, we add a "cup" generator denoted

 $: 2 \to 0$ (U)

with its equations given in Fig. 3.

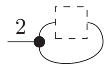
▶ **Definition 13.** *Let* **BU** *be the prop obtained from* (BIALG) *and* (U)*, quotiented by equations in Figs. 2 and 3, where the empty diagram is the identity on the monoidal unit.*

Just as **B** captures ordinary matrices, **BU** captures adjacency matrices:

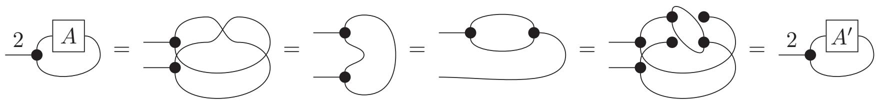
▶ **Proposition 14.** *For* $n \in \mathbb{N}$*, the hom-set* $[n, 0]$ *of* **BU** *is in bijection with* $n \times n$ *adjacency matrices, in the sense of Definition 2.*

We have seen that the relationship between matrices and diagrams in **B** is that the former encode the path information from the latter. Thus an $m \times n$ matrix is a diagram from m to n. Adding the cup and the additional equations means that, in general, a diagram from n to 0 in **BU** "encapsulates" an $n \times n$ matrix that expresses connectivity information in a similar way to adjacency matrices. We now give a concrete derivation to demonstrate this.

▶ **Example 15.** The equivalence relation of adjacency matrices is captured by the equations of Fig. 3. Consider matrices $A = \left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right) \sim \left(\begin{smallmatrix} 0 & 2 \\ 0 & 0 \end{smallmatrix}\right) = A'$. The morphism in **BU** is obtained by constructing their diagram in **B** as in Example 12 and "plugging" them in the following.

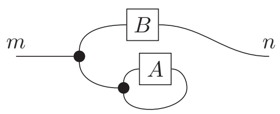


As shown below, the two diagrams obtained are equated by the axioms of **BU**.



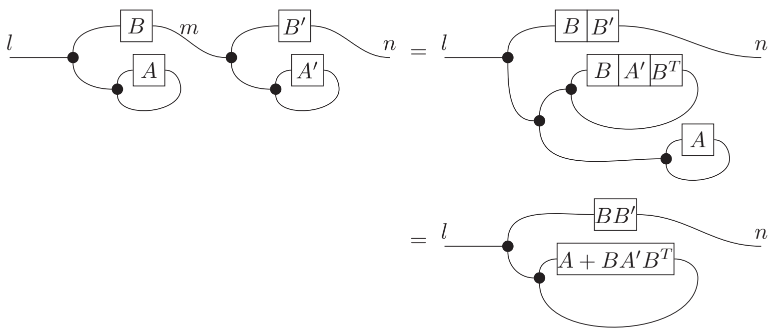
The prop **BU** can be given a straightforward combinatorial characterisation as the prop **Adj**.

▶ **Definition 16 (Adj).** *A morphism* $\alpha\colon m \to n$ *in the prop* **Adj** *[7] is a pair* $(B, [A])$*, where* $B \in \mathbf{Mat}_{\mathbb{N}}(m, n)$ *is a matrix, while* $[A]$*, with* $A \in \mathbf{Mat}_{\mathbb{N}}(m, m)$*, is an adjacency matrix. The components of* **Adj** *morphisms can be read off a "normal form" for* **BU** *arrows, as follows.*



Composition in **Adj** becomes intuitive when visualised with string diagrams.

$(B, [A]) \circ (B', [A']) = (BB', [A + BA'B^T])$



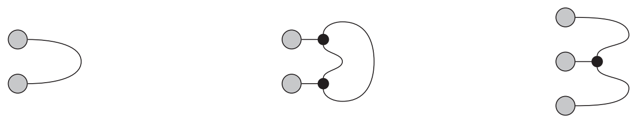
▶ **Proposition 17.** **BU** *is isomorphic to the prop* **Adj**. ◀

The proof is similar to the case for $\mathbb{Z}_2$ [7]. An extension of **BU** with just one additional generator and no additional equations yields the prop **Grph** of central interest for us.

▶ **Definition 18.** *The prop* **Grph** *is obtained from the generators in* (BIALG) *and* (U) *together with a generator*  $: 0 \to 1$. *The equations are those of Figs. 2 and 3.*

As we shall see, arrows $0 \to 0$ in **Grph** are precisely finite undirected multigraphs taken up to isomorphism: the additional generator plays the role of a graph vertex.

▶ **Example 19.** For example, the first of the following represents a multigraph with two vertices, connected by a single edge. The second one, two vertices connected by two edges. The third one, is a multigraph with three vertices and two edges between them.



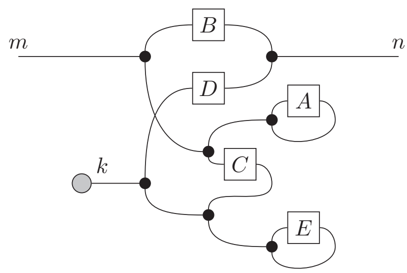
While the arrows $[0,0]$ are (iso classes of) multigraphs, general arrows can be understood as open graphs. Roughly speaking, they are graphs together with interfaces, and data that specifies the connectivity of the graph to its interfaces. We make this explicit below. Indeed, we shall see (Theorem 23) that the prop $\mathcal{A}$, defined below, is isomorphic to **Grph** – for this reason we use **Grph** string diagrams to illustrate its structure.

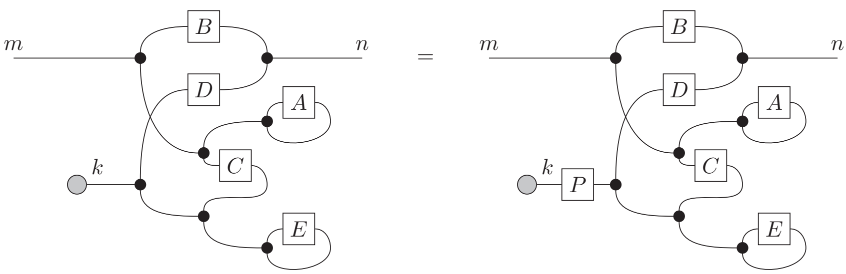
▶ **Definition 20** (The prop $\mathcal{A}$). *A morphism* $\Gamma\colon m \to n$ *in the prop* $\mathcal{A}$ *is defined by*

$$\Gamma = (k, [A], B, C, D, [E]) \tag{3}$$

where $k \in \mathbb{N}$, $A \in \mathbf{Mat}_{\mathbb{N}}(m,m)$, $B \in \mathbf{Mat}_{\mathbb{N}}(m,n)$, $C \in \mathbf{Mat}_{\mathbb{N}}(m,k)$, $D \in \mathbf{Mat}_{\mathbb{N}}(k,n)$ *and* $E \in \mathbf{Mat}_{\mathbb{N}}(k,k)$. *Similarly to* **Adj** *(Definition 16), the components of* (3) *can be read off a "normal form" for arrows of* **Grph**, *as visualised below right.*

Tuples (3) *are taken up to an equivalence relation that captures the fact that the order of the vertices is immaterial. Let* $\Gamma \sim \Gamma'$ *iff they are morphisms of the same type,* $\Gamma, \Gamma'\colon m \to n$ *with* k *vertices, and there is a permutation matrix* $P \in \mathbf{Mat}(k,k)$ *such that* $\Gamma' = (k, [A], B, CP^T, PD, [PEP^T])$. *The justification for this equivalence is the equality of the following two string diagrams in* **Grph**, *below (for the details, see Appendix A on page 18).*





It is worthwhile to give some intuition for the components of (3). The idea is that an arrow Γ specifies a multigraph $G = (k, [E])$, and:

- B specifies connections between the two boundaries, bypassing G
- C specifies connections between the left boundary and G
- D specifies connections between G and the right boundary
- A specifies connections between the interfaces on the left boundary. This allows Γ to introduce connections between the vertices of an "earlier" open graph Δ. See Example 21.

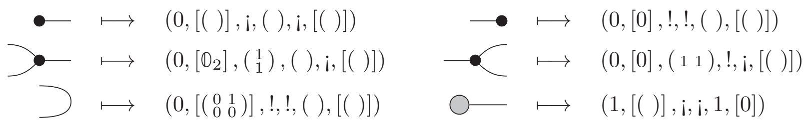
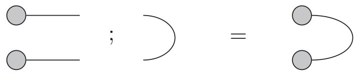


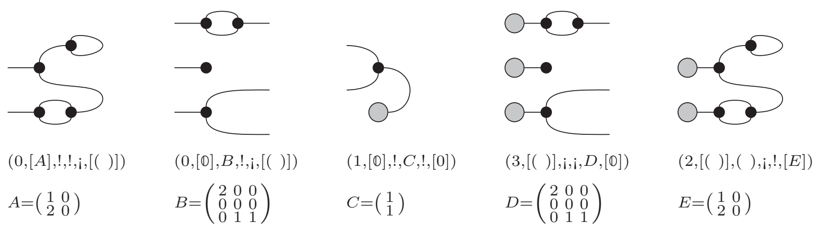
Figure 4 Image of θ on the generators

Defining composition in $\mathcal{A}$ is straightforward, given the above intuitions, but the details are rather tedious: see Lemma 33 in Appendix A.

▶ **Example 21.** In a composite $\Delta\ ;\Gamma$, Γ may introduce edges between the vertices of Δ. Indeed, the first diagram in Example 19 can be decomposed:



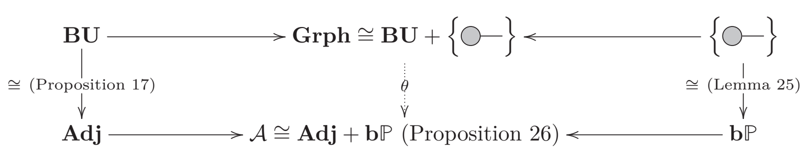
▶ **Example 22.** The following show the role of $\mathcal{A}$-morphism components, when isolated. The leftmost open graph has only left-side ports. It introduces a self-loop and two connections. The second has only connections between the left and right interfaces; the first left port is connected twice to the first right port, the second port is disconnected, and the third left port is connected to the second and third right ports. The third open graph has one vertex connected to the two left ports. The fourth has three vertices connected to the right ports, following the specification in the second. The rightmost (closed) multigraph has its vertices connected according to the specification of the leftmost vertex-less open graph. We write ! for matrices without columns, ¡ for matrices without rows and ( ) for the empty matrix.

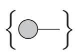
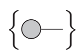
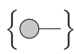


The main result in this section is the following.

▶ **Theorem 23.** *There is an isomorphism of props* $\theta\colon \mathbf{Grph} \to \mathcal{A}$.

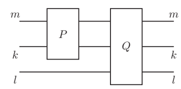
The remainder of this section builds a proof of the above, summarised in the diagram below.



First, note that **Grph** is the coproduct $\mathbf{BU}+$  in the category of props, where  is the free prop on a single generator $0 \to 1$. Next, we characterise  as $\mathbf{b}\mathbb{P}$, defined below, in Lemma 25. Given that $\mathbf{BU} \cong \mathbf{Adj}$, as shown in Proposition 17, to show the existence of θ it suffices to show that $\mathcal{A}$ satisfies the universal property of the coproduct $\mathbf{Adj} + \mathbf{b}\mathbb{P}$, which is Proposition 26. The action of θ on the generators of **Grph** is in Fig. 4.

▶ **Definition 24** ($\mathbf{b}\mathbb{P}$). *The prop of* bound permutations $\mathbf{b}\mathbb{P}$ *has as morphisms $m \to m+k$ pairs $[(k,P)]$ where $k \in \mathbb{N}$ and $P \in \mathbf{Mat}_{\mathbb{N}}(m+k, m+k)$ is a permutation matrix. Such pairs are identified to ensure that the order of the lower k rows of P is immaterial. Roughly speaking, considering P as a permutation of $m+k$ inputs to $m+k$ outputs, in $[(k,P)]$ the final k inputs are "bound". Explicitly, $(k,P) \sim (k,P')$ iff there is a permutation $\sigma \in \mathbf{Mat}_{\mathbb{N}}(k,k)$ st $P = \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P'$. Composition is defined:*

$$(l,Q) \circ (k,P) = (k+l, \left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_l \end{smallmatrix}\right) Q)$$



Identities are identity matrices $id_n = (0, \mathbb{1}_n)$. The fact that $\mathbf{b}\mathbb{P}$ is a prop is Lemma 34 in Appendix A.

▶ **Lemma 25.** $\mathbf{b}\mathbb{P}$ *is isomorphic to* $\{$○—$\}$.

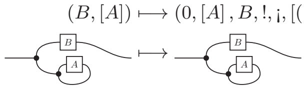
Proof. Let us call $\phi = (0, (1)) \colon 0 \to 1$, which is a morphism in $\mathbf{b}\mathbb{P}$. We show directly that, for any other prop $\mathbb{P}$ that contains a morphism $v \colon 0 \to 1$, there is a unique prop homomorphism $\alpha^{\#} \colon \mathbf{b}\mathbb{P} \to \mathbb{P}$ such that $\alpha^{\#}(\phi) = v$. The details are given as Lemma 35 in Appendix A. ◀

Given the results of Proposition 17 and Lemma 25, we obtain the isomorphism $\theta \colon \mathbf{Grph} \to \mathcal{A}$, thereby completing the proof of Theorem 23, by showing that:

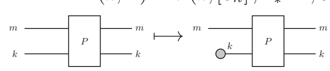
▶ **Proposition 26.** $\mathcal{A}$ *satisfies the universal property of the coproduct* $\mathbf{Adj} + \mathbf{b}\mathbb{P}$.

Proof. In order to show that $\mathcal{A}$ is a coproduct $\mathbf{Adj} + \mathbf{b}\mathbb{P}$, we define the two inclusions.

$$\begin{aligned} i_1 \colon \mathbf{Adj} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (B, [A]) &\longmapsto (0, [A], B, !, ¡, [(\ )]) \end{aligned}$$



$$\begin{aligned} i_2 \colon \mathbf{b}\mathbb{P} &\longrightarrow \mathcal{A} \\ n &\longmapsto n \\ (k, P) &\longmapsto (k, [\mathbb{0}_n], P_*^{[1,n]}, \mathbb{0}_{nk}, P_*^{[n+1,n+k]}, [\mathbb{0}_k]) \end{aligned}$$



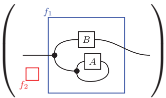
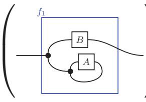
We indicate with $P_*^{[1,n]}$ the first n rows of the matrix P and, similarly, with $P_*^{[n+1,n+k]}$ the rows between the $n+1$-th and the $n+k$-th. It is not difficult to show that these are indeed homomorphism, the details are given as Claim 36 in Appendix A.

Now, we show that, for any other prop $\mathcal{C}$ with prop homomorphisms $\mathbf{Adj} \xrightarrow{f_1} \mathcal{C} \xleftarrow{f_2} \mathbf{b}\mathbb{P}$, there exists a unique prop homomorphism $H \colon \mathcal{A} \to \mathcal{C}$ such that $H \circ i_1 = f_1$ and $H \circ i_2 = f_2$.

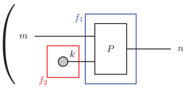
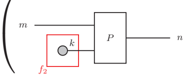
$$\begin{aligned} H \colon \mathcal{A} &\longrightarrow \mathcal{C} \\ n &\longmapsto n \\ (k, [A], B, C, D, [E]) &\longmapsto f_1\left(\left(\begin{smallmatrix} B \\ D \end{smallmatrix}\right), [\left(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix}\right)]\right) \circ (\mathbb{1}_m \otimes f_2(k, \mathbb{1}_k)) \end{aligned}$$

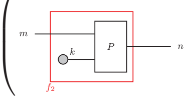
We verify that H is a homomorphism in Lemma 37 in Appendix A. Next, we confirm that $H \circ i_1 = f_1$ and $H \circ i_2 = f_2$, where two functor boxes [8] for f_1 and f_2 are coloured:

$$H \circ i_1(B, [A]) = H(0, [A], B, !, ¡, [(\,)])$$

$$= f_1(B, [A]) \circ (\mathbb{1}_m \otimes f_2(0, \mathbb{1}_0)) \Bigg($$  $$\Bigg) = f_1(B, [A]) \Bigg($$  $$\Bigg)$$

$$H \circ i_2(k, P) = H(k, [\mathbb{0}_n], P_*^{[1,n]}, \mathbb{0}_{nk}, P_*^{[n+1,n+k]}, [\mathbb{0}_k])$$

$$= f_1(P, [\mathbb{0}_{n+k}]) \circ (\mathbb{1}_n \otimes f_2(k, \mathbb{1}_k)) \Bigg($$  $$\Bigg) = P \circ f_2(k, \mathbb{1}_{n+k}) \Bigg($$  $$\Bigg)$$

$$= f_2(k, P) \Bigg($$  $$\Bigg)$$

Moreover, H is the unique prop homomorphism with these properties. In fact, suppose there is $H' \colon \mathcal{A} \to C$ such that $H' \circ i_1 = f_1$ and $H' \circ i_2 = f_2$. Then:

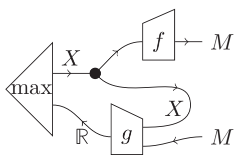
$$\begin{aligned} H'(k, [A], B, C, D, [E]) =& H'(i_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (\mathbb{1}_m \otimes i_2(k, \mathbb{1}_k))) \\ =& H' i_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (H'(\mathbb{1}_m) \otimes H' i_2(k, \mathbb{1}_k)) \\ =& f_1((\begin{smallmatrix} B \\ D \end{smallmatrix}), [(\begin{smallmatrix} A & C \\ \mathbb{0} & E \end{smallmatrix})]) \circ (\mathbb{1}_m \otimes f_2(k, \mathbb{1}_k)) = H(k, [A], B, C, D, [E]). \blacktriangleleft \end{aligned}$$

5 Games on graphs via functorial semantics

Here we show that monoid network games $\mathcal{N}$ define monoidal functors $F_{\mathcal{N}} \colon \mathbf{Grph} \to \mathbf{Game}$, which is our main contribution. To every open graph Γ, $F_{\mathcal{N}}$ associates an open game, where $\mathcal{N}$ is played on Γ. We give an explicit account of the $F_{\mathcal{N}}$-image of open graphs Γ, using Theorem 23. We also explain how $F_{\mathcal{N}}$ acts on closed graphs, giving classical games.

Since **Grph** is given by generators and equations, it suffices to define $F_{\mathcal{N}}$ on the generators and show that the equations are respected. Fix a monoid network game $\mathcal{N} = (X, M, f, g)$.

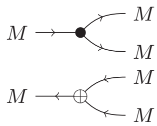
- On objects, $F_{\mathcal{N}}(1) = (\begin{smallmatrix} M \\ M \end{smallmatrix})$. Thus, for $n \in \mathbf{Grph}$, we have $F_{\mathcal{N}}(n) = (\begin{smallmatrix} M^n \\ M^n \end{smallmatrix})$
- The vertex ○—— $: 0 \to 1$ is mapped to the open game $F_{\mathcal{N}}($○——$)\colon\ (\begin{smallmatrix} 1 \\ 1 \end{smallmatrix}) \overset{X}{\nrightarrow} (\begin{smallmatrix} M \\ M \end{smallmatrix})$ defined
  - $\Sigma_{F_{\mathcal{N}}(\circ-)} = X$
  - $\mathbb{P}_{F_{\mathcal{N}}(\circ-)}(x_i, *) = f(x_i)$
  - $\mathbb{C}_{F_{\mathcal{N}}(\circ-)}(x_i, *, m) = *$
  - $(x_i, x_i') \in \mathbb{B}_{F_{\mathcal{N}}(\circ-)}(*, \kappa : M \to M)$ iff $x_i' \in \arg\max_{x_i'':X} g(x_i'', \kappa(f(x_i'')))$



- The generators (BIALG) are mapped to the bialgebra structure on (M, M) induced by the monoid action of M. Specifically, they are:

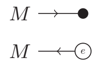
$F_{\mathcal{N}}(\text{—●<})\colon \binom{M}{M} \xrightarrow{1} \binom{M^2}{M^2}$

$$\begin{cases} \mathbb{P}(*, m) = (m, m) \\ \mathbb{C}(*, m_1, m_2, m_3) = m_2 \oplus m_3 \end{cases}$$



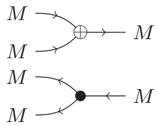
$F_{\mathcal{N}}(\text{—●})\colon \binom{M}{M} \xrightarrow{1} \binom{1}{1}$

$$\begin{cases} \mathbb{P}(*, m) = * \\ \mathbb{C}(*, m, *) = e \end{cases}$$



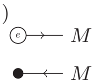
$F_{\mathcal{N}}(\text{>●—})\colon \binom{M^2}{M^2} \xrightarrow{1} \binom{M}{M}$

$$\begin{cases} \mathbb{P}(*, m_1, m_2) = m_1 \oplus m_2 \\ \mathbb{C}(*, m_1, m_2, m_3) = (m_1, m_1) \end{cases}$$



$F_{\mathcal{N}}(\text{●—})\colon \binom{1}{1} \xrightarrow{1} \binom{M}{M}$

$$\begin{cases} \mathbb{P}(*, *) = e \\ \mathbb{C}(*, *, m) = * \end{cases}$$

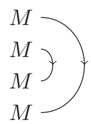


where each of these open games is built from lifted functions (Definition 11).

- ⊃ $: 2 \to 0$ is mapped to the following open game (see [13])

$F_{\mathcal{N}}(\supset)\colon \begin{pmatrix} M^2 \\ M^2 \end{pmatrix} \xrightarrow{1} \binom{1}{1}$

$$\begin{cases} \mathbb{P}(*, m_1, m_2) = * \\ \mathbb{C}(*, m_1, m_2, *) = (m_2, m_1) \end{cases}$$



To prove that $F_{\mathcal{N}}$ is a symmetric monoidal functor it suffices to show that the equations of **Grph** are respected; this is a straightforward but somewhat lengthy computation.

▶ **Theorem 27.** *$F_{\mathcal{N}}$ defines a symmetric monoidal functor* **Grph** $\to$ **Game**.

Proof. See Appendix B, on page 23. ◀

Note that $F_{\mathcal{N}}$ does *not* respect axioms (C1) or (C2) of [7], so it does not define a functor **ABUV** $\to$ **Game** in the terminology of *loc. cit.* This, together with the increased expressivity of multigraphs over simple graphs, motivates our extension from **ABUV** to **Grph**.

Theorem 23 gives a convenient "normal form" for the arrows of **Grph**, which we use to give an explicit description of the image of any (open) graph Γ under $F_{\mathcal{N}}$. First, we specialise to closed graphs that yield ordinary network games. This result—a sanity check for our compositional framework—is a corollary of the more general Theorem 29, proved subsequently.

▶ **Corollary 28.** *Let $\mathcal{N} = (X, M, f, g)$ be a monoid network game, and consider $\Gamma : 0 \to 0$ in* **Grph***, an undirected multigraph with k vertices. Then the game $F_{\mathcal{N}}(\Gamma)\colon \binom{1}{1} \xrightarrow{X^k} \binom{1}{1}$ has:*
- *$\Sigma_{F_{\mathcal{N}}(\Gamma)} = X^k$ as its strategy profiles,*
- *$\mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(*, *) \subseteq X^k \times X^k$ is the best response relation of $\mathcal{N}$ played on Γ.*

Note that while the expressions in the statement of Theorem 29 below may seem involved, they are actually derived in an entirely principled, compositional manner from the generators of **Grph**. Indeed, the proof is by structural induction on the morphisms of **Grph**.

▶ **Theorem 29.** *Let $\mathcal{N} = (X, M, f, g)$ be a monoid network game. Let $\Gamma : i \to j$ be a morphism in* **Grph** *with k vertices st $\theta(\Gamma) = (k, [A], B, C, D, [E])$, where $A : i \times i$, $B : i \times j$, $C : i \times k$, $D : k \times j$ and $E : k \times k$. Then the open game $F_{\mathcal{N}}(\Gamma)\colon \begin{pmatrix} M^i \\ M^i \end{pmatrix} \xrightarrow{X^k} \begin{pmatrix} M^j \\ M^j \end{pmatrix}$ has:*
- *set of strategy profiles $\Sigma(F_{\mathcal{N}}(\Gamma)) = X^k$*
- *play function $\mathbb{P}_{F_{\mathcal{N}}(\Gamma)} : X^k \times M^i \to M^j$ given by $\mathbb{P}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma}, \underline{x}) = B^T \underline{x} \oplus D^T f(\underline{\sigma})$*
- *coplay function $\mathbb{C}_{F_{\mathcal{N}}(\Gamma)} : X^k \times M^i \times M^j \to M^i$ is $\mathbb{C}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma}, \underline{x}, \underline{r}) = (A + A^T)\underline{x} \oplus B\underline{r} \oplus Cf(\underline{\sigma})$*

- *best response relation* $\mathbb{B}_{F_{\mathcal{N}}(\Gamma)} : M^i \times (M^j \to M^j) \to \mathcal{P}(X^k \times X^k)$ *is* $(\underline{\sigma}, \underline{\sigma}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(\underline{x}, \kappa)$ *iff, for all* k,

$$\sigma'_k \in \operatorname*{argmax}_{s \in X} g\left(s, (C^T)^k_* \underline{x} \oplus D^k_* \kappa \left(B^T \underline{x} \oplus D^T f(\underline{\sigma}\,[k \mapsto s])\right) \oplus (E + E^T)^k_* f(\underline{\sigma}\,[k \mapsto s])\right)$$

Proof. See Appendix B on page 23. ◀

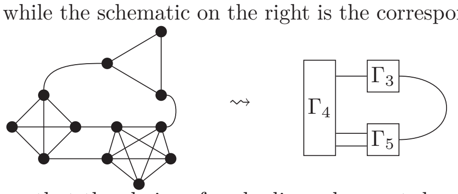
6 Examples

We return to examples: the majority (Example 4), the best-shot public goods (Example 5) and the weakest-link public goods (Example 6) games, and demonstrate various applications of our framework. We first show that to compute the Nash equilibrium of the majority game played on interconnected cliques is to calculate equilibria of its clique subgames.

▶ **Example 30** (Majority game). In the majority game the best response can be decomposed into the best responses of its components. Let $\mathcal{N}$ be the monoid network game for the majority game, defined on pg. 5, and consider a graph composed of N cliques, as follows:

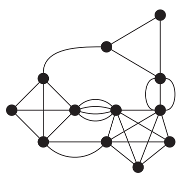
- each vertex of each clique can be connected to at most one vertex of another clique,
- in each clique there is at least one vertex not connected to any vertex outside its clique.

Such graphs decompose as N open graphs, each a clique with some boundary connections. We omit the details and give, instead, an illustrative example: below left is a picture of three connected cliques, while the schematic on the right is the corresponding expression in **Grph**.



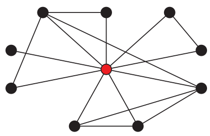
It is easy to show that the choice of each clique does not depend on the choices of other cliques. Indeed, the Nash equilibria of the majority game played on connected cliques in our sense are those strategy profiles where, in every clique, all players make the same choice. In particular, there are 2^N Nash equilibria.

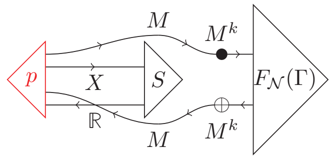
In some cases, players can take into account the choice of another player with a different intensity. This can be modelled by changing the number of edges between the vertices. Let us consider the above example with some of the vertices connected multiple times. This modification of the network—illustrated below—reflects in a modification of the equilibria, which are now strategy profiles in which every player takes the same choice.



In the best-shot public goods game (Example 5), the Nash equilibrium is when no player invests. In Example 31, we show how the compositional description is useful to adapt the model to a slightly different situation. We can imagine that one of the players now has access to incentives to invest in the public good. This scenario is represented by modifying the game and allowing one player to interact with the environment, which is the source of the incentives for this player. This modification "opens" the game to one of type $\left(\begin{smallmatrix}1\\1\end{smallmatrix}\right) \to \left(\begin{smallmatrix}X\\\mathbb{R}\end{smallmatrix}\right)$: as a result, the Nash equilibrium changes. This is a simple model of a common economic situation, 'solving' a social dilemma by external intervention, for example by regulation [12].

▶ **Example 31** (Best-shot public goods game). Consider the best-shot public goods game played on a graph that contains a vertex connected to all other vertices. Removing the central vertex from this graph leaves an open graph that we will call Γ.





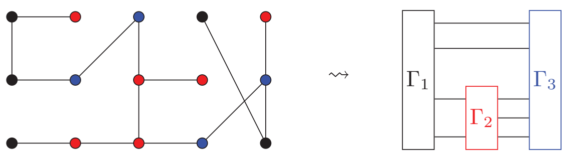
Here, $F_{\mathcal{N}}(\Gamma)$ is the best-shot public goods game played on the open graph Γ, p is the central player that has been substituted, and S is the external open game that influences p. The utility function of player p and the coplay function of S are as follows.

$$u_p(\Gamma, x) = \begin{cases} 1 - c + \delta & \text{if } x_p = 1 \\ 1 - \epsilon & \text{if } x_p = 0 \wedge \exists (p, j) \in E_\Gamma \; x_j = 1 \\ 1 - c & \text{if } \forall j \; x_j = 0 \end{cases} \qquad S\colon \binom{X}{\mathbb{R}} \overset{1}{\nrightarrow} \binom{1}{1} \qquad \mathbb{C}(*, x, *) = \begin{cases} \delta & \text{if } x = 1 \\ -\epsilon & \text{if } x = 0 \end{cases}$$

The addition of the open game S and the modification of player p modifies the Nash equilibrium to be the strategy profile where only the central player invests. The idea is that the "external" agent S incentivises the central player p to invest.

Our last example illustrates a common situation where the compositional description of a game does *not* allow a compositional analysis of the best response. However, in this case, compositionality can be used to obtain a variant of the weakest-link public goods game (Example 6) where different cost functions are used in different parts of the graph G. The desired game is obtained by composing such open games according to the structure of G.

▶ **Example 32** (Weakest-link public goods game). Consider the weakest-link public goods game played on a connected graph G. Suppose that players have different cost functions. We partition them according to their cost functions, and use this partition to decompose the G into an expression in **Grph**, as illustrated for a particular example below:



While the definition above uses our compositional techniques, the Nash equilibrium is calculated on the resulting closed game, and is a strategy profile where every player invests equally, with utility depending on individual cost functions. While it may be unsatisfying, this failure of Nash equilibria to be compositional can be seen as an inherent feature of game theory. In particular it is already present in the theory of open games; the passage from graphs to games is nevertheless fully compositional.

7 Conclusions

Our contribution is a compositional account of network games via strict monoidal functors. This adds a class of network games to the games that have been expressed in compositional game theory [13, 2]. Of independent interest is our work on the category **Grph**, extending [7]. This is an approach to "open graphs" that, as we have seen, is compatible with the structure of open games, and in future work we will identify other uses of this category.

We also intend to extend the class of open graphs to *directed* open graphs. The motivation for this is that, in some network games, interactions between players are *not* bidirectional. Consider, for example, a variant of the majority game where there is an "influencer": a player whose choice affects the choices of other players, but is not in turn conversely affected.

We will also extend the menagerie of games that can be played on a graph. We plan to study games with more generic utility functions, incomplete information, and repeated games. It could also prove interesting to study natural transformations between the functors that define games, and explore the game theoretical relevance of such transformations.

References

1. John C Baez and Brendan Fong. A compositional framework for passive linear networks, 2015. arXiv preprint: `https://arxiv.org/abs/1504.05625`.
2. Joe Bolt, Jules Hedges, and Philipp Zahn. Bayesian open games, 2019. arXiv preprint: `https://arxiv.org/abs/1910.03656`.
3. Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Pawel Sobocinski, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In Martin Grohe, Eric Koskinen, and Natarajan Shankar, editors, *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016*, pages 710–719. ACM, 2016. `doi:10.1145/2933575.2935316`.
4. Filippo Bonchi, Pawel Sobocinski, and Fabio Zanasi. The calculus of signal flow diagrams I: linear relations on streams. *Inf. Comput.*, 252:2–29, 2017. `doi:10.1016/j.ic.2016.03.002`.
5. Yann Bramoullé, Andrea Galeotti, and Brian Rogers. *The Oxford handbook of the economics of networks*. Oxford University Press, 2016.
6. Roberto Bruni, Hernán C. Melgratti, and Ugo Montanari. A connector algebra for P/T nets interactions. In *Concurrency Theory (CONCUR '11)*, volume 6901 of *LNCS*, pages 312–326. Springer, 2011. `doi:10.1007/978-3-642-23217-6_21`.
7. Apiwat Chantawibul and Paweł Sobociński. Towards compositional graph theory. In *Proceedings of MFPS'15*, volume 319 of *ENTCS*, pages 121–136, 2015. `doi:10.1016/j.entcs.2015.12.009`.
8. J. R. B. Cockett and R. A. G. Seely. Linearly distributive functors. *Journal of Pure and Applied Algebra*, 143(1-3):155–203, 1999.
9. Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.
10. James H Fowler and Nicholas A Christakis. Dynamic spread of happiness in a large social network: longitudinal analysis over 20 years in the framingham heart study. *BMJ*, 337, 2008. `doi:10.1136/bmj.a2338`.
11. Andrea Galeotti. Talking, searching and pricing. *International Economic Review*, 51(4):1159–1174, 2010. `doi:10.1111/j.1468-2354.2010.00614.x`.
12. Andrea Galeotti, Benjamin Golub, and Sanjeev Goyal. Targeting interventions in networks. Forthcoming in *Econometrica*, 2019.
13. Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of Logic in Computer Science (LiCS) 2018*. ACM, 2018. `doi:10.1145/3209108.3209165`.
14. Matthew Jackson and Yves Zenou. Games on networks. In *Handbook of game theory with economic applications*, volume 4, chapter 3, pages 95–163. Elsevier, 2015. `doi:10.1016/B978-0-444-53766-9.00003-3`.
15. Stephen Lack. Composing PROPs. *Theor. App. Categories*, 13(9):147–163, 2004.
16. Qiu Li, MingChu Li, Lin Lv, Cheng Guo, and Kun Lu. A new prediction model of infectious diseases with vaccination strategies based on evolutionary game theory. *Chaos, Solitons & Fractals*, 104:51–60, 2017. `doi:10.1016/j.chaos.2017.07.022`.
17. Saunders Mac Lane. Categorical algebra. *Bull. Amer. Math. Soc.*, 71:40–106, 1965.

18 Saunders Mac Lane. *Categories for the Working Mathematician.* Springer-Verlag New York, 1978.

19 Frank Joseph Oles. *A Category-theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, Syracuse, NY, USA, 1982. AAI8301650.

20 Alfred Tarski. The concept of truth in the languages of the deductive sciences. *Prace Towarzystwa Naukowego Warszawskiego, Wydzial III Nauk Matematyczno-Fizycznych*, 34(13-172):198, 1933.

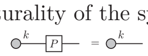
21 Alfred Tarski and Robert L Vaught. Arithmetical extensions of relational systems. *Compositio mathematica*, 13:81–102, 1957.

22 Giorgio Topa. Social Interactions, Local Spillovers and Unemployment. *The Review of Economic Studies*, 68(2):261–295, April 2001. doi:10.1111/1467-937X.00169.

23 Glynn Winskel. *The formal semantics of programming languages: an introduction.* MIT press, 1993.

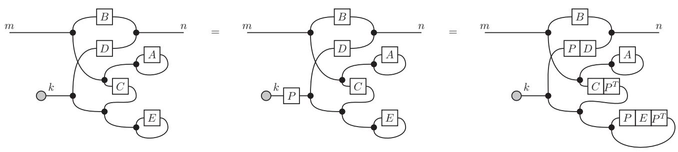
24 Fabio Zanasi. *Interacting Hopf Algebras: the theory of linear systems.* PhD thesis, École Normale Supérieure, 2015.

A Proofs for Section 4

Details for definition 20. By naturality of the symmetries, the vertex generators commute with any permutation matrix P: 

.

Thus, we can show that $\Gamma = (k, [A], B, C, D, [E])$ and $\Gamma' = (k, [A], B, CP^T, PD, [PEP^T])$ represent the same open graph.

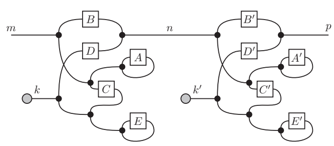


◀

▶ **Lemma 33.** *$\mathcal{A}$ is a prop.*

Proof. We start by proving that $\mathcal{A}$ is a category. The diagram below can be rewritten, using the axioms of **B**, as a diagram of the form shown in Definition 20. The components of the normal form obtained in this way give the algebraic definition of the composition.

$$\Gamma' \circ \Gamma = \left(k + k', \left[A + BA'B^T\right], BB', \left(C + B(A' + A'^T)D^T | BC'\right), \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix}\right), \left[\left(\begin{smallmatrix} E + DA'D^T & DC' \\ \mathbb{0} & E' \end{smallmatrix}\right)\right]\right)$$



Identities are defined in the obvious way: $\mathbb{1}_n = (0, [\mathbb{0}_n], \mathbb{1}_n, !, ¡, [(\ )])$.

The definition of composition is coherent with the equivalence classes because, whenever $\Gamma \sim \Gamma_0$ with matrix P and $\Gamma' \sim \Gamma'_0$ with matrix P', $\Gamma' \circ \Gamma \sim \Gamma'_0 \circ \Gamma_0$ with matrix $\left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right)$. Composition is associative because the matrices relative to the vertices are $[\ ]$-equivalent. Clearly, composition is unital and we proved that $\mathcal{A}$ is a category. Now we prove that it is monoidal.

Lead by the interpretation of the matrices that define a morphism, we define monoidal product as follows.

$$\Gamma \otimes \Gamma' = \left(k+k', \left[\left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right)\right], \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right), \left(\begin{smallmatrix} C & \mathbb{0} \\ \mathbb{0} & C' \end{smallmatrix}\right), \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right), \left[\left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right)\right]\right)$$

The monoidal unit is the empty diagram: $\mathbb{I} = (0, [(\,)], (\,), (\,), (\,), [(\,)])$

The monoidal product is well-defined on equivalence classes because, whenever $\Gamma \sim \Gamma_0$ with matrix P and $\Gamma' \sim \Gamma'_0$ with matrix P', $\Gamma \otimes \Gamma' \sim \Gamma_0 \otimes \Gamma'_0$ with matrix $\left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right)$. Clearly, monoidal product is strictly associative and unital. Therefore, the pentagon and the triangle equations [18] hold trivially. The monoidal product is a bifunctor because $(\Gamma_0 \circ \Gamma) \otimes (\Gamma'_0 \circ \Gamma_0) \sim (\Gamma_0 \otimes \Gamma'_0) \circ (\Gamma \otimes \Gamma')$ with permutation matrix $P = \left(\begin{smallmatrix} \mathbb{1} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1} \end{smallmatrix}\right)$.

Thus, $\mathcal{A}$ is a monoidal category. Finally, we prove that it is symmetric. Let $\sigma_{m,n}$ indicate the symmetry: $\sigma_{m,n} = \left(0, [\mathbb{0}], \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1}_m \\ \mathbb{1}_n & \mathbb{0} \end{smallmatrix}\right), !, ¡, [(\,)]\right)$

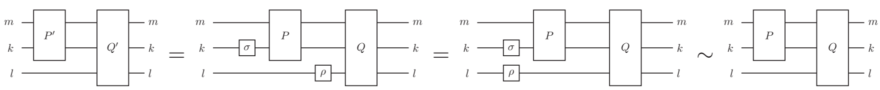
Clearly, the symmetry is its own inverse: $\sigma_{m,n} \circ \sigma_{n,m} = \mathbb{1}_{m+n}$.

Moreover, σ is natural as $\sigma_{n,n'} \circ (\Gamma \otimes \Gamma') \sim (\Gamma' \otimes \Gamma) \circ \sigma_{m,m'}$ with permutation matrix $P = \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1} \\ \mathbb{1} & \mathbb{0} \end{smallmatrix}\right)$.

Lastly, the symmetry satisfies the hexagon equations. Thus, $\mathcal{A}$ is a symmetric monoidal category whose objects are natural numbers. In other words, it is a prop. ◀

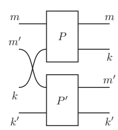
▶ **Lemma 34.** $\mathbf{b}\mathbb{P}$ *is a prop.*

Proof. The proof proceeds exactly as the previous one. We will use diagrammatic calculus of **Mat** for the permutation matrix of the morphisms in order to make the proofs more readable. We start by proving that $\mathbf{b}\mathbb{P}$ is a category. Composition is well-defined on equivalence classes by the monoidal structure of **Mat**. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$ and $(l, Q) \sim (l, \left(\begin{smallmatrix} \mathbb{1}_{m+k} & \mathbb{0} \\ \mathbb{0} & \rho \end{smallmatrix}\right) Q) = (l, Q')$.

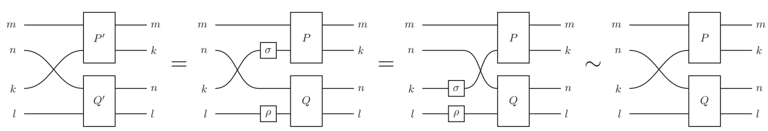


with permutation matrix $\left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \sigma & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \rho \end{smallmatrix}\right)$. Composition is clearly associative and unital because it is associative and unital in **Mat**. The monoidal product is defined with a symmetry on the left because we need to keep track of which of the inputs are bound.

$$(k, P) \otimes (k', P') = \left(k+k', \begin{pmatrix} \mathbb{1}_m & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{m'} & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_k & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{k'} \end{pmatrix} \left(\begin{smallmatrix} P & \mathbb{0} \\ \mathbb{0} & P' \end{smallmatrix}\right)\right)$$

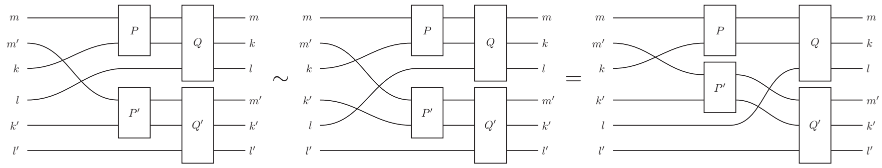


The monoidal unit is the empty diagram: $\mathbb{I} = (0, (\,))$. The monoidal product is well-defined on equivalence classes by naturality of the symmetries in **Mat**. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$ and $(l, Q) \sim (l, \left(\begin{smallmatrix} \mathbb{1}_n & \mathbb{0} \\ \mathbb{0} & \rho \end{smallmatrix}\right) Q) = (l, Q')$.



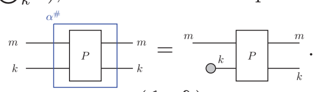
with permutation matrix $\left(\begin{smallmatrix} \mathbb{1}_{m+n} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \sigma & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \rho \end{smallmatrix}\right)$. The monoidal product is a functor because we can change the order in which we enumerate the vertices and because symmetries are natural in

Mat.



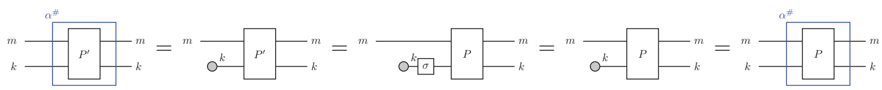
with matrix $\begin{pmatrix} \mathbb{1}_{m+m'+k} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1}_{k'} & \mathbb{0} \\ \mathbb{0} & \mathbb{1}_l & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{1}_{l'} \end{pmatrix}$. The monoidal product is clearly unital. The symmetry is lifted from **Mat**: $\sigma_{m,n} = (0, \left(\begin{smallmatrix} \mathbb{0} & \mathbb{1}_m \\ \mathbb{1}_n & \mathbb{0} \end{smallmatrix}\right))$. The symmetry is its own inverse and it satisfies the hexagon equations because it does so in **Mat**.
Therefore, $\mathbf{b}\mathbb{P}$ is a prop. ◀

▶ **Lemma 35.** $\mathbf{b}\mathbb{P}$ *is isomorphic to the free prop on one generator* $0 \to 1$.

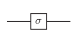
Proof. Define $\alpha^{\#} : \mathbf{b}\mathbb{P} \longrightarrow \mathbb{P}$ to be $\alpha^{\#}(k, P) = P \circ (\mathbb{1}_m \otimes \bigotimes_k v)$, where $P \in \mathbb{P}$ is the product of the symmetries that form P in $\mathbf{b}\mathbb{P}$. Diagrammatically, 

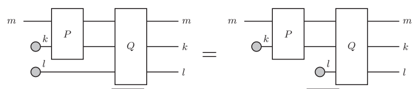
.

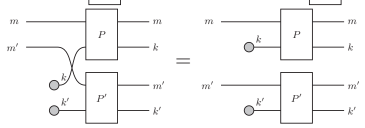
We prove that $\alpha^{\#}$ is well-defined on equivalence classes. Let $(k, P) \sim (k, \left(\begin{smallmatrix} \mathbb{1}_m & \mathbb{0} \\ \mathbb{0} & \sigma \end{smallmatrix}\right) P) = (k, P')$.



We show graphically that $\alpha^{\#}$ is a prop homomorphism

$\alpha^{\#}(\mathbb{0}) = \quad = \mathbb{0}$ $\quad \alpha^{\#}(\mathbb{1}_n) =$ $= \mathbb{1}_n$ $\quad \alpha^{\#}(0, \sigma) =$  $= \sigma$

$\alpha^{\#}((l, Q) \circ (k, P)) =$  $= \alpha^{\#}(l, Q) \circ \alpha^{\#}(k, P)$

$\alpha^{\#}((k, P) \otimes (k', P')) =$  $= \alpha^{\#}(k, P) \otimes \alpha^{\#}(k', P')$

and, by its definition,

$$\alpha^{\#}(\phi) = v$$

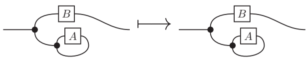
Moreover, $\alpha^{\#}$ is the unique morphism $\mathbf{b}\mathbb{P} \to \mathbb{P}$ with this property. In fact, suppose there is $\beta : \mathbf{b}\mathbb{P} \to \mathbb{P}$ such that $\beta(\phi) = v$. Then,

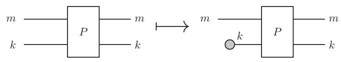
$$\begin{aligned} \beta(k, P) =& \beta((0, P) \circ ((0, \mathbb{1}_n) \otimes (k, \mathbb{1}_k))) = \beta(0, P) \circ (\beta(0, \mathbb{1}_n) \otimes \beta(k, \mathbb{1}_k)) \\ =& P \circ (\mathbb{1}_n \otimes \bigotimes_k v) = a^{\#}(k, P) \end{aligned}$$

Then $\mathbf{b}\mathbb{P}$ is isomorphic to the free prop over one generator $0 \to 1$. ◀

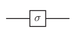
▷ Claim 36. The following are prop homomorphisms.

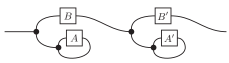
$$i_1\colon \mathbf{Adj} \longrightarrow \mathcal{A} \qquad\qquad i_2\colon \mathbf{b}\mathbb{P} \longrightarrow \mathcal{A}$$
$$n \longmapsto n \qquad\qquad n \longmapsto n$$
$$(B,[A]) \longmapsto (0,[A],B,!,¡,[(\ )]) \qquad\qquad (k,P) \longmapsto (k,[\mathbb{0}_n],P_*^{[1,n]},\mathbb{0}_{nk},P_*^{[n+1,n+k]},[\mathbb{0}_k])$$

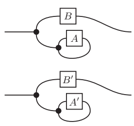


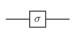


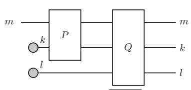
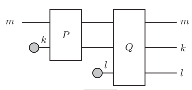
Proof. We prove graphically that they are prop homomorphisms.

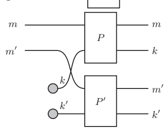
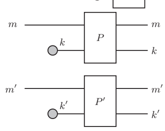
$i_1(\mathbb{0}) = \quad = \mathbb{0}$ $\quad i_1(\mathbb{1}_n) =$ $= \mathbb{1}_n$ $\quad i_1(\sigma,[(\ )]) =$  $= \sigma$

$i_1((B',[A']) \circ (B,[A])) =$  $= i_1(B',[A']) \circ i_1(B,[A])$

$i_1((B,[A]) \otimes (B',[A'])) =$  $= i_1(B,[A]) \otimes i_1(B',[A'])$

$i_2(\mathbb{0}) = \quad = \mathbb{0}$ $\quad i_2(\mathbb{1}_n) =$ $= \mathbb{1}_n$ $\quad i_2(0,\sigma) =$  $= \sigma$

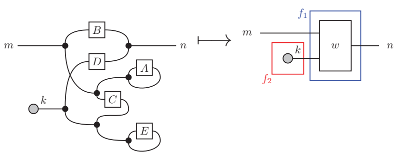
$i_2((l,Q) \circ (k,P)) =$  $=$  $= i_2(l,Q) \circ i_1(k,P)$

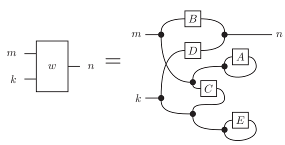
$i_2((k,P) \otimes (k',P')) =$  $=$  $= i_2(k,P) \otimes i_2(k',P')$

◀

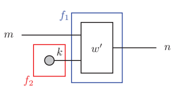
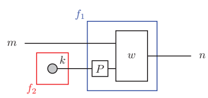
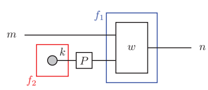
▶ **Lemma 37.** *H, defined on page 12, is a prop homomorphism.*

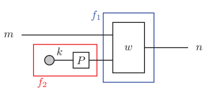
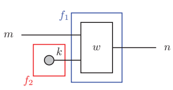
Proof. Recall that $H\colon \mathcal{A} \to \mathcal{C}$ is identity on objects and $H(k,[A],B,C,D,[E]) = f_1\left(\left(\begin{smallmatrix}B\\D\end{smallmatrix}\right),\left[\left(\begin{smallmatrix}A & C\\ \mathbb{0} & E\end{smallmatrix}\right)\right]\right) \circ (\mathbb{1}_m \otimes f_2(k,\mathbb{1}_k))$. By calling $w = \left(\left(\begin{smallmatrix}B\\D\end{smallmatrix}\right),\left[\left(\begin{smallmatrix}A & C\\ \mathbb{0} & E\end{smallmatrix}\right)\right]\right)$, which is a morphism in **Adj**, we can depict the image of H diagrammatically.



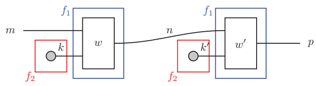
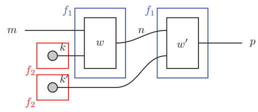


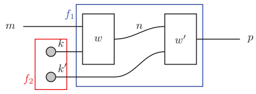
We need to prove that H is well-defined on equivalence classes. Let $\Gamma = (k,[A],B,C,D,[E]) \sim (k,[A],B,CP^T,PD,[PEP^T]) = \Gamma'$.

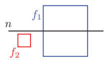
$H(\Gamma') =$  $=$  $=$ 

$=$  $=$  $= H(\Gamma)$

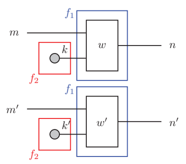
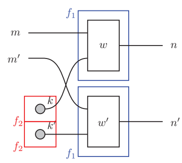
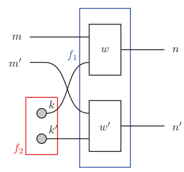
We prove that H is a prop homomorphism. Clearly, H is identity on objects. Moreover, it preserves composition, as it is shown by the diagrams.

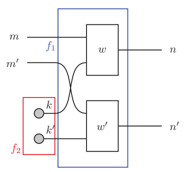
$H(\Gamma') \circ H(\Gamma) =$  $=$ 

$=$  $= H(\Gamma' \circ \Gamma)$

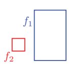
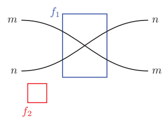
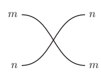
H preserves identities: $H(\mathbb{1}_n) =$  $=$ $= \mathbb{1}_n$.

H preserves monoidal product. This is also more clearly seen with string diagrams.

$H(\Gamma) \otimes H(\Gamma') =$  $=$  $=$ 

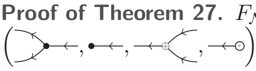
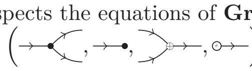
$=$  $= H(\Gamma \otimes \Gamma')$

It is easy to show that H preserves monoidal unit and symmetries.

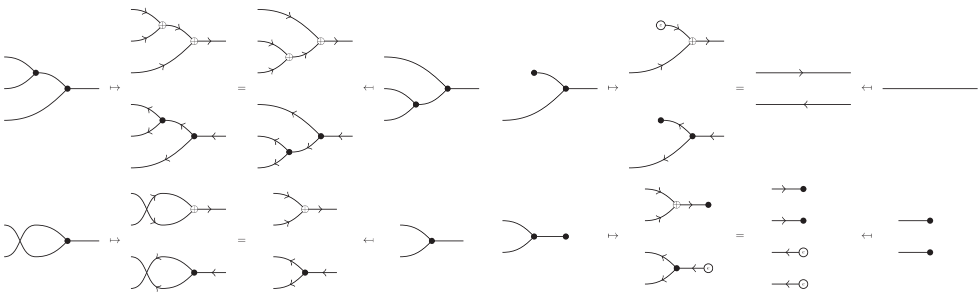
$H(\mathbb{I}) =$  $=$ $= \mathbb{I}$ $\quad H(\sigma_{m,n}) =$  $=$  $= \sigma_{m,n}$

◀

B Proofs for Sections 5

Proof of Theorem 27. $F_{\mathcal{N}}$ respects the equations of **Grph** because both the tuples  and  satisfy the commutative bialgebra axioms in figure 2, and they both interact as in figure 3 with the cup .

We explain in detail that the functor $F_{\mathcal{N}}$ preserves associativity of the black monoid, the rest of the equations are written with the same convention. We write on the left-most and right-most sides morphisms in **Grph** that, by associativity, they must be equal. In the centre, we write the morphisms in **Game** to which they are mapped (indicated with $\mapsto$) by the functor $F_{\mathcal{N}}$. These morphisms are equal in **Game** by associativity of the monoid operation $\oplus$ on M and coassociativity of copying. Thus, we can say that $F_{\mathcal{N}}$ preserves associativity of the black monoid.



and similarly for their transposed versions. ◀

Proof of Theorem 29. The proof proceeds by structural induction on Γ.
It is straightforward to check from the definition of $F_{\mathcal{N}}$ that the generators of **Grph** are sent to open games of the required form.
We need to check that composition is of the form as in the statement. We compute explicitly its play, coplay and best response functions.

$$\mathbb{P}_{F_{\mathcal{N}}(\Gamma' \circ \Gamma)}((\underline{\sigma}, \underline{\sigma}'), \underline{x}) = \mathbb{P}_{F_{\mathcal{N}}(\Gamma')}(\underline{\sigma}', B^T \underline{x} \oplus D^T f(\underline{\sigma})) = (BB')^T \underline{x} \oplus \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix} \right)^T f\left(\left(\begin{smallmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{smallmatrix} \right) \right)$$

$$\mathbb{C}_{F_{\mathcal{N}}(\Gamma' \circ \Gamma)}((\underline{\sigma}, \underline{\sigma}'), \underline{x}, \underline{q}) = \mathbb{C}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma}, \underline{x}, \mathbb{C}_{F_{\mathcal{N}}(\Gamma')}(\underline{\sigma}', \mathbb{P}_{F_{\mathcal{N}}(\Gamma)}(\underline{\sigma}, \underline{x}), \underline{q}))$$
$$= ((A + BA'B^T) + (A + BA'B^T)^T)\underline{x} \oplus BB'\underline{q} \oplus (C + B(A' + A'^T)D^T | BC')f\left(\left(\begin{smallmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{smallmatrix} \right) \right)$$

$$(\underline{\rho}, \underline{\rho}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma' \circ \Gamma)}(\underline{x}, \kappa)$$
$$\Leftrightarrow (\underline{\sigma}, \underline{\sigma}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma)}(\underline{x}, \kappa \circ F_{\mathcal{N}}(\Gamma')_{\underline{\tau}}) \wedge (\underline{\tau}, \underline{\tau}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma')}(F_{\mathcal{N}}(\Gamma)_{\underline{\sigma}} \circ \underline{x}, \kappa)$$
$$\Leftrightarrow \forall a = 1, ..., k + k'$$
$$s \in \operatorname*{argmax}_{s \in X} g\Bigg(s, \left(\begin{smallmatrix} C^T + D(A' + A'^T)B^T \\ C'^T B^T \end{smallmatrix} \right)^a_* \underline{x}$$
$$\oplus \left(\begin{smallmatrix} DB' \\ D' \end{smallmatrix} \right)^a_* \kappa((BB')^T \underline{x} \oplus (B'^T D^T | D'^T) f(\underline{\rho}[a \mapsto \rho'_a]))$$
$$\oplus \left(\begin{smallmatrix} E + E^T + D(A' + A'^T)D^T & DC' \\ C'^T D^T & E' + E'^T \end{smallmatrix} \right)^a_* f(\underline{\rho}[a \mapsto \rho'_a])\Bigg)$$

Similarly, we show that monoidal product has the desired form.

$$\mathbb{P}_{F_{\mathcal{N}}(\Gamma\otimes\Gamma')}((\underline{\sigma},\underline{\sigma}'),(\underline{x},\underline{x}')) = \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right)^T \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T f(\begin{pmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{pmatrix})$$

$$\begin{aligned}
&\mathbb{C}_{F_{\mathcal{N}}(\Gamma\otimes\Gamma')}((\underline{\sigma},\underline{\sigma}'),(\underline{x},\underline{x}'),(\underline{r},\underline{r}')) \\
&= \left(\left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right) + \left(\begin{smallmatrix} A & \mathbb{0} \\ \mathbb{0} & A' \end{smallmatrix}\right)^T\right) \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right) \begin{pmatrix} \underline{r} \\ \underline{r}' \end{pmatrix} \oplus \left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right) f(\begin{pmatrix} \underline{\sigma} \\ \underline{\sigma}' \end{pmatrix})
\end{aligned}$$

$$\begin{aligned}
&(\underline{\rho},\underline{\rho}') \in \mathbb{B}_{F_{\mathcal{N}}(\Gamma'\otimes\Gamma)}((\underline{x},\underline{x}'),\langle\kappa,\kappa'\rangle) \\
\Leftrightarrow &\forall a = \; ..., k+k' \;\; \rho'_a \in \operatorname*{argmax}_{s\in X} g\Big(s, \left(\left(\begin{smallmatrix} C & \mathbb{0} \\ \mathbb{0} & C' \end{smallmatrix}\right)^T\right)^a_* \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \oplus \left(\left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T\right)^a_* \langle\kappa,\kappa'\rangle((\left(\begin{smallmatrix} B & \mathbb{0} \\ \mathbb{0} & B' \end{smallmatrix}\right)^T \begin{pmatrix} \underline{x} \\ \underline{x}' \end{pmatrix} \\
&\oplus \left(\begin{smallmatrix} D & \mathbb{0} \\ \mathbb{0} & D' \end{smallmatrix}\right)^T f(\underline{\rho}\,[a \mapsto s])) \oplus \left(\left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right) + \left(\begin{smallmatrix} E & \mathbb{0} \\ \mathbb{0} & E' \end{smallmatrix}\right)^T\right)^a_* f(\underline{\rho}\,[a \mapsto s])\Big)
\end{aligned}$$

◀

Elena Di Lavore

November 7, 2023

Born in Milano, Italy, 13/08/1995
elenatalita@gmail.com

Address: Akadeemia tee 21/1, Tallinn 12611, Estonia
+393665334335

PhD candidate in theoretical computer science under the supervision of Professor Paweł Sobociński.

Publications

[1] 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].

[2] 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.

[3] 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.

[4] 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, 2023, pp. 268–283. DOI: 10.4204/EPTCS.380.16.

[5] 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.LO].

[6] 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].

[7] 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].

[8] Giovanni de Felice, Elena Di Lavore, Mario Román, and Alexis Toumi. "Functorial Language Games for Question Answering". In: *Electronic Proceedings in Theoretical Computer Science*. Vol. 333. Open Publishing Association, Feb. 2021, pp. 311–321. DOI: 10.4204/eptcs.333.21.

Note. As customary in mathematics, all my publications list the authors in alphabetical order.

Awards

Kleene Award to the best student paper [5], ACM/IEEE Sym. Log. Comp. Sci. 2022
Exemptions for High Academic Performance (Politecnico di Milano) 2015–2017
Best Freshers Award (Politecnico di Milano) 2015

Academic commitments

- (since May 2023) Member of the executive board of the Compositionality journal.
- (September 2022) Local co-organiser of the 9^{th} Symposium on Compositional Structures.
- (May 2022) Program committee member of the Applied Category Theory conference.
- (2021–2023) Co-organiser of the Applied Category Theory Adjoint School.
- Reviewer for conferences (LiCS, MFPS, ...) and journals (TAC, RAIRO, MSCS, ...).

Education

Tallinn University of Technology — Estonia
PhD — *2019–2023*

- Thesis: Monoidal Width
  Supervisor: Professor Paweł Sobociński
- Teaching experience as TA for the introductory course on Category Theory

University of Oxford — United Kingdom
MSc in Mathematics and Foundations of Computer Science — *2018–2019*

- Thesis: Subgame Perfection in Compositional Game Theory
  Supervisors: Dr Jules Hedges, Dr Jamie Vicary
- Mark: Merit

Università di Pisa — Italy
BSc in Mathematics — *2017–2018*

- Thesis: Data-driven Estimation for Nash Equilibria
  Supervisor: Professor Giancarlo Bigi
- Mark: 110 cum laude / 110

Politecnico di Milano — Italy
BSc in Mathematical Engineering — *2014–2017*

- Thesis: Floquet Theory Applied to a Perturbed Wave Equation
  Supervisor: Professor Gianni Arioli
- Mark: 110 cum laude / 110
- Studies abroad: Erasmus program at Linnaeus University, Växjö, Sweden

Other skills

Language skills: Italian (native speaker), English (C1).

Programming languages: basic knowledge of Idris, Matlab, C, R.

Elena Di Lavore

November 7, 2023

Sünniaeg ja -koht: Milano, Itaalia, 13/08/1995
elena.di@taltech.ee
Aadress: Akadeemia tee 21/1, Tallinn 12611, Estonia
+393665334335

Teoreetilise arvutiteaduse doktorant, juhendaja professor Paweł Sobociński.

Publikatsioonid

[1] 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].

[2] 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.

[3] 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.

[4] 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, 2023, pp. 268–283. DOI: 10.4204/EPTCS.380.16.

[5] 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.LO].

[6] 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].

[7] 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].

[8] Giovanni de Felice, Elena Di Lavore, Mario Román, and Alexis Toumi. "Functorial Language Games for Question Answering". In: *Electronic Proceedings in Theoretical Computer Science*. Vol. 333. Open Publishing Association, Feb. 2021, pp. 311–321. DOI: 10.4204/eptcs.333.21.

Märkus. Nagu matemaatikas kombeks, on kõigis minu väljaannetes autorid kirjas perekonnanimede tähestikulises järjekorras.

Autasud

Kleene Award to the best student paper [5], ACM/IEEE Sym. Log. Comp. Sci. 2022
Exemptions for High Academic Performance (Politecnico di Milano) 2015–2017
Best Freshers Award (Politecnico di Milano) 2015

Teadustegevus

- (alates mai 2023) Ajakirja Compositionality tegevjuhatuse liige.
- (september 2022) Kohalise kaaskorraldaja 9^{th} Symposium on Compositional Structures.
- (mai 2022) Programmkomitee liige Applied Category Theory konverentsil.
- (2021–2023) Kaaskorraldaja koolis Applied Category Theory Adjoint School.
- Referent konverentsidel (LiCS, MFPS, ...) ja ajakirjades (TAC, RAIRO, MSCS, ...).

Haridus

Tallinna Tehnikaülikool Eesti
Doktorantuur *2019–2023*

- Töö pealkiri: Monoidal Width
  Juhendaja: Professor Paweł Sobociński
- Õpetamiskogemus õppeassistendine aines sissejuhatus kategooriateooriasse

University of Oxford Ühendkuningriik
Magistrikraad Matemaatikas ja arvutiteaduse alustes *2018–2019*

- Töö pealkiri: Subgame Perfection in Compositional Game Theory
  Juhendajad: Dr Jules Hedges, Dr Jamie Vicary
- Hinne: Merit

Università di Pisa Itaalia
Bakalaureusekraad Matemaatikas *2017–2018*

- Töö pealkiri: Data-driven Estimation for Nash Equilibria
  Juhendaja: Professor Giancarlo Bigi
- Hinne: 110 cum laude / 110

Politecnico di Milano Itaalia
Bakalaureusekraad Ravamdusmatemaatikas *2014–2017*

- Töö pealkiri: Floquet Theory Applied to a Perturbed Wave Equation
  Juhendaja: Professor Gianni Arioli
- Hinne: 110 cum laude / 110
- Õpingud välismaal: Erasmus programm Linnéuniversitetet, Växjö, Rootsi

Muud oskused

Keelteoskus: itaalia keel (emakeel), inglise keel (C1).

Programmeerimiskeeled (algteadmised): Idris, Matlab, C, R.

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-076-8 (PDF)