DOCTORAL THESIS

Monoidal Context Theory

Mario Román García

TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY
TALLINN 2023

TALLINN UNIVERSITY OF TECHNOLOGY
DOCTORAL THESIS
54/2023

Monoidal Context Theory

MARIO ROMÁN GARCÍA

TALTECH PRESS

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 the 1st October 2023

Supervisor: Pawel Maria Sobocinski,
Department of Software Science, School of Information Technologies,
Tallinn University of Technology,
Tallinn, Estonia

Opponents: Professor Paul-André Melliès,
Université Paris Denis Diderot,
Paris, France

Professor Guy McCusker,
University of Bath,
Bath, United Kingdom

Defence of the thesis: 16 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.

Mario Román García

Mario Román

signature

Copyright: Mario Román García, 2023
ISSN 2585-6898 (publication)
ISBN 978-9916-80-081-2 (publication)
ISSN 2585-6901 (PDF)
ISBN 978-9916-80-082-9 (PDF)
Printed by Koopia Niini & Rauam

TALLINNA TEHNIKAÜLIKOOL
DOKTORITÖÖ
54/2023

Monoidiliste kontekstide teooria

MARIO ROMÁN GARCÍA

KIRJASTUS

Monoidal Context Theory

Mario Román

ABSTRACT. We universally characterize the produoidal category of monoidal lenses over a monoidal category. In the same way that each category induces a cofree promonoidal category of spliced arrows, each monoidal category induces a cofree produoidal category of monoidal spliced arrows; monoidal lenses are the free normalization of the cofree produoidal category of monoidal spliced arrows.

We apply the characterization of symmetric monoidal lenses to the analysis of multi-party message-passing protocols. We introduce a minimalistic axiomatization of message passing – message theories – and we construct combinatorially the free message theory over a set. Symmetric monoidal lenses are the derivations of the free message theory over a symmetric monoidal category.

MONOIDILISTE KONTEKSTIDE TEOORIA

KOKKUVÕTE. Karakteriseerime monoidiliste läätsede produoidilise kategooria universaalomaduse abil. Nii nagu iga kategooria indutseerib pleissitud noolte kovaba promonoidilise kategooria, indutseerib monoidiline kategooria monoidiliste pleissnoolte kovaba produoidilise kategooria; monoidilised läätsed on monoidiliste pleissnoolte kovaba produoidilise kategooria vaba normalisatsioon.

Kasutame sümmeetriliste monoidiliste läätsede karakterisatsiooni mitme osapoole sõnumiedastusprotokollide analüüsimiseks. Toome sisse sõnumiedastuse minimalistliku aksiomatisatsiooni – sõnumiteooriaid – ja konstrueerime vaba sõnumiteooria etteantud hulgal. Sümmeetrilised monoidilised läätsed on sümmeetrilise monoidilise kategooria vaba sõnumiteooria tuletised.

Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Mario Román García

11th November 2023

Acknowledgements. I would like to thank my PhD advisor, Pawel Sobocinski. Pawel has an exceptional ability to separate the scientifically promising ideas from the noise; he gave me the support, encouragement and freedom to pursue the research on this thesis. Pawel always said he wanted to replicate in Tallinn the atmosphere of Bob Walters' group in Sydney and I am particularly thankful for the result. I am also very grateful to Nicoletta Sabadini, for her advice and for sharing her encyclopedic knowledge of both automata and the history of Como.

Most ideas were cultivated at group meetings, and I want to thank Ed, Chad, Clémence, Nathan, Diana, Fosco, Elina, Amar, Cole, Philipp, Ekaterina, Niccoló, Michele, Andrea, and the rest of the *Tarkvarateaduse Instituut* for all the math and time we shared. I am very grateful to Niels, for his contagious enthusiasm and much useful feedback on this thesis. Special thanks go to Matt for his attention to detail and mathematical elegance, great discussions and ideas, and equally great book recommendations.

I learned and enjoyed a lot on short but productive visits to Pisa, Como, Oxford and Paris, and I want to thank Filippo, Alessandro, Vladimir, Louis and Davidad for many insightful discussions during this thesis. I had the privilege of having Giovanni, James, and Dylan as coauthors and I learned a lot from each one of them.

I thank the constant support of my parents, my brother Víctor, and my friends; I especially thank David and Esperanza for finding the best cafés in Granada. I thank Anna, Paolo, Enrico and Andrea for the time at the lake. Finally, I thank Elena: for all the fun we had writing each joint paper, and for all the happiness, math and drawings of these four years.

Contents

Declaration of Originality 8
Acknowledgements 9
Preface 16
Introduction 17
Processes and Diagrams 17
Algebra and Duoidal Algebra 18
Fundamental Structures for Message Passing 19
Global Effects 21
Monoidal Context Theory 21
Overview 22
Chapter 1: Process Theories 22
Chapter 2: Context Theory 22
Chapter 3: Monoidal Context Theory 22
Chapter 4: Monoidal Message Passing 22
Contributions 23
Publications 24

Chapter 1. Monoidal Process Theory 27
Monoidal Process Theory 27
1. Monoidal Categories 28
1.1. Strict Monoidal Categories 28
1.2. Some Words on Syntax 29
1.3. String Diagrams of Strict Monoidal Categories 30
1.4. Example: Crema di Mascarpone 33
1.5. Bibliography 33
2. Non-Strict Monoidal Categories 35
2.1. Non-Strictness 35
2.2. Coherence 36
2.3. String Diagrams of Monoidal Categories 36
2.4. Bibliography 37
3. String Diagrams of Bicategories 38
3.1. String diagrams of 2-categories 38

3.2. Bicategories 39
3.3. Example: Adjunctions 39
3.4. Bibliography 40
4. Symmetric Monoidal Categories and Do-Notation 42
4.1. Commutative Monoidal Categories 42
4.2. Symmetric Monoidal Categories 42
4.3. Do-Notation 45
4.4. Symmetry in Do-notation 47
4.5. Quotienting Do-notation 48
4.6. Example: the XOR Variable Swap 51
4.7. Bibliography 52
5. Cartesianity: Determinism and Totality 54
5.1. Cartesian Monoidal Categories 54
5.2. Partial Markov Categories 56
5.3. Bibliography 59
6. Premonoidal Categories 61
6.1. Premonoidal Categories 61
6.2. Effectful and Freyd Categories 63
6.3. Bibliography 65
7. String Diagrams for Premonoidal Categories 66
7.1. Effectful Polygraphs 66
7.2. Adding Runtime 66
7.3. Example: a Theory of Global State 73
7.4. Bibliography 74

Chapter 2. Context Theory 77
Context Theory 77
1. Profunctors and Coends 78
1.1. Profunctors 78
1.2. Dinaturality and Composition 79
1.3. Coend Calculus 80
1.4. The Point of Coend Calculus 80
1.5. Promonads 82
1.6. Bibliography 83
2. Multicategories 84
2.1. Multicategories 84
2.2. The Category of Multicategories 84
2.3. Application: Shufflings 85
3. Malleable Multicategories 86
3.1. Promonoidal Categories 87
3.2. Promonoidal Categories are Malleable Multicategories 88

3.3. Bibliography 90
4. The Splice-Contour Adjunction 91
4.1. Contour of a multicategory 91
4.2. Spliced Arrows 92
4.3. Splice-Contour Adjunction 93
4.4. Promonoidal Splice-Contour 94

Chapter 3. Monoidal Context Theory 97
Monoidal Context Theory 97
1. Duoidal categories 98
1.1. Duoidal Categories 98
1.2. Communication via Duoidals 98
1.3. Duoidals via adjoint monoids 99
1.4. Be Careful with Duoidal Coherence 101
1.5. Bibliography 103
2. Normal Duoidal Categories 104
2.1. Normalization of duoidal categories 104
2.2. Physical duoidal categories 105
2.3. Physical Lax Tensor of a Physical Duoidal Category 107
2.4. Bibliography 108
3. Produoidal Decomposition of Monoidal Categories 109
3.1. Produoidal categories 109
3.2. Monoidal Contour of a Produoidal Category 110
3.3. Produoidal Splice of a Monoidal Category 112
3.4. A Representable Parallel Structure 115
3.5. Bibliography 116
4. Interlude: Produoidal Normalization 117
4.1. Normal Produoidal Categories 117
4.2. The Normalization Monad 117
4.3. Symmetric Normalization 122
4.4. Bibliography 122
5. Monoidal Lenses 124
5.1. The Category of Monoidal Lenses 124
5.2. Symmetric Monoidal Lenses 126
5.3. Towards Message Theories 127
5.4. Bibliography 130

Chapter 4. Monoidal Message Passing 133
Monoidal Message Passing 133
1. Message Theories 134
1.1. Message Theories 134
1.2. Properties of a Message Theory 136

1.3. Coherence for Message Theories 139
Bibliography 141
2. Physical Monoidal Multicategories, and Shufflings 142
2.1. Symmetric Multicategories 142
2.2. Monoidal Multicategories 142
2.3. Physical Monoidal Multicategories 143
2.4. Shuffling 143
2.5. Bibliography 145
3. Polarization 146
3.1. Monoidal Polarization 146
3.2. Monoidal Polarization is Not Enough 147
3.3. Polarization of a Physical Monoidal Multicategory 149
3.4. Bibliography 149
4. Polar Shuffles 151
4.1. Polar Shuffles 151
4.2. Encoding of polar shuffles 151
4.3. The Multicategory of Polar Shuffles 152
4.4. Message Theories are Algebras of Polar Shuffles 154
4.5. Bibliography 156
5. Processes versus Sessions 157
5.1. Processes of a message theory 157
5.2. Sessions of a process theory 159
5.3. Sessions versus Processes 162
5.4. Example: One-Time Pad, as a Message Session 164
5.5. Case Study: Causal versus Evidential Decision Theories 166
Bibliography 170

Chapter 5. Conclusions and Further Work 173
Conclusions 173
Monoidal Context Theory 173
Monoidal Message Passing 174
Future Work 175

Bibliography 177

Appendix A. Supplementary material 193
1. Coherence diagrams for a duoidal category 194
2. Polycategories 196
2.1. Polycategories 196
2.2. The Category of Polycategories 197
2.3. Polycategorical Contour 198
2.4. Malleable Polycategories 199

2.5. Prostar-Autonomous Categories 200
2.6. Prostar Autonomous are Malleable Polycategories 201
2.7. Splice of a Polycategory 202
2.8. Bibliography 203

Appendix B. Publications 205
1. Span(Graph): a Canonical Feedback Algebra of Open Transition Systems 205
2. Monoidal Streams for Dataflow Programming 255
3. Promonads and String Diagrams for Effectful Categories 292
4. The Produoidal Algebra of Process Decomposition 311
5. Open Diagrams via Coend Calculus 368
6. Collages of String Diagrams 386
7. A Canonical Algebra of Open Transition Systems 414
8. Profunctor Optics, a Categorical Update 439

Appendix C. Curriculum Vitae 485

Appendix D. Curriculum Vitae (Eesti keeles) 489

Appendix E. Non-Exclusive License for Reproduction and Publication 493

Preface

Understanding and correctly designing intelligent and explainable systems could be both, if we get it right, one of the most beneficial human advancements; and, if we get it wrong, an existential risk for humanity [Ord20]. Humanity's need for languages and formalisms for trustworthy complex systems is now an urge.

Mathematics may possibly be the only right tool for this; but mathematics has not always been concerned with complex and interconnected systems. John von Neumann, talking about the intelligent and complex system that is the human brain, famously noted that

> the outward forms of our mathematics are not absolutely relevant from the point of view of evaluating what the mathematical or logical language truly used by the central nervous system is. However, the above remarks about reliability and logical and arithmetical depth prove that whatever the system is, it cannot fail to differ considerably from what we consciously and explicitly consider as mathematics.
>
> – John Von Neumann, *The Computer and The Brain* [vN58].

Meanwhile, when we try to describe big interconnected networks with linear algebra, geometry and calculus, even with all of our achievements, we seem to miss the point: things get extremely complicated, computationally intractable, humanly unimaginable; and we declare our defeat, we resort to vague analogies, and we ask an impenetrable pile of linear algebra to be our oracle.

This does not need to be our strategy: mathematics and computer science do not advance with bigger computations; they advance with new conceptual understanding. The past century saw the rise of conceptual mathematics and theoretical computer science – the kind of mathematics that took seriously the most elementary notions and cultivated them to tame complex abstractions and systems [LS09, PC07, Gro85]. Slowly but surely, the development of the conceptual theory of categories has brought us to a point where we can forget about comforting but vague analogies and start talking about complex systems formally and scientifically.

This thesis is part of the ongoing effort to find better languages and reasoning tools for science, epistemology, causality and probability: both intuitive graphical syntaxes for humans to reason with, and formal languages for computers, linked by a trusted and transparent mathematical formalism.

Introduction

Processes and Diagrams. Processes come intuitively to us; descriptions of processes arose independently all across science and engineering, in the form of diagrams, flowcharts or prose. We reason with them and we depict them all the time, but that does not mean that we always know how to interpret them: many diagrams in computer science and elsewhere do not have clear formal semantics, so we relegate them to serving merely as sources of intuition and inspiration.

> The notation has been found very useful in practice as it greatly simplifies the appearance of complicated tensor or spinor equations, the various interrelations expressed being discernable at a glance. Unfortunately the notation seems to be of value mainly for private calculations because it cannot be printed in the normal way. – Penrose and Rindler, *Spinors and Spacetime* [PR84]

Diagrams deserve better: we can lift diagrams from mere intuitions to mathematical structures; we can defend the legitimate and exceptional conceptual mathematics we now have to talk about processes and diagrams. This thesis follows the framework of symmetric monoidal categories. Processes that pass resources around and that compose sequentially and in parallel form symmetric monoidal categories; diagrams that depict these processes are no less than a sound and complete formal syntax for symmetric monoidal categories (e.g. Figure 1).

We will develop formal syntaxes for the compositional description of process, in particular for – but not restricted to – probabilistic, effectful and non-classical processes. We make use of category theory as a foundational tool: category theory allows us to characterize a syntactic construction as the one generating a universal semantics object and, at the same time, it provides a robust classification framework for mathematical structures.

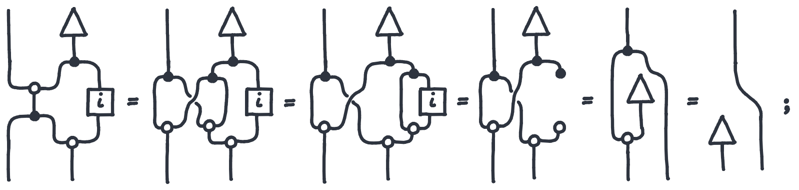


FIGURE 1. String-diagrammatic correctness proof for the One-time pad protocol (Proposition 5.10, [BK22]).

Algebra and Duoidal Algebra. The main technical idea of this thesis is natural: in the same way that the analysis of classical algebraic theories required the development of multicategories – and more precisely, of cartesian multicategories and Lawvere theories – the analysis of process theories, which are themselves two-dimensional algebraic theories, requires the development of monoidal multicategories and duoidal categories.

Multicategories, or colored operads, are mathematical structures that describe algebraic theories. In 1963, Lawvere introduced a categorical approach to universal algebra [Law63]: a theory can be captured by the cartesian multicategory containing all of its derived operations, and this notion is invariant to the specific primitive operations we choose to present the theory. This idea opens the field of functorial semantics: theories are categories, models are functors, and homomorphisms are natural transformations. More importantly, Lawvere's thesis gives a robust account of classical algebra that can be modified to suit our needs: the same framework can be employed for deductive systems [Lam69], higher-order algebra [Lam86], relational algebra [BPS17], or partial algebra [DLLNS21].

How does it apply to process theories? Monoidal categories and multicategories are not structured enough for the task of describing 2-dimensional structures themselves: we need duoidal categories and produoidal categories [Str12]. Intermediate algebraic expressions with variables are not complete expressions; they are only *contexts* into which we can plug values, and context is of central importance in computer science: we model not only processes but also the environment in which they act. While the algebra of 1-dimensional context is commonplace in applications like parsing [MZ22], the same concept was missing for 2-dimensional syntaxes, which are still less frequent in computer science [UVZ18, ES22].

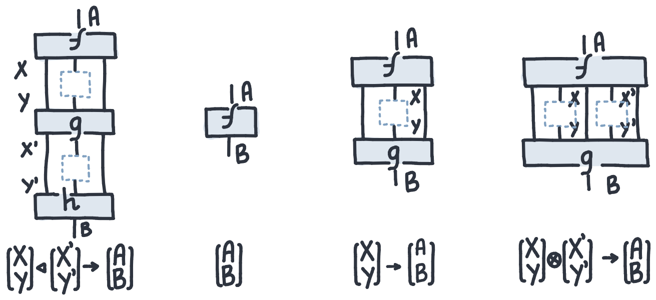

A
f
X
Y
g
X'
Y'
h
B
$\begin{bmatrix}X\\Y\end{bmatrix} \triangleleft \begin{bmatrix}X'\\Y'\end{bmatrix} \to \begin{bmatrix}A\\B\end{bmatrix}$
A
f
B
$\begin{bmatrix}A\\B\end{bmatrix}$
A
f
X
Y
g
B
$\begin{bmatrix}X\\Y\end{bmatrix} \to \begin{bmatrix}A\\B\end{bmatrix}$
A
f
X
Y
X'
Y'
g
B
$\begin{bmatrix}X\\Y\end{bmatrix} \otimes \begin{bmatrix}X'\\Y'\end{bmatrix} \to \begin{bmatrix}A\\B\end{bmatrix}$

FIGURE 2. A depiction of monoidal lenses, or incomplete processes.

Duoidal categories are well-known and there is a reasonable body of literature primarily concerned with applications in pure algebra and algebraic topology [AM10, Str12]; but the usage of duoidal categories to study processes is less frequent: two notable examples are the treatment of commutativity in the work of Garner and López Franco [GF16], and the study of "compositional dependencies" in the recent work of Spivak and Shapiro [SS22]. In this text, duoidal categories and monoidal multicategories allow us to postulate axioms for modularity and message passing; these axioms apply to any symmetric monoidal category, or any process theory.

Fundamental Structures for Message Passing. This main idea has an immediate consequence that we explore in the second part of this thesis: we can now develop an algebra for incomplete processes and their communication. While *concurrent* software has been intensively studied since the early 60s, the theoretical research landscape remains quite fragmented: we do not have a satisfactory understanding of the underlying mathematical principles of concurrency, and the proliferation of models has not helped us understand how they relate. Indeed, Abramsky [Abr05] argued in 2006 that we simply do not know what the fundamental structures of concurrency are.

A way to identify such principles and arrive at more canonical models is to look for logical or universal properties. An example of the former is the discovery of and work on Curry-Howard style connections between calculi for concurrency and fragments of linear logic, which led to the development of session types [Hon93, Dd09]. We take the latter route: departing from monoidal categories and their theory of context, we universally characterize a minimalistic axiomatization of message passing in process theories.

Concurrent message passing assumes two principles: interleaving and polarization. Polarization is a categorical technique to construct dualities; and in message passing, it constructs the duality between sending and receiving [CS07, Nes21, Mel21]. Interleaving is well-known in concurrency, and it models the ability of multiple processes to advance in parallel by mixing their global effects: imagine multiple processes determined by a sequence of statements; their concurrent execution may shuffle these statements in any possible order – the only requirement is to preserve the relative order of statements within any single process. We will not only propose a minimalistic axiomatization of message passing from these two principles, but we will also characterize the universal structures for message passing on a process theory.

Briefly, we assume polarized types, $X^\bullet$ and X°, that correspond to *sending* and *receiving*; and ordered lists of types describe sessions. Our axioms ask that *(i)* a sending port can be linked to a receiving port; *(ii)* echoing allows us to receive and then send; *(iii)* sequences of actions can be interleaved by a shuffling τ; and *(iv)* there exists a no-operation that does nothing.

$$\frac{\Gamma, X^{\bullet}, X^{\circ}, \Delta}{\Gamma, \Delta}\ (\textsc{com}) \qquad \frac{}{X^{\circ}, X^{\bullet}}\ (\textsc{spw}) \qquad \frac{\Gamma \qquad \Delta}{\tau(\Gamma, \Delta)}\ (\textsc{shf}_{\tau}) \qquad \frac{}{()}\ (\textsc{nop})$$

FIGURE 3. Type-theoretic presentation of a message theory.

This is a naive logic of message passing, but its strength is that it can be characterized mathematically using duoidal categories and, more concretely, physical monoidal multicategories, which we introduce. This paves the way to an adjunction that characterizes the free message theory on top of any process theory. The idea is simple but powerful: in order to construct message theories, we need to add global effects for *sending* and *receiving* to our process theories [OY16]; Theorem 5.9 notices that the diagrams for resulting effectful process theories can be wired precisely in the ways that the minimalistic logic of message passing prescribes.

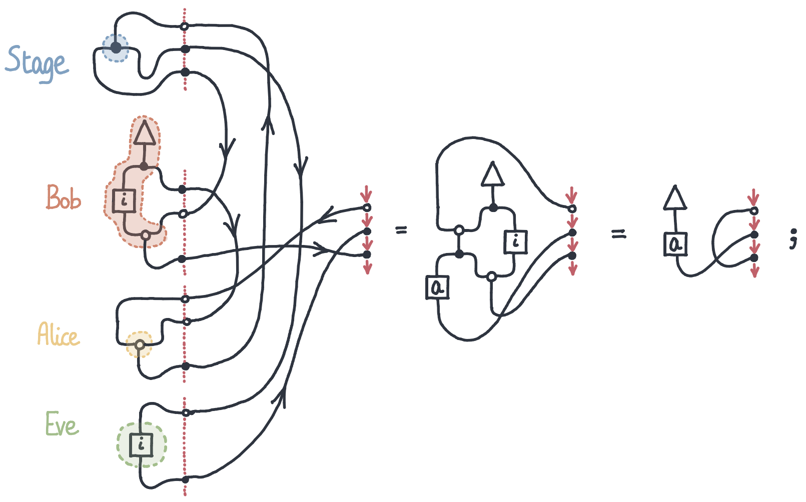


FIGURE 4. One-time pad protocol, split in four actors, mixed with a shuffle.

This means that the only addition to our process are two global effects (sending and receiving), that we depict using special red wires in the string diagrams. Each party in a session will have one of these red wires, and the logic of message passing allows us to combine them together. For instance, if the one-time pad protocol consists of a party (say, Alice) sending a message to another party (say, Bob), with an attacker (say, Eve), sharing a Stage that only allows broadcasting of messages; then these are four parties that connect together (Figure 4).

Global Effects. It remains then to explain the idea of global effects. Most imperative programming languages assume that there exist a *global state* that the program affects. Full parallelism is not possible when two programs need to change this global state in a specific order: they could run into *race conditions* [Huf54].

However, mathematical theories of processes often assume no global state; processes do not interact with each other except when it is explicit that they do. This property is called *purity* in some functional programming languages [HJW⁺92] and that makes it easier to reason with them. The problem is that even pure functional programming languages need some techniques to change global state, and mathematical structures like *monads* [Mog91] or *arrows* [Hug00] achieve precisely this – they take a pure theory and endow it with global effects.

Effects, monads and arrows create premonoidal categories [Pow02, HJ06]. These are not monoidal categories, but Alan Jeffrey [Jef97a] still introduced a string diagrammatic calculus for them: it is similar to the string diagrammatic calculus of monoidal categories, but it adds a red wire to control effects. This thesis proves that the extra red wire ensures a sound and complete graphical calculus for premonoidal categories.

Monoidal Context Theory. All these ideas align to produce a theory of contexts, or incomplete processes, in monoidal categories. Each monoidal category can generate a premonoidal category with the global effects of sending and receiving. The string diagrams of this new premonoidal category can be combined using the logic of message theories, and in fact, they form the free message theory on top of the original process theory: we can use them to reason and decompose multi-party processes in arbitrary process theories.

Overview

Chapter 1: Process Theories. Chapter 1 is an introduction to monoidal categories and their string diagrammatic syntax. Section 1 defines strict monoidal categories in terms of process theories and introduces their string diagrams. Section 4 defines their *symmetric* counterpart and their type theory in terms of do-notation, while Sections 2 and 3 extend string diagrams to non-strict monoidal categories and bicategories, variants that we will employ later.

Section 6 is an introduction to premonoidal categories and effectful categories. Section 7 gives their string diagrammatic calculus and proves its soundness and completeness. Finally, Section 5 studies linearity, copying and discarding in terms of monoidal categories. This concludes a basic treatment of processes in terms of monoidal categories.

Chapter 2: Context Theory. Chapter 2 introduces profunctors, in Section 1, and multicategories, in Section 2, as the mathematical tools to analyze decomposition. Profunctors provide a canonical equivalence relation, dinaturality, that we use whenever we study decomposition; in fact, it brings us to consider malleable multicategories in Section 3. Section 4 presents the splice-contour adjunction between a category and its malleable multicategory of incomplete terms, or contexts.

Chapter 3: Monoidal Context Theory. Chapter 3 brings context theory to the monoidal setting. Section 1 and Section 2 introduce duoidal categories and normal duoidal categories. The duoidal counterpart of malleable multicategories are produoidal categories and we introduce their splice-contour adjunction in Section 3. The idempotent normalization monad of produoidal categories is constructed in Section 4, and it is used in Section 5 to normalize monoidal spliced arrows and obtain a universal characterization of monoidal lenses.

Chapter 4: Monoidal Message Passing. Chapter 4 starts defining message theories in Section 1. Section 2 studies its categorical semantics in terms of physical monoidal multicategories. Section 3 introduces polarization and opens the way for Section 4 to define polar shuffles and prove that they form a free polarized monoidal multicategory. Section 5 constructs an adjunction between process theories and message theories.

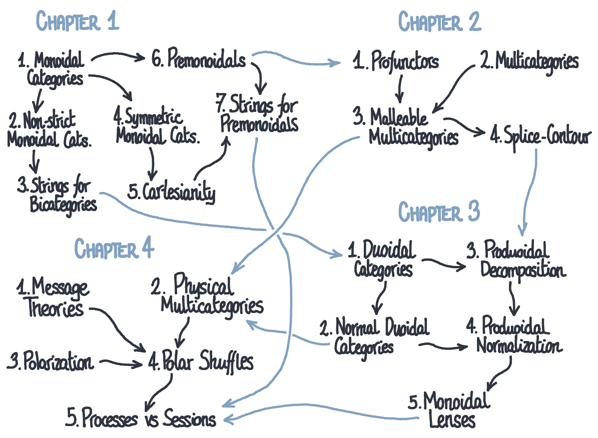


Figure 5. Chapter dependencies.

Contributions

The main results of this thesis are Theorem 5.3 and Theorem 5.9. They universally characterize, in two different ways, the produoidal structure of incomplete diagrams: the former is used for a theory of monoidal context, the latter is used for message passing.

The definition of *message theory* (Definitions 1.1 and 1.2 and proposition 1.4) is novel. There does not seem to be literature specifically on physical monoidal multicategories (Definition 2.4) nor on the observation that shuffles form the free one (Proposition 2.9) – even when, admittedly, these are all variations on the idea of physical duoidal categories and an old result by Grabowski [Gra81]. We give a different presentation of polarization in monoidal categories (Proposition 3.4), we discuss the problems of polarization in monoidal categories (Proposition 3.6) and we propose a solution describing polarization in physical monoidal multicategories (Definition 3.7). The definitions of polar shuffle (Definitions 4.1 and 4.2) and their physical monoidal multicategory (Theorem 4.7) are new contributions, as it is its proposed characterization as a free polarized physical monoidal multicategory (Theorem 4.11). Our main contribution is the final adjunction between sessions and processes (Theorem 5.9).

Duoidal categories are well-known, but we write down some observations about coherence in Proposition 1.6 and we contribute the definition of the physical tensor (Definition 2.14). Our main contribution is not only the monoidal splice-contour adjunction (Theorem 3.10); the adjunctions between produoidal categories and normal produoidal categories, and between symmetric produoidal categories and *physical* produoidal categories, with the construction of an idempotent monad Theorems 4.6 and 4.10, are contributions to pure category theory. Theorem 5.3 consitutes the first universal characterization of the whole produoidal category of lenses.

Even when do-notation is well-known, a categorical treatment like the one in Theorem 4.21 seemed to be missing from the literature; it is based in an exposition of string diagrams that is unusual in that it takes adjunctions as the main construction (Theorem 2.5). The string diagrams for premonoidal categories and effectful categories are a new formalization (Theorem 7.8) that is detailed in other papers by this author [Rom22]. We propose a new way of seeing coend calculus (Section 1.4) that is used briefly in this thesis but that is more extensively explained in other papers by this author [Rom20b]. The only contribution that we claim while translating the splice-contour adjunction to promonoidal categories is realizing their characterization as malleable multicategories (Proposition 3.10), which is admittedly a new spin on the usual characterization as closed multicategories.

Publications

The following is the list of publications authored or coauthored during the preparation of this thesis. As is customary in mathematics, we list authors in alphabetical order.

(1) Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and Mario Román. *Profunctor optics, a categorical update.* Accepted at *Compositionality*, preprint abs/2001.07488, 2020, [CEG$^+$20].
The author wrote the manuscript, proved the main theorem and prepared the literature review. The main ideas come from discussions with all the authors during the Applied Category Theory School. Bartosz Milewski proposed the original research problem. Jeremy Gibbons provided feedback and supervised previous work by this author on a similar topic. Derek Elkins and Fosco Loregian helped improve the mathematical presentation. Emily Pillmore revised and improved the Haskell implementation. Bryce Clarke noticed an important mistake in the first version of this work and proposed a solution.

(2) Mario Román. Open diagrams via coend calculus. *Applied Category Theory 2020. Electronic Proceedings in Theoretical Computer Science*, 333:65–78, Feb 2021, [Rom20b].
The author is the single author of this manuscript. The author defined the research question and all of the results of the paper.

(3) Guillaume Boisseau, Chad Nester, and Mario Román. Cornering optics. In *Applied Category Theory 2022*, Preprint abs/2205.00842, 2022, [BNR22].
The author found a problem with the initial versions of the draft and helped resolving it. The main idea for this paper is due to Guillaume Boisseau and Chad Nester. Chad Nester wrote the final version.

(4) Mario Román. Promonads and string diagrams for effectful categories. In Jade Master and Martha Lewis, editors, *Proceedings Fifth International Conference on Applied Category Theory, ACT 2022, Glasgow, United Kingdom, 18-22 July 2022*, volume 380 of *EPTCS*, pages 344–361, 2022, [Rom22].
The author is the single author of this manuscript. The author defined the research question and all of the results of the paper.

(5) Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Pawel Sobocinski. A canonical algebra of open transition systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2021, [LGR$^+$21].
The author proposed and proved the main theorem. The author wrote the manuscript together with Pawel Sobocinski and Elena Di Lavore. Nicoletta Sabadini proposed the research topic. Alessandro Gianola provided multiple examples.

(6) Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Pawel Sobocinski. Span(graph): a canonical feedback algebra of open transition systems. *Softw. Syst. Model.*, 22(2):495–520, 2023 [LGR$^+$23].
The author proposed and proved the main theorem. The author wrote the manuscript with help from Pawel Sobocinski and Elena Di Lavore. Nicoletta Sabadini proposed the research topic. Alessandro Gianola provided multiple examples.

(7) James Hefford and Mario Román. Optics for premonoidal categories. *Applied Category Theory 2023*, abs/2305.02906, 2023 [HR23].
The author proposed the main example and helped define the main results. James Hefford wrote the final manuscript.

(8) 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, LICS '22*, New York, NY, USA, 2022. Association for Computing Machinery. Kleene Award to the best student paper.
The author wrote most of the manuscript, proposed and proved the main theorem. Elena Di Lavore provided feedback on the manuscript and helped write appendices to the paper. Elena Di Lavore and the author found the motivation for this paper in discussion. Giovanni de Felice and the author proved together a result on controlled stochastic processes.

(9) Dylan Braithwaite and Mario Román. Collages of string diagrams. *Applied Category Theory 2023, preprint arXiv:2305.02675*, 2023 [BR23].
The author wrote most of the manuscript, proposed and proved the main theorem. Dylan Braithwaite had independently published a piece on a similar topic and revised the text, proposed multiple changes and linked the theory of collages.

(10) Elena Di Lavore and Mario Román. Evidential decision theory via partial Markov categories. In *Logic In Computer Science (LICS'23)*, pages 1–14, 2023 [LR23].
The author proposed multiple of the main results of this paper. Elena Di Lavore and the author wrote the paper together. Elena Di Lavore identified the research area.

(11) Matt Earnshaw, James Hefford, and Mario Román. The Produoidal Algebra of Process Decomposition, 2023. *In Peer-Review*, [EHR23].
The author wrote most of this article and proposed the main theorem and its proof. Matt Earnshaw found the link between the theory of splice-contour adjunctions and some previous literature. James Hefford provided the link with the theory of Tambara modules.

The Produoidal Algebra of Process Decomposition is the main unpublished work (currently in peer-review) that guides the writing of the main chapter of this thesis. It develops the universal characterization of monoidal lenses and forms the basis of Chapter 3 and Chapter 4. *Promonads and String Diagrams for Effectful Categories*, adapted, was used as the basis of Section 6 and Section 7 of Chapter 1.

CHAPTER 1

Monoidal Process Theory

Monoidal Process Theory

This chapter gives an overview of monoidal categories, their variants and their syntaxes. Monoidal categories are our framework of choice for *process theories*: we claim that the minimalistic axioms of monoidal categories capture what a process theory is and we assume them for the rest of the thesis.

Section 1 recalls monoidal categories and their string diagrams. Section 2 shows that the same axioms and syntax apply to *non-strict* monoidal categories and Section 3 extends them to bicategories, which we will briefly use later. Section 4 presents our definitive notion of process theory: symmetric monoidal categories. Symmetric monoidal categories have two syntaxes that are not commonly presented together: a string diagrammatic syntax in terms of hypergraphs and a term theoretic syntax – Hughes' *do-notation* [Hug00]. We argue that these two syntaxes further justify symmetric monoidal categories as a natural setting for processes.

There is a final concept that has been traditionally left out of monoidal categories: computational effects. We argue in Sections 6 and 7 that, far from being a problem that requires an extension of monoidal categories, as usually assumed, computational effects can still use the same diagrammatic syntax of string diagrams. This will be crucial for the next chapters in *message passing*: messages will constitute a computational effect, but our results in this chapter allow us to model them without having to leave the syntax of monoidal categories.

1. Monoidal Categories

1.1. Strict Monoidal Categories. Monoidal categories are an algebra of processes, with minimal axioms. The definition of monoidal category – and this thesis – follow a particular tradition of conceptual mathematics: *category theory*. Category theory aims to extract mathematical structures in an abstract and general form. As one such structure, monoidal categories are permissive: process theories like quantum maps and Markov kernels form monoidal categories [AC09, HV19, Fri20, CJ19]; and even relations among sets or the homomorphisms of modules over a ring form monoidal categories [BSS18, Alu21]. We start by reinterpreting MacLane's axioms for a monoidal category [ML71] in terms of processes.

Definition 1.1. A **strict monoidal category** $\mathbb{C}$ consists of a monoid of *objects*, or resources, $(\mathbb{C}_{obj}, \otimes, I)$, and a collection of *morphisms*, or processes, $\mathbb{C}(X;Y)$, indexed by an input $X \in \mathbb{C}_{obj}$ and an output $Y \in \mathbb{C}_{obj}$. A *strict monoidal category* is endowed with operations for the sequential and parallel composition of processes, respectively

$$(\fatsemi)\colon \mathbb{C}(X;Y) \times \mathbb{C}(Y;Z) \to \mathbb{C}(X;Z),$$
$$(\otimes)\colon \mathbb{C}(X;Y) \times \mathbb{C}(X';Y') \to \mathbb{C}(X \otimes X';Y \otimes Y'),$$

and a family of *identity* morphisms, $\mathrm{id}_X \in \mathbb{C}(X;X)$. Strict monoidal categories must satisfy the following axioms.

(1) Sequencing is unital, $f \fatsemi \mathrm{id}_Y = f$ and $\mathrm{id}_X \fatsemi f = f$.
(2) Sequencing is associative, $f \fatsemi (g \fatsemi h) = (f \fatsemi g) \fatsemi h$.
(3) Tensoring is unital, $f \otimes \mathrm{id}_I = f$ and $\mathrm{id}_I \otimes f = f$.
(4) Tensoring is associative, $f \otimes (g \otimes h) = (f \otimes g) \otimes h$.
(5) Tensoring and identities interchange, $\mathrm{id}_A \otimes \mathrm{id}_B = \mathrm{id}_{A \otimes B}$.
(6) Tensoring and sequencing interchange,

$$(f \fatsemi g) \otimes (f' \fatsemi g') = (f \otimes f') \fatsemi (g \otimes g').$$

Remark 1.2 (Process theories)**.** Objects are also known as *types* or *resources* [CFS16]. If X and Y are both resources, it is reasonable to assume their joint occurrence is also a resource, $X \otimes Y$; this joining operation, called *tensor* $(\otimes)$, must be unital with the empty resource I. Morphisms represent *transformations* or *processes*. If we have a process transforming X into Y and a process transforming Y into Z, we can *sequence* them $(\fatsemi)$ and create a process that transforms X into Z. The process that does nothing, the identity (id), is neutral for sequential composition. Similarly, transforming X into Y and transforming X' into Y' gives a way of transforming the joint object $X \otimes X'$ into $Y \otimes Y'$. Whenever we accept these basic constructions and axioms, we end up with strict monoidal categories.

Once we have accepted these basic axioms, the next sections develop a syntax for monoidal categories: *string diagrams.* String diagrams are an intuitive syntax for process that is sound and complete for the previous axioms.

1.2. Some Words on Syntax. What makes a mathematical syntax practical? Different syntaxes highlight different aspects of a proof, and we consider better those that make the more bureaucratic steps invisible. Syntaxes are an explicit construction of the free mathematical object with some algebraic structure; what makes them efficient is *how* we construct them.

For instance, how to prove that, in a group, the inverse of a multiplication is the reversed multiplication of the inverses? Usually, we simply observe that

$$(x \cdot y) \cdot (y^{-1} \cdot x^{-1}) = \cancel{x} \cdot \cancel{y \cdot y}^{-1} \cdot \cancel{x}^{-1} = e;$$

that is, a simple computation checks that each letter is cancelled by its inverse. But we could be more bureaucratic and argue that the correct proof is, actually,

$$\begin{aligned}(x \cdot y) \cdot (y^{-1} \cdot x^{-1}) &\overset{(i)}{=} x \cdot (y \cdot (y^{-1} \cdot x^{-1})) \\ &\overset{(ii)}{=} x \cdot ((y \cdot y^{-1}) \cdot x^{-1}) \\ &\overset{(iii)}{=} x \cdot (e \cdot x^{-1}) \\ &\overset{(iv)}{=} x \cdot x^{-1} \\ &\overset{(v)}{=} e.\end{aligned}$$

This proof uses associativity (*i, ii*), the definition of inverse (*iii, v*), and unitality (*iv*). What makes these two proofs different? We can argue that, implicitly, they are using different syntaxes, constructed in different ways [Shu16].

The bureaucratic syntax implicitly assumes the tautological construction of a free group. The free group on a set is generated by the elements of the set, the binary multiplication $(\cdot)$, the unit (e), and the inverse unary operator $({}^{-1})$; then, it is quotiented by associativity, unitality, and the inverse axioms. Tautological constructions only allow bureaucratic proofs – but we can do better.

How does one construct free objects non-tautologically? The usual strategy is to first show that some combinatorial structure possesses the desired algebraic structure (say, it forms a group with some selected elements). This combinatorial structure will be as simple as possible, will relegate most steps to computation, and will use minimal quotienting. The result that makes this recipe work is freeness: the fact that it defines an adjunction (say, there exists a unique map to any group with some elements).

More concretely, in our example, we know of a better classical construction of the free group: reduced words. *Reduced words* are lists containing some generators and their inverses. The only condition is that they cannot contain ocurrences of a generator followed by its inverse: they get automatically cancelled out.

Definition 1.3. Given a set A, the *reduced words* over it, $\mathsf{Word}(A)$, are lists of polarized elements of A – that is, a or a^{-1} for each $a \in A$ – not containing the substrings aa^{-1} or $a^{-1}a$ for any element $a \in A$.

Definition 1.4. The multiplication of two reduced words is inductively defined: if the first word is empty, then the multiplication is defined to be the second, $e \cdot w_2 = w_2$; however, if the first word consists of a letter and a word, aw_1 or $a^{-1}w_1$, then we consider two cases: we first compute $w_1 \cdot w_2$ by induction; if this word starts by the inverse of the first letter, $a^{-1} \cdot w'$ or $a \cdot w'$, then they both reduce and the multiplication is $w' \cdot w_2$, otherwise, we just append the first letter, $a(w_1 \cdot w_2)$.

Remark 1.5. It is non-trivial to prove that this multiplication is associative: the effort we put in here is the ease we get in return every time we use the syntax. We spare the reader this proof and we focus only on showcasing the syntax.

Proposition 1.6. *The inverse of a multiplication is the reversed multiplication of the inverses.*

Proof. Reduced words form a group, in fact, the free group over some generators. In the group of reduced words, $(xy) \cdot (y^{-1}x^{-1}) = e$ holds by definition. Because of freeness, there is a unique group homomorphism mapping this equality to any two elements of any other group. □

The core of this argument has been to construct, combinatorially, a left adjoint $\mathsf{Words} \colon \mathbf{Set} \to \mathbf{Group}$ to the forgetful functor $\mathsf{Forget} \colon \mathbf{Group} \to \mathbf{Set}$. This thesis will use adjoints as a more compositional way to discuss syntax. Let us start with the first of these syntaxes: string diagrams for monoidal categories.

1.3. String Diagrams of Strict Monoidal Categories. Monoidal categories have a sound and complete syntax in terms of string diagrams [JS91], which is the one we will use during this text. We may prefer the classical axioms of monoidal categories when proving that some category is indeed monoidal, but proving equalities in a monoidal category is easier using deformations of string diagrams – we will not need to remember the formulas. Accepting string diagrams and deformations as a criterion for equality is equivalent to accepting the axioms of strict monoidal categories: whenever we accept one, we accept the other.

A first example of this syntax describing a process is in Figure 1 [Sob13]. String diagrams construct an adjunction between a category of polygraphs and a category of strict monoidal categories.

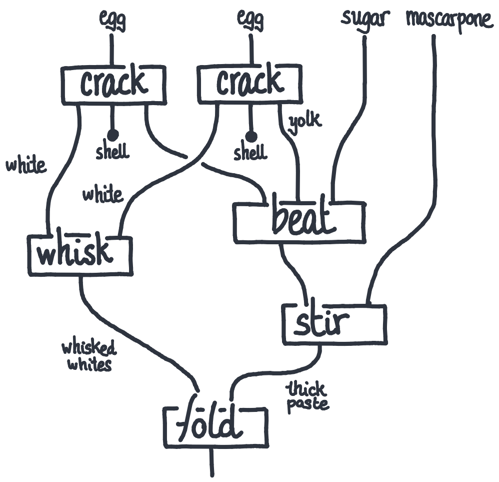


FIGURE 1. Process of preparing a Crema di Mascarpone, adapted from Sobocinski.

Definition 1.7. A *polygraph* $\mathcal{G}$ (analogue of a *multigraph* [Shu16]) is given by a set of objects, $\mathcal{G}_{obj}$, and a set of arrows $\mathcal{G}(A_0, \dots, A_n; B_0, \dots, B_m)$ for any two sequences of objects $A_0, \dots, A_n$ and $B_0, \dots, B_m$. A morphism of polygraphs $f \colon \mathcal{G} \to \mathcal{H}$ is a function between their object sets, $f_{\text{o}} \colon \mathcal{G}_{obj} \to \mathcal{H}_{obj}$, and a family of functions between their corresponding morphism sets for any two sequences of objects

$$f \colon \mathcal{G}(A_0, \dots, A_n; B_0, \dots, B_m) \to \mathcal{H}(f_{\text{o}}A_0, \dots, f_{\text{o}}A_n; f_{\text{o}}B_0, \dots, f_{\text{o}}B_m).$$

Polygraphs with polygraph homomorphisms form a category, **PolyGraph**.

Definition 1.8. A strict *monoidal functor*, $F \colon \mathbb{C} \to \mathbb{D}$, is a monoid homorphism between their object sets, $F_{obj} \colon \mathbb{C}_{obj} \to \mathbb{D}_{obj}$, and an assignment taking any morphism $f \in \mathbb{C}(X; Y)$ to a morphism $F(f) \in \mathbb{D}(FX; FY)$. A functor must preserve sequential composition, $F(f \mathbin{⨟} g) = F(f) \mathbin{⨟} F(g)$; parallel composition, $F(f \otimes g) = F(f) \otimes F(g)$; and identities, $F(\text{id}) = \text{id}$. Strict monoidal categories with strict monoidal functors form a category, $\mathbf{MonCat}_{\mathsf{Str}}$.

Definition 1.9. A *string diagram* over a *polygraph* $\mathcal{G}$ (or *progressive plane graph* in the work of Joyal and Street [JS91, Definition 1.1]) is a graph Γ embedded in the squared interval such that

(1) the boundary of the graph touches only the top and the bottom of the square, $\delta\Gamma \subseteq \{0, 1\} \times [0, 1]$;

(2) and the second projection is injective on each component of the graph without its vertices, $\Gamma - \Gamma_0$; this makes it acyclic and progressive.

We call to the components of $\Gamma - \Gamma_0$ *wires*, W; we call the vertices of the graph *nodes*, Γ_0. Wires must be labelled by the objects of the polygraph, $o \colon W \to \mathcal{G}_{obj}$, nodes must be labelled by the generators of the polygraph, $m \colon \Gamma_0 \to \mathcal{G}$; and each node must be connected to wires exactly typed by the objects of its generator – a string diagram must be well-typed.

Lemma 1.10. *String diagrams over a polygraph $\mathcal{G}$ form a monoidal category, which we call* $\mathsf{String}(\mathcal{G})$*. This determines a functor,*

$$\mathsf{String} \colon \mathbf{PolyGraph} \to \mathbf{MonCat}_{\mathsf{Str}}.$$

Proof sketch. The objects of the category are lists of objects of the polygraph, which we write as $[X_0, \dots, X_n]$, for $X_i \in \mathcal{G}_{obj}$. These form a (free) monoid with concatenation and the empty list.

Morphisms $[X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ are string diagrams over the polygraph $\mathcal{G}$ such that *(i)* the ordered list of wires that touches the upper boundary is typed by $[X_0, \dots, X_n]$, and *(ii)* the ordered list of wires that touches the lower boundary is typed by $[Y_0, \dots, Y_m]$.

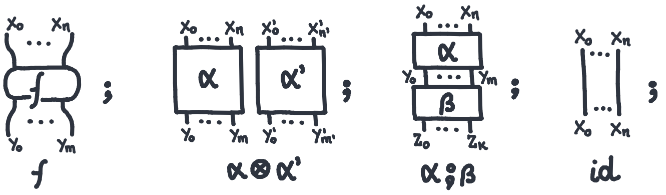


Figure 2. Strict monoidal category of string diagrams.

Figure 2 describes the operations of the category. The parallel composition of two diagrams $\alpha \colon [X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ and $\alpha' \colon [X'_0, \dots, X'_{n'}] \to [Y'_0, \dots, Y'_{m'}]$ is their horizontal juxtaposition. The sequential composition of two diagrams $\alpha \colon [X_0, \dots, X_n] \to [Y_0, \dots, Y_m]$ and $\beta \colon [Y_0, \dots, Y_m] \to [Z_0, \dots, Z_k]$ is the diagram obtained by vertical juxtaposition linking the outputs of the first to the inputs of the second. The identity on the object $[X_0, \dots, X_n]$ is given by a diagram containing n identity wires labelled by these objects. □

Lemma 1.11. *Forgetting about the sequential and parallel composition defines a functor from monoidal categories to polygraphs,*

$$\mathsf{Forget} \colon \mathbf{MonCat}_{\mathsf{Str}} \to \mathbf{PolyGraph}.$$

Proof. Any monoidal category $\mathbb{C}$ can be seen as a polygraph $\mathsf{Forget}(\mathbb{C})$ where the edges are determined by the morphisms,

$$\mathsf{Forget}(\mathbb{C})(A_0, \dots, A_n; B_0, \dots, B_m) = \mathbb{C}(A_0 \otimes \dots \otimes A_n, B_0 \otimes \dots \otimes B_m),$$

and we forget about composition and tensoring. It can be checked, by its definition, that any strict monoidal functor induces a homomorphism on the underlying polygraphs. □

Theorem 1.12 (Joyal and Street, [JS91, Theorem 2.3]). *There exists an adjunction between polygraphs and strict monoidal categories,* $\mathsf{String} \dashv \mathsf{Forget}$*. Given a polygraph $\mathcal{G}$, the free strict monoidal category $\mathsf{String}(\mathcal{G})$ is the strict monoidal category that has as morphisms the string diagrams over the generators of the polygraph; the underlying polygraph determines the right adjoint.*

1.4. Example: Crema di Mascarpone. This first example shows how to construct morphisms in a monoidal category. The theory for preparing crema di mascarpone contains the following resources,

$$\{\mathsf{egg}, \mathsf{white}, \mathsf{yolk}, \mathsf{shell}, \mathsf{whisked\ white}, \mathsf{sugar}, \mathsf{mascarpone}, \mathsf{paste}, \mathsf{thick\ paste}, \mathsf{crema}\}.$$

These resources are the objects of the polygraph also containing the following seven generators, as in Figure 3.

(1) crack: $\mathsf{egg} \to \mathsf{white} \otimes \mathsf{shell} \otimes \mathsf{yolk}$,
(2) beat: $\mathsf{yolk} \otimes \mathsf{yolk} \otimes \mathsf{sugar} \to \mathsf{paste}$,
(3) stir: $\mathsf{paste} \otimes \mathsf{mascarpone} \to \mathsf{thick\ paste}$,
(4) whisk: $\mathsf{white} \otimes \mathsf{white} \to \mathsf{whisked\ whites}$,
(5) fold: $\mathsf{whisked\ whites} \otimes \mathsf{thick\ paste} \to \mathsf{cream}$,
(6) swap: $\mathsf{yolk} \otimes \mathsf{white} \to \mathsf{white} \otimes \mathsf{yolk}$,
(7) discard: $\mathsf{shell} \to \mathrm{I}$.

All these resources form a polygraph $\mathcal{C}$. Thanks to the adjunction between polygraphs and monoidal categories, deciding how to interpret the resources and the generators of the polygraph in any monoidal category, $\mathcal{C} \to \mathsf{Forget}(\mathbb{D})$, is the same as creating a strict monoidal functor that interprets string diagrams in that category, $\mathsf{String}(\mathcal{C}) \to \mathbb{D}$. In particular, it interprets Figure 1 as a morphism in the monoidal category $\mathbb{D}$.

Usually, discarding and swapping are better understood as global structure with which all of the objects of the category are endowed. Even better than asking for the last generator in Figure 3, we will ask for the ability to copy or to swap resources freely. This is done via cartesian monoidal categories or symmetric monoidal categories, respectively.

1.5. Bibliography. Monoidal categories, together with their coherence theorem, were first introduced by MacLane [Mac63, Mac78], explicit equivalence theorems are given later by Joyal and Street [JS93]. String diagrams as progressive

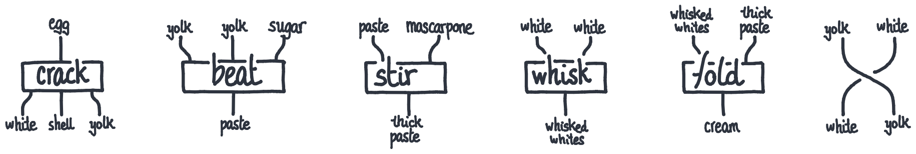


FIGURE 3. Polygraph for the theory of mascarpone.

graphs were introduced by Joyal [JS91]. Our presentation follows the *resource theories* of Coecke, Fritz and Spekkens [CFS16].

Petri nets and process calculi are alternative mathematical approaches to process theory; at the same time, they are arguably particular cases of monoidal categories [Sob10]. Monoidal categories as circuits and processes are explicitly pioneered by Sabadini, Walters, Carboni and Street [KSW97, CW87]. The string diagrammatic recipe for "crema di mascarpone" is an adaptation of a blog post by Sobocinski [Sob13]. We also follow the ideas of Shulman on categorical logic [Shu16].

2. Non-Strict Monoidal Categories

We have just argued for the axioms of strict monoidal categories as our process theories and string diagrams as our syntax. The bad news is that there exists a technicality preventing many interesting examples from ever forming a strict monoidal category; the good news is that neither the axioms of monoidal categories nor our string diagrams need to change at all to accommodate this technicality: this curious phenomenon is possible thanks to MacLane's *strictification and coherence* results [ML71]. This section introduces non-strict, or general monoidal categories, and it immediately details how MacLane's results ratify the axioms of strict monoidal categories.

2.1. Non-Strictness. The axioms of strict monoidal categories are enough to study a broader class of mathematical structures: non-strict monoidal categories.

What is a non-strict monoidal category? In a monoidal category, the tensor $(\otimes)$ is not required to be associative or unital. Because of how we usually construct our definitions, it is not always the case that $X \otimes (Y \otimes Z) = (X \otimes Y) \otimes Z$. Consider the theory of sets and functions, with the cartesian product $(\times)$ as the tensor. If we define

$$X \times Y = \{(x, y) \mid x \in X, y \in Y\},$$

then, simply, $X \times (Y \times Z) \neq (X \times Y) \times Z$. However, it is still the case that there exist two functions $X \times (Y \times Z) \to (X \times Y) \times Z$ and $(X \times Y) \times Z \to X \times (Y \times Z)$ and that these functions are mutual inverses for sequential composition. Whenever this happens in a category, we say these two objects are *isomorphic* $(\cong)$ and the morphisms are *isomorphisms.* In a monoidal category, the tensor is not associative and unital but it is still associative and unital up to an isomorphism. This may sound like a minor technicality, but it makes many examples fail to form a monoidal category.

Definition 2.1. A **monoidal category**, $(\mathbb{C}, \otimes, I, \alpha, \lambda, \rho)$, is a category $\mathbb{C}$ equipped with a pair of functors $(\otimes)\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, and $I \in \mathbb{C}$, called tensor and unit respectively. and three families of isomorphisms called the *coherence maps*:

(1) the associator $\alpha_{X,Y,Z}\colon (X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$,
(2) the left unitor $\lambda_A\colon I \otimes A \cong A$ and
(3) the right unitor $\rho_A\colon A \otimes I \cong A$.

These three families of maps must be natural, meaning they commute with other well-typed morphisms of the monoidal category. Moreover, these must be such that every formally well-typed equation between coherence maps holds.

Proposition 2.2. *A strict monoidal category is precisely a monoidal category where α, λ, and ρ are identities.*

Proof. The naturality of the coherence maps, whenever these are identities, is the same as the associativity and unitality of the tensor (3 and 4). Functoriality of the tensor is the same as the interchange axioms (5 and 6); while the functoriality of the unit is trivially true. □

2.2. Coherence. The conditions of the definition of a non-strict monoidal category may seem too strong: we are asking that a wide family of equations (all the formally well-typed ones) hold. Fortunately, the *coherence theorem* shows that simply checking two families of equations is enough.

Theorem 2.3 (MacLane, [ML71]). *The three different families of coherence maps* (α, λ, ρ) *satisfy all formally well-typed equations between them whenever they satisfy the* triangle *and* pentagon *equations:*

(1) $\alpha_{X,I,Y} \fatsemi (\mathrm{id}_X \otimes \lambda_Y) = \rho_X \otimes \mathrm{id}_Y$*, and*
(2) $(\alpha_{X,Y,Z} \otimes \mathrm{id}) \fatsemi \alpha_{X,Y\otimes Z,W} \fatsemi (\mathrm{id}_X \otimes \alpha_{Y,Z,W}) = \alpha_{X\otimes Y,Z,W} \fatsemi \alpha_{X,Y,Z\otimes W}$.

Finally, the theorem that allows strict monoidal categories to talk about non-strict monoidal categories is the *strictification theorem* that says that any monoidal category is monoidally equivalent to a strict one. This does determine an adjunction with extra structure, making it a 2-adjunction [Cam19].

Theorem 2.4 (Joyal and Street, [JS91]). *Any monoidal category is equivalent via a strong monoidal functor to a strict one. There exists a 2-adjunction between the category of monoidal categories and strong monoidal functors and the category of strict monoidal categories with strict monoidal functors. Moreover, the unit of this 2-adjunction is an equivalence.*

2.3. String Diagrams of Monoidal Categories. While it is true that we can construct the free strict monoidal category on a polygraph, it is not true yet that we know how to construct the free monoidal category (the non-strict one) over a polygraph; in fact, this seems to be impossible. This could be misread as saying that string diagrams are not equally sound and complete for monoidal categories. Nothing is further from the truth: even if there are now multiple ways of interpreting a string diagram in a monoidal category, these are essentially equal – they define isomorphic functors.

Theorem 2.5. *There is a pseudoadjunction between the locally discrete 2-category of polygraphs and the 2-category of monoidal categories, strong monoidal functors and monoidal natural transformations.*

Proof sketch. This pseudoadjunction arises as a combination of two different 2-adjunctions. The first one is the adjunction between polygraphs and strict monoidal categories we have studied before. The second one is described in Theorem 2.4, and its unit defines an equivalence: every monoidal category is equivalent to a strict one.

$$\begin{tikzcd}
\text{PolyGraph} \arrow[rr, bend left, "\text{String}"] & \text{MonCat} \arrow[l, bend left, "\text{Forget}"] \arrow[l, phantom, "\perp"] \arrow[r, bend left, "\text{Strict}"] \arrow[r, phantom, "\perp"] & \text{MonCat}_{\text{str}} \arrow[l, bend left, "\text{Include}"]
\end{tikzcd}$$

FIGURE 4. Pseudoadjunctions between polygraphs and monoidal categories.

It is well known that each time that we have two adjunctions in this disposition we can reduce one along the other, provided that the unit of the former is invertible (see Proposition 3.10). In this case, the unit is not exactly invertible but merely an equivalence: as a consequence, we obtain not another 2-adjunction but merely a pseudoadjunction. This concludes the proof. □

2.4. Bibliography. The coherence results go back to MacLane, Joyal and Street [Mac63, JS91]; Hermida arrived at the same result via multicategories [Her01, Had18]; and there is a more modern account by Becerra detailing the 2-adjunction between strict and non-strict monoidal categories [Bec23], which Campbell studies for bicategories [Cam19].

3. String Diagrams of Bicategories

Bicategories are the second extension of monoidal categories that we will employ during the text. If monoidal categories were well-suited to reason about process theories, bicategories, one level up, are well-suited to reason about categories themselves; however, their string diagrammatic syntaxes are very close.

String diagrams of monoidal categories can be easily extended to bicategories if we allow ourselves to color the regions. Coloring the regions simply constrains which objects can be tensored: this algebraic structure is a bicategory, also known as a *weak 2-category*. In a bicategory, two objects must coincide along a boundary to be tensored.

Definition 3.1. A strict *2-category* $\mathbb{B}$ consists of a collection of *objects*, or 0-cells, $\mathbb{B}_{obj}$, and a category of *morphisms* or 1-cells between any two objects, $\mathbb{B}(A; B)$. A strict *2-category* is endowed with operations for the parallel composition of 1-cells,

$$\begin{aligned}(\mathbin{;})\colon \mathbb{B}(A; B) \times \mathbb{B}(B; C) &\to \mathbb{B}(A; C),\\ (I_A)\colon \mathbb{B}(A; A),&\end{aligned}$$

that are associative and unital both on objects and morphisms, meaning that $(X \mathbin{;} Y) \mathbin{;} Z = X \mathbin{;} (Y \mathbin{;} Z)$, and $I_A \mathbin{;} X = X = X \mathbin{;} I_B$. Bicategories must satisfy the following axioms, making parallel composition a functor:

(1) parallel composition is unital, $f \mathbin{;} \mathrm{id} = f$, and $\mathrm{id} \mathbin{;} f = f$;
(2) parallel composition is associative, $f \mathbin{;} (g \mathbin{;} h) = (f \mathbin{;} g) \mathbin{;} h$;
(3) compositions are unital, $\mathrm{id} \mathbin{;} \mathrm{id} = \mathrm{id}$;
(4) compositions interchange, $(f \mathbin{⨾} g) \mathbin{;} (f' \mathbin{⨾} g') = (f \mathbin{;} f') \mathbin{⨾} (g \mathbin{;} g')$.

Remark 3.2. A single-object strict 2-category is exactly a strict monoidal category.

3.1. String diagrams of 2-categories. Let us briefly comment on how the string diagrams of monoidal categories extend to bicategories. We repeat the same definitions and the same theorems, just taking care of matching the boundaries this time.

Definition 3.3. A *bigraph*, or *2-graph*, $\mathcal{B}$ is given by a set of objects, $\mathcal{B}_{obj}$; a set of arrows between any two objects, $\mathcal{B}(A; B)$; and a set of 2-arrows between any two paths of arrows, $\mathcal{B}(X_0, \dots, X_n; Y_0, \dots, Y_m)$.

A bigraph homomorphism, $f\colon \mathcal{A} \to \mathcal{B}$, is a function between their object sets, $f_o\colon \mathcal{A}_{obj} \to \mathcal{B}_{obj}$; a family of functions between their corresponding arrow sets, $f\colon \mathcal{A}(A; B) \to \mathcal{B}(f(A), f(B))$; and a family of functions between their corresponding 2-arrow sets,

$$f\colon \mathcal{A}(X_0, \dots, X_n; Y_0, \dots, Y_m) \to \mathcal{B}(fX_0, \dots, fX_n; fY_0, \dots, fY_m).$$

Bigraphs with bigraph homomorphisms form a category, **BiGraph**.

Definition 3.4. A string diagram over a bigraph $\mathcal{A}$ is a string diagram over the polygraph formed by arrows and 2-arrows, additionally satisfying that each region is labelled by an object of the bigraph, and in such a way that any wire is labelled by an arrow connecting the labels of the two regions.

THEOREM 3.5. *There is an adjunction between bicategorical graphs and 2-categories with strict 2-functors between them. The left adjoint is given by colored string diagrams over the bigraph.*

3.2. Bicategories. Strict monoidal categories have a weak analogue that still shares the same syntax – (weak, or non-strict) monoidal categories. In the same way, strict 2-categories have a weak analogue that shares the same syntax: weak 2-categories, sometimes called bicategories.

Definition 3.6. A *bicategory* $(\mathbb{B}, ;, I, \alpha, \lambda, \rho)$ is a collection of 0-cells, $\mathbb{B}_{obj}$, together with a category $\mathbb{B}(A; B)$ between any two 0-cells, $A, B \in \mathbb{B}_{obj}$, and functors

$$(;)\colon \mathbb{B}(A;B) \times \mathbb{B}(B;C) \to \mathbb{B}(A;C), \text{ and } I_A \colon \mathbb{B}(A;A),$$

that are associative and unital up to isomorphism, meaning that there exist natural isomorphisms describing associativity $\alpha_{X,Y,Z} \colon (X \otimes Y) \otimes Z \cong X \otimes (Y \otimes Z)$, left unitality $\lambda_X \colon I_A \otimes X \cong X$ and right unitality $\rho_X \colon X \otimes I_B \cong B$.

Conjecture 3.7. *There is a pseudoadjunction between the locally discrete 2-category of bicategorical graphs and the 2-category of bicategories, pseudofunctors and icons (see Campbell, Garner and Gurski's work for the higher structure of the strictification adjunction [Cam19, GG09]).*

3.3. Example: Adjunctions. We exemplify the usage of string diagrams for bicategories in an abstract definition of adjunctions. We then use string diagrams to prove a theorem about adjunctions.

Definition 3.8. The *theory of a duality* in a bicategory contains two 0-cells A and B; it contains two 1-cells between them, $L \colon A \to B$ and $R \colon B \to A$, and it contains two 2-cells, $\varepsilon \colon L ⨾ R \to I$ and $\eta \colon I \to R ⨾ L$, that satisfy $(\mathrm{id} \otimes \eta) ⨾ (\varepsilon \otimes \mathrm{id}) = \mathrm{id}$ and $(\eta \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes \varepsilon) = \mathrm{id}$.

Remark 3.9. Adjunctions are dualities in the bicategory of categories, functors, and natural transformations.

Proposition 3.10 (Reducing an adjunction). *Let $F \colon \mathbb{A} \to \mathbb{C}$ and $H ⨾ U \colon \mathbb{C} \to \mathbb{A}$ determine an adjunction $(F, H ⨾ U, \eta, \varepsilon)$ and let $P \colon \mathbb{B} \to \mathbb{C}$ determine a second adjunction (P, H, u, c) such that the unit $u \colon I \to P ⨾ H$ is a natural isomorphism (as in Figure 6). Then, $F ⨾ H$ is left adjoint to U.*

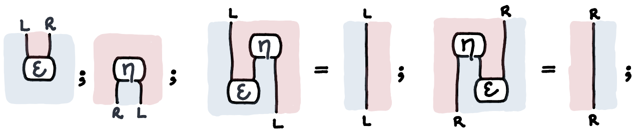


FIGURE 5. Theory of a duality.

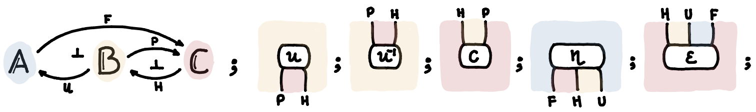


FIGURE 6. Setting for reducing an adjunction.

PROOF. We employ the string diagrammatic calculus of bicategories for the bicategory of categories, functors and natural transformations [Mar14]. We define the morphisms in Figure 7 to be the unit and the counit of the adjunction. We then prove that they satisfy the snake equations in Figures 7 and 8.

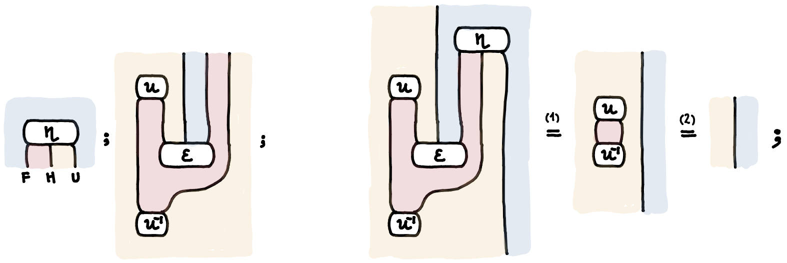


FIGURE 7. Unit and counit of the reduced adjunction (left). First snake equation (right).

In the first snake equation, in Figure 7, we use *(i)* that there is a duality (η, ε), and *(ii)* that u is invertible. In the second snake equation, in Figure 8, we use *(i)* that there is a duality (u, c), *(ii)* that u is invertible, *(iii)* that there is a duality (u, c), again; and *(iv)* that there is a duality (η, ε). □

3.4. Bibliography. String diagrams for bicategories are usually interpreted as the Poincaré dual of *globular pasting diagrams*, which were used by Bénabou

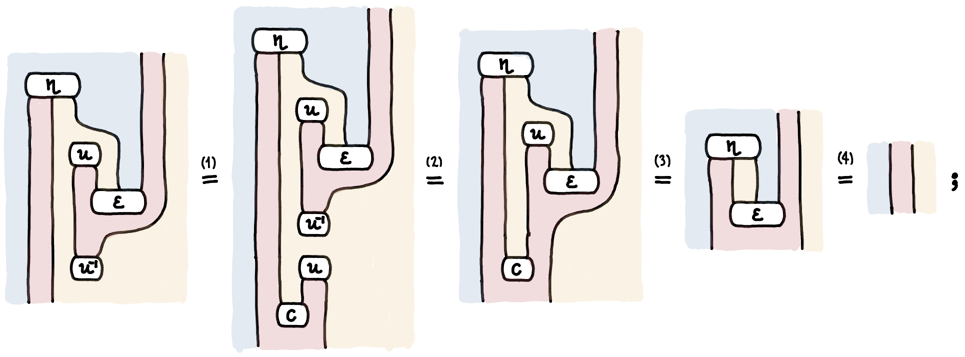


Figure 8. Second snake equation.

since the introduction of bicategories [Bén67]. Marsden uses string diagrams for bicategories to study basic formal category theory [Mar14].

4. Symmetric Monoidal Categories and Do-Notation

Setting strictness aside, an extra axiom sharpens monoidal categories for the study of process theories: *symmetry*. Symmetry states that the position that resources occupy is not important: $A \otimes B$ is worth the same as $B \otimes A$. Assuming symmetry simplifies process syntax drammatically – in fact, it also enables a new syntax that mimics imperative programming – and it is arguably an axiom that we need before we can really talk of *processes*.

This section introduces symmetric monoidal categories, their specialized string diagrams in terms of hypergraphs [BGK⁺16], and their do-notation syntax [Hug00].

4.1. Commutative Monoidal Categories.

4.1. Commutative Monoidal Categories. So far, the meaning of the tensor ($\otimes$) in process theories has arguably been too general: so far, we have asked $A \otimes B$ to represent the juxtaposition of resources – resources form a monoid. Why not a commutative monoid? If we interpret objects as bags of resources, it seems clear that a commutative monoid would be more appropriate. However, imposing commutativity naively fails catastrophically: the individuality of each resource disappears [MM90].

Definition 4.1. A *commutative monoidal category* is a strict monoidal category $(\mathbb{C}, \otimes, I)$ where objects form a commutative monoid, $A \otimes A' = A' \otimes A$ for each $A, A' \in \mathbb{C}_{obj}$, and the tensor of morphisms is also commutative, $f \otimes f' = f' \otimes f$ for any two $f \colon A \to B$ and $f' \colon A' \to B'$.

Proposition 4.2. *In a commutative monoidal category, resources do not have individuality: it does not matter to which of them we apply a transformation, and not even the order in which we apply them. More formally,*

$$(f \mathbin{⨟} g) \otimes \mathrm{id} = (f \otimes g) = \mathrm{id} \otimes (g \mathbin{⨟} f)$$

for each two transformations of the same resource, $f \colon X \to X$ and $g \colon X \to X$.

This may be useful in specific applications. Indeed, it is one of the crucial ideas behind the formalization of Petri nets as monoids in the work of Meseguer and Montanari [MM90] – commutativity represents the "collective token philosophy" of Petri nets [BGMS21]. However, for our purposes, we will need a more refined notion that does not destroy the individuality of our resources: this notion is given by symmetric monoidal categories.

4.2. Symmetric Monoidal Categories. *Symmetric monoidal categories* do not assume that the monoid of objects is commutative; they only assume that there is a family of processes that allow us to reorder resources, making it commutative "up to an invertible process". In practice, this means that even when $X \otimes Y \neq Y \otimes X$, there exists a process $\sigma_{X,Y} \colon X \otimes Y \to Y \otimes X$ that is invertible. These are our definitive notion of *process theory*.

Definition 4.3. A strict *symmetric monoidal category* (or a *permutative category*) is a strict monoidal category $(\mathbb{C}, \otimes, I)$ endowed with a family of maps $\sigma_{X,Y} \colon X \otimes Y \to Y \otimes X$ that satisfy the following equations describing how to

(1) swap nothing, $\sigma_{I,X} = \mathrm{id}_X = \sigma_{X,I}$;
(2) swap resources on the left, $\sigma_{X,Y\otimes Z} = (\sigma_{X,Y} \otimes \mathrm{id}_Z) ⨾ (\mathrm{id}_Y \otimes \sigma_{X,Z})$;
(3) swap resources on the right, $\sigma_{X\otimes Y,Z} = (\mathrm{id}_X \otimes \sigma_{Y,Z}) ⨾ (\sigma_{X,Z} \otimes \mathrm{id}_Y)$;
(4) reverse a swap with a swap, $\sigma_{X,Y} ⨾ \sigma_{Y,X} = \mathrm{id}_X \otimes \mathrm{id}_Y$;
(5) swap and apply transformations, $\sigma_{X,Y} ⨾ (f \otimes g) = (g \otimes f) ⨾ \sigma_{X',Y'}$;
(6) and swap in any order,

$$(\sigma_{X,Y} \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes \sigma_{X,Z}) ⨾ (\sigma_{Y,Z} \otimes \mathrm{id}) = (\mathrm{id} \otimes \sigma_{Y,Z}) ⨾ (\sigma_{X,Z} \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes \sigma_{X,Y}).$$

The first three axioms are especially important for clarifying all the rest: they say that the swapping process of any two objects in a freely generated monoidal category is determined by the swapping process of the generators. This already allow us to have a first string diagrammatic calculus for symmetric monoidal categories: the swap on the generators is represented by wires crossing; the swap on arbitrary objects is constructed from it; the rest of the axioms are better understood in terms of string diagrams (Figure 9).

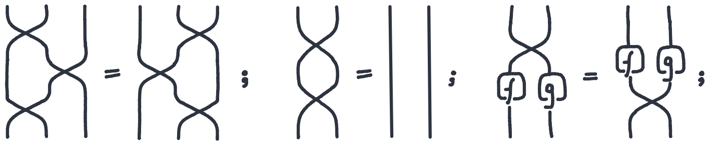


FIGURE 9. Theory of strict symmetric monoidal categories.

However, this is an inefficient syntax: it forces us to explicitly deal with the axioms of the swap. A better syntax would make them transparent, and state that the only thing that matters in a string diagram for symmetric monoidal categories is where the wires are ultimately connected – that is, we only care about the underlying hypergraph. A detailed presentation of the string diagrams of symmetric monoidal categories as hypergraphs is in the work of Bonchi, Sobocinski, Zanasi, and others [BSZ14, BGK$^+$16].

Definition 4.4. A *hypergraph* (V, E) consists of a set of nodes, V, and a set of directed hyperedges E connecting lists of vertices to lists of vertices, that is, $e \colon [v_1, \ldots, v_n] \to [w_1, \ldots w_m]$ for each $e \in E$. We say a hypergraph is acylic if contains no loops.

Definition 4.5. A *hypergraph labelled over a polygraph* $\mathcal{G}$, is a hypergraph (V, E) such that each vertex $v \in V$ is assigned an object of the polygraph, $l(v) \in \mathcal{G}_{obj}$, and each hyperedge $e\colon [v_1, \dots, v_n] \to [w_1, \dots w_m]$ is assigned an edge

$$l(e)\colon l(v_1), \dots, l(v_n) \to l(w_1), \dots, l(w_m),$$

preserving the type of its vertices.

Definition 4.6. A *symmetric string diagram* from $[X_1, \dots, X_n]$ to $[Y_1, \dots, Y_m]$ is an acyclic hypergraph (V, E) labelled by a polygraph $\mathcal{G}$, such that each vertex appears exactly once as an input and exactly once as an output, and endowed with two distinguished unlabelled hyperedges:

$$\text{the input } i\colon [] \to [x_1, \dots, x_n], \text{ and the output } o\colon [y_1, \dots, y_m] \to [],$$

typed by $l(x_1) = X_1, \dots, l(x_n) = X_n$ and $l(y_1) = Y_1, \dots, l(y_m) = Y_m$.

Proposition 4.7. *Symmetric string diagrams over a polygraph $\mathcal{G}$ form a symmetric monoidal category,* $\mathsf{String}_\sigma(\mathcal{G})$.

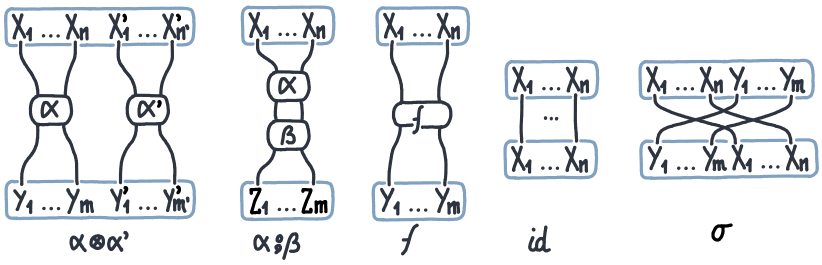


FIGURE 10. Symmetric monoidal category of string diagrams.

PROOF. A summary of the construction is in Figure 10: wires are vertices and hyperedges are nodes. Let us describe the category. The objects are lists of objects of the polygraph. Tensoring of string diagrams of type $[X_1, ..., X_n] \to [Y_1, ..., Y_m]$ and $[X'_1, ..., X'_n] \to [Y'_1, ..., Y'_m]$ is defined by the disjoint union of the hypergraphs – merging input and output edges – into a string diagram $[X_1, ..., X_n, X'_1, ..., X'_n] \to [Y_1, ..., Y_m, Y'_1, ..., Y'_m]$. Composition of string diagrams of type $[X_1, ..., X_n] \to [Y_1, ..., Y_m]$ and $[Y_1, ..., Y_m] \to [Z_1, ..., Z_p]$ is constructed by glueing the vertices along the output edge of the first and the input edge of the second, which disappear producing a string diagram $[X_1, \dots, X_n] \to [Z_1, \dots, Z_p]$. Generators are included as single hyperedges labelled by them. The identity, $[X_1, \dots, X_n] \to [X_1, \dots, X_n]$, consists only of the input and output edges, connected by a list of vertices. Symmetries are defined by twisting the connections of the input and output edges. Finally, we can check that string diagrams satisfy the axioms of symmetric monoidal categories. □

Definition 4.8. A *strict symmetric monoidal functor* between two strict symmetric monoidal categories, $F\colon \mathbb{C} \to \mathbb{D}$, is a strict monoidal functor that moreover preserves symmetries: $F(\sigma)\colon F(X) \otimes F(Y) \to F(Y) \otimes F(X)$ is the symmetry on $F(X)$ and $F(Y)$. Strict symmetric monoidal categories and strict symmetric monoidal functors form a category, $\mathbf{SymMonCat}_{\mathsf{Str}}$.

Proposition 4.9. *The construction of symmetric string diagrams extends to a functor from polygraphs to symmetric monoidal categories,*

$$\mathsf{String}_{\sigma}\colon \mathbf{PolyGraph} \to \mathbf{SymMonCat}_{\mathsf{Str}}.$$

Proposition 4.10. *There exists a forgetful functor from strict symmetric monoidal categories to polygraphs that takes the objects as the vertices of the polygraph and the morphisms as the edges,*

$$\mathsf{Forget}\colon \mathbf{SymMonCat}_{\mathsf{Str}} \to \mathbf{PolyGraph}.$$

THEOREM 4.11. *String diagrams for symmetric monoidal categories form the free strict symmetric monoidal category over a polygraph: there exists an adjunction* $\mathsf{String}_{\sigma} \dashv \mathsf{Forget}$.

PROOF SKETCH. We have already shown that string diagrams form a symmetric monoidal category. It only remains to show that there exists a unique strict symmetric monoidal functor to any strict symmetric monoidal category. The assignment is determined by the fact that each string diagram is constructed from the generators and the operations of a symmetric monoidal category, the difficulty is in showing that this assignment is well-defined. We refer the reader to the work of Bonchi and others [BGK+16]. □

The syntax of symmetric string diagrams as hypergraphs is more efficient: to check equality, only the connectivity of the wires matters, and we no longer need to track the specific blocks forming the diagram.

4.3. Do-Notation.

There is a second practical syntax for symmetric monoidal categories that links string diagrams to programming: Hughes' *arrow do-notation* [Hug00, Pat01]. It comes from *functional programming*, but it is precisely a representation of *imperative programming*. The main idea is that, in a string diagram, we can label the wires by variable names, and simply declare which nodes take which inputs and outputs to reconstruct the string diagram. In a certain sense, this is the graph encoding of a string diagram, but it closely resembles an imperative program.

Example 4.12 (Crema di Mascarpone in Do-notation)**.** Consider the same process for "crema di mascarpone" that we detailed in Section 1.4. This time, we can directly assume that we are in a symmetric monoidal category. The translation of the string diagram of Figure 1 is the following code in Figure 11.

```
cremaDiMascarpone(egg1, egg2, sugar, mascarpone):
  crack(egg1) → (white1, shell1, yolk1)
  crack(egg2) → (white2, shell2, yolk2)
  whisk(white1, white2) → whiskedWhites
  beat(yolk1, yolk2, sugar) → paste
  stir(paste, mascarpone) → thinkPaste
  fold(whiskedWhites, thickPaste) → crema
  return(crema)
```

FIGURE 11. Do-notation recipe for Crema di Mascarpone.

Let us formalize do-notation in the style of a type-theory: we will work with variables, we consider them to be unique and we work implicitly up to α-equivalence (or renaming of variables). Our notion of signature is again that of a polygraph: our basic types will be the objects of the polygraph, and we will have a rule for each one of the generators in the polygraph.

Definition 4.13. A *derivation* in the proof theory of do-notation over a polygraph is defined inductively to be either

(1) a single `return` statement, given by a permutation; or
(2) the application of a generator $f \colon A_0, \dots, A_n \to B_0, \dots, B_m$, given by a choice of generator and an insertion of variables, followed by a derivation.

$$\frac{}{\mathsf{a}_0{:}A_0, \dots, \mathsf{a}_n{:}A_n \gg_\tau () \vdash \mathsf{return}(\mathsf{a}_0, \dots, \mathsf{a}_n) : A_0 \otimes \dots \otimes A_n} \ (\textsc{Return})$$

$$\frac{\mathsf{b}_0{:}B_0, \dots, \mathsf{b}_m{:}B_m, \Gamma \vdash \mathsf{t} : \Delta}{\mathsf{a}_0{:}A_0, \dots, \mathsf{a}_n{:}A_n \gg_\tau \Gamma \vdash \mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_n) \to \mathsf{b}_0, \dots, \mathsf{b}_m \fatsemi \mathsf{t} : \Delta} \ (f)$$

FIGURE 12. Do-notation for symmetric monoidal categories.

Before continuing, then, it is important to understand what an *insertion* is. An insertion captures how many ways we have of inserting some n new terms into a list of m terms. The new n terms can be freely permuted, but the list of m terms must preserve their relative order. This is the combinatorial structure that will track how variables are used in a derivation.

Definition 4.14. We define the family of insertions of n terms into m terms, $\mathsf{Ins}(n, m)$, inductively. There exists a single way of inserting zero terms into any list of terms, $\mathsf{Ins}(0, m) = 1$; inserting $n + 1$ terms into a list of m terms amounts to choosing the position of the first among $m + 1$ possible choices, and inserting the rest of the terms,

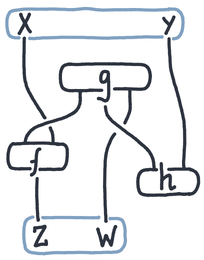
$$\mathsf{Ins}(n + 1, m) = (m + 1) \times \mathsf{Ins}(n, m + 1).$$

As a consequence, the number of possible insertions is $\mathsf{Ins}(n, m) = (m + n)!/m!$.

We write $a_1, \dots, a_n \gg_\tau \Gamma$ to refer to the list resulting from the insertion of the variables $a_1, \dots, a_n$ into the list Γ, of length m, according to the insertion $\tau \in \mathsf{Ins}(n, m)$.

Remark 4.15. Accordingly, the only information that a return statement may carry is that of an insertion $\mathsf{Ins}(n, 0)$, which is equivalently a permutation of the n elements that are being returned. The information carried by a generator statement is not only a generator n-to-m but also an insertion $\mathsf{Ins}(n, \#\Gamma)$ of n variables on the context of the derivation.

Example 4.16. Let us provide an example of the correspondence between the different notations: a Rosetta's stone translating between string diagrams, terms of do-notation, and their corresponding derivations (Figure 13).



$$\begin{array}{c}
\hline
\mathsf{z}{:}Z, \mathsf{w}{:}W \vdash \mathsf{return}(\mathsf{z}, \mathsf{w}) : Z \otimes W \\
\hline
\mathsf{z}{:}Z, \mathsf{b}{:}B, \mathsf{w}{:}W, \mathsf{y}{:}Y \vdash \mathsf{h}(\mathsf{b}, \mathsf{y}) \to ()\,⨟ \\
\qquad\qquad \mathsf{return}(\mathsf{z}, \mathsf{w}) : Z \otimes W \\
\hline
\mathsf{a}{:}A, \mathsf{b}{:}B, \mathsf{w}{:}W, \mathsf{x}{:}X, \mathsf{y}{:}Y \vdash \mathsf{f}(\mathsf{a}, \mathsf{x}) \to \mathsf{z}\,⨟ \\
\qquad\qquad \mathsf{h}(\mathsf{b}, \mathsf{y}) \to ()\,⨟ \\
\qquad\qquad \mathsf{return}(\mathsf{z}, \mathsf{w}) : Z \otimes W \\
\hline
\mathsf{x}{:}X, \mathsf{y}{:}Y \vdash \mathsf{g}() \to (\mathsf{a}, \mathsf{b}, \mathsf{w})\,⨟ \\
\qquad \mathsf{f}(\mathsf{a}, \mathsf{x}) \to \mathsf{z}\,⨟ \\
\qquad \mathsf{h}(\mathsf{b}, \mathsf{y}) \to ()\,⨟ \\
\qquad \mathsf{return}(z, w) : Z \otimes W
\end{array}$$

FIGURE 13. String diagram and derivation of a term.

4.4. Symmetry in Do-notation. At this point, using insertions may seem complicated: why not simply assume an exchange rule that allows us to permute variables freely? The problem we would encounter is that exchanges introduce redundancy: there would be multiple ways of writing the same term, depending on where we place the symmetries (Shulman describes the same problem for a different notation [Shu16]). This is not a catastrophic problem – we could still quotient them out appropriately – but it would make the construction much more complicated than simply dealing with the combinatorial structure of insertions upfront.

The better solution is to have exchange appear as a derived, admissible rule, rather than a primitive.

Proposition 4.17. *Exchange is admissible in Do-notation for symmetric monoidal categories. Any derivation* $\Gamma, \mathsf{x}, \mathsf{y}, \Delta \vdash \mathsf{t} : X$ *admits a derivation* $\Gamma, \mathsf{y}, \mathsf{x}, \Delta \vdash \mathsf{t} : X$*.*

Proof. We proceed by structural induction. The base case is a single return statement, written as $\mathsf{a_0}, \dots, \mathsf{x}, \mathsf{y}, \dots, \mathsf{a_n} \gg_\sigma () \vdash \mathsf{return}(\mathsf{a_0}, \dots, \mathsf{x}, \dots, \mathsf{y}, \dots \mathsf{a_n}) : X$. Permuting x, y in the insertion τ gives us a new insertion τ_{xy} deriving the same statement under a different context

$$\mathsf{a_0}, \dots, \mathsf{y}, \mathsf{x}, \dots, \mathsf{a_n} \gg_{\tau_{\mathsf{xy}}} () \vdash \mathsf{return}(\mathsf{a_0}, \dots, \mathsf{x}, \dots, \mathsf{y}, \dots \mathsf{a_n}) : X.$$

Consider now an application of a generator,

$$\mathsf{a_0}, \dots, \mathsf{a_n} \gg_\tau \Gamma \vdash \mathsf{f}(\mathsf{a_0}, \dots, \mathsf{a_n}) \to \mathsf{b_0}, \dots, \mathsf{b_m} \fatsemi \mathsf{term} : X.$$

There are two possible cases here: (1) if any of x, y is among the inserted variables, $\mathsf{a_0}, \dots, \mathsf{a_n}$, then we may simply exchange them by changing the order in which they are inserted; (2) if not, then $\Gamma = \Gamma', \mathsf{x}, \mathsf{y}, \Delta'$, and we apply the induction hypothesis over the derivation $\Gamma', \mathsf{x}, \mathsf{y}, \Delta' \vdash \mathsf{term} : X$. □

Indeed, this is enough to ensure that terms do correspond to derivations: we may simply write a term and there is a unique way it could have been extracted from the context.

Proposition 4.18. *Do-notation terms in a given context have a unique derivation.*

Proof. By structural induction, a term is either a single return or an application of some generator (f). Any single term $\mathsf{x_0}, \dots, \mathsf{x_n} \vdash \mathsf{return}(\mathsf{a_0}, \dots, \mathsf{a_n})$ has a unique derivation: namely, the one that inserts the a's into the x's permuting them in the only possible order. The insertion τ must be the only one making $\tau(\mathsf{x_i}) = \mathsf{a_i}$.

Consider now an application of a generator, $\Delta \vdash \mathsf{f}(\mathsf{a_0}, \dots, \mathsf{a_n}) \to \mathsf{b_0}, \dots, \mathsf{b_m} \fatsemi \mathsf{t} : X$. We know that it must come from $\Delta = (\mathsf{a_0}, \dots, \mathsf{a_n} \gg_\tau \Gamma)$, and this forces $\Gamma = (\Delta - \{\mathsf{a_0}, \dots, \mathsf{a_n}\})$ and the value of the insertion τ: it is the only one that turns Γ into Δ. Once Γ has been determined, we apply the induction hypothesis to get the derivation of $\mathsf{b_0}, \dots, \mathsf{b_m}, \Gamma \vdash \mathsf{t} : X$. □

4.5. Quotienting Do-notation. Our logic is freely constructed, but it is not yet a logic of monoidal categories: technically, it misses interchange. As it stands, it is actually a logic for symmetric premonoidal categories, which we will study later. We shall only add the following rule – the interchange rule – in order to convert it into a calculus of symmetric monoidal categories.

Interchange rule. Consider a derivation of the following term, where $\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}$ and $\mathsf{c}_0, \dots \mathsf{c}_\mathsf{p}$ are two lists of *distinct* variables, meaning that no b_i appears in c_j – and conversely, no d_i appears in a_j. Then, we can interchange the two first statements of the term.

$$\begin{array}{ll} \Gamma \vdash \mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n}) \to \mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m} \fatsemi & \\ \qquad \mathsf{g}(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \to \mathsf{d}_0, \dots, \mathsf{d}_\mathsf{p} \fatsemi & \\ \qquad \mathsf{term} : \Delta & \end{array} \equiv \begin{array}{ll} \Gamma \vdash \mathsf{g}(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \to \mathsf{d}_0, \dots, \mathsf{d}_\mathsf{p} \fatsemi & \\ \qquad \mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n}) \to \mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m} \fatsemi & \\ \qquad \mathsf{term} : \Delta & \end{array}$$

FIGURE 14. Interchange rule.

Proposition 4.19. *We show this is well defined. Whenever the left hand side of the interchange rule is a valid term, and variables are distinct, the right hand side is also a valid term.*

PROOF. First, we reason that the term on the left-hand side must have a derivation tree of the following form.

$$\dfrac{\dfrac{\mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}, \mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}, \Delta \vdash \mathsf{term} : C}{\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}, (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p} \gg_\rho \Delta) \vdash \begin{array}{l}\mathsf{g}(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \to (\mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}) \fatsemi \\ \mathsf{term} : C\end{array}}}{\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n} \gg_\sigma (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p} \gg_\rho \Delta) \vdash \begin{array}{l}\mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n}) \to (\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}) \fatsemi \\ \mathsf{g}(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \to (\mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}) \fatsemi \\ \mathsf{term} : C\end{array}}$$

We now argue for this. In the second line of the derivation we use that, because of variables being distinct, the only possible context $(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \gg_\tau \Psi$ we can use must factor as $\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}, (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \gg_\rho \Delta$ for some Δ and ρ. We can then check by the form of the rules that the final context Γ must be of the form $\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n} \gg_\sigma (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p} \gg_\rho \Delta)$ for some insertion σ. All this means that the following derivation is also valid.

$$\dfrac{\dfrac{\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}, \mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}, \Delta \vdash \mathsf{term} : X}{\mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}, (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p} \gg \Delta) \vdash \begin{array}{l}\mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n}) \to \mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m} \fatsemi \\ t : X\end{array}}}{\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n} \gg (\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p} \gg \Delta) \vdash \begin{array}{l}\mathsf{g}(\mathsf{c}_0, \dots, \mathsf{c}_\mathsf{p}) \to (\mathsf{d}_0, \dots, \mathsf{d}_\mathsf{q}) \fatsemi \\ \mathsf{f}(\mathsf{a}_0, \dots, \mathsf{a}_\mathsf{n}) \to (\mathsf{b}_0, \dots, \mathsf{b}_\mathsf{m}) \fatsemi \\ \mathsf{term} : X\end{array}}$$

This derivation proves that the right side term is also valid. □

Finally, we can start proving that do-notation terms are a syntax for symmetric monoidal categories: they form the free strict symmetric monoidal category over a signature.

Lemma 4.20. *Do-notation terms over a polygraph $\mathcal{G}$, quotiented by the interchange rule, define a monoidal category,* $\mathsf{Do}(\mathcal{G})$*. This construction induces a functor* $\mathsf{Do}\colon \mathbf{PolyGraph} \to \mathbf{SymMonCat}_{\mathsf{Str}}$.

Proof sketch. We will describe the operations of this strict symmetric monoidal category. Let us start by **composition**: consider derivations $\Gamma \vdash \mathsf{t}\colon B_1 \otimes \ldots \otimes B_m$ and $\mathsf{b_1}{:}B_1, \ldots, \mathsf{b_m}{:}B_m \vdash \mathsf{s}\colon \Psi$. We will construct, by induction over t, a derivation $\Gamma \vdash \mathsf{comp}(t, s)\colon \Psi$, that will define their *composition.* If t is a return statement, then Γ is a permutation of $\mathsf{b_1}{:}B_1, \ldots, \mathsf{b_m}{:}B_m$ and, by exchange, we obtain $\Gamma \vdash s\colon \Psi$ and we define it as the composition. Whenever t consists of a generator followed by a derivation, $\Gamma \vdash t\colon \Psi$ must be of the form

$$(\mathsf{x_1}, \ldots, \mathsf{x_n}) \gg \Gamma' \vdash \mathsf{f}(\mathsf{x_1}, \ldots, \mathsf{x_n}) \to \mathsf{y_1}, \ldots, \mathsf{y_m} \fatsemi \mathsf{t'} : \Psi,$$

and we define $\mathsf{comp}(t, s)$ to mean $\mathsf{f}(\mathsf{x_1}, \ldots, \mathsf{x_n}) \to \mathsf{y_1}, \ldots, \mathsf{y_m} \fatsemi \mathsf{comp}(\mathsf{t'}, \mathsf{s})$, using the $y_1, \ldots, y_m, \Gamma' \vdash t'\colon B_1 \otimes \ldots \otimes B_m$ we just obtained. Put simply, we have removed the last return statement from one of the derivations, taking care of permutations, and then concatenated both (see Figure 15).

Let us now define **tensoring**: consider derivations $\Gamma \vdash t\colon \Delta$ and $\Gamma' \vdash t'\colon \Delta'$. We start by noticing that, if we have a derivation $\Gamma \vdash \mathsf{t}\colon \Delta$, we can construct, by induction, a derivation $\Gamma, \mathsf{z}{:}Z \vdash \mathsf{t}_z : \Delta \otimes Z$ for any $\mathsf{z}{:}Z$: the return case consists of adding an extra variable to the permutation, the generator case is not affected by the presence of an extra variable. Repeating this reasoning and using exchange, we can obtain terms $\Gamma, \Gamma' \vdash \mathsf{t}_{\Gamma'}\colon \Delta \otimes \Gamma'$ and $\Delta, \Gamma' \vdash \mathsf{t'}_{\Delta}\colon \Delta \otimes \Delta'$ that we can compose into $\Gamma, \Gamma' \vdash \mathsf{comp}(\mathsf{t}_{\Gamma'}, \mathsf{t'}_{\Delta'})\colon \Delta \otimes \Delta'$. The order of composition does not matter because of the *interchange law.* Put simply, we write the two terms one after the other, taking care not to mix the variables (see Figure 15, for a graphical intuition).

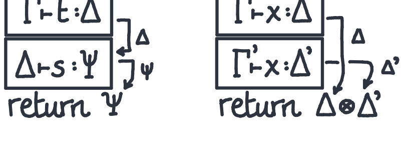


Figure 15. Composition and tensoring in do-notation.

Finally, we must define **identities**, **generators** and **symmetries**. These are the following three terms.

$$\begin{aligned} \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n} &\vdash \mathsf{return}(\mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}) \\ \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n} &\vdash \mathsf{f}(\mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}) \to (\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{m}) \mathbin{⨾} \mathsf{return}(\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{m}) \\ \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}, \mathsf{b}_1, \dots \mathsf{b}_\mathsf{m} &\vdash \mathsf{return}(\mathsf{b}_1, \dots, \mathsf{b}_\mathsf{m}, \mathsf{a}_1, \dots, \mathsf{a}_\mathsf{n}) \end{aligned}$$

We can check that these operations define a category and a monoidal category. The interchange law of monoidal categories is extracted from the interchange law that we imposed in do-notation. □

THEOREM 4.21. *There is an adjunction between polygraphs and the category of strict symmetric monoidal categories given by do-notation terms,* $\mathsf{Do} \dashv \mathsf{Forget}$. *Moreover, do-notation terms and string diagrams are naturally isomorphic.*

PROOF SKETCH. We have already proven that do-notation constructs a symmetric monoidal category over a polygraph. We need to show that there is a unique map out of the category constructed by do-notation that commutes with any assignment of the polygraph to a monoidal category.

The first part would be to prove the initiality of the syntax. We can do so by structural induction: any term is either a return statement or a composition of a generator with a symmetry and a term. In the first case, the symmetry determined by the return statement must be mapped to the corresponding symmetry in any symmetric monoidal category; in the second case, the image of the generator is determined, and the rest of the term is determined by structural induction. In conclusion, the image of any do-notation term is determined in any symmetric monoidal category over which we map the polygraph of generators.

The core of the proof is in showing that this unique possible assignment is well-defined: the only equality imposed on do-notation is the interchange law, but this law corresponds, under the assignment, to the interchange law of symmetric monoidal categories.

Finally, both functors,

$$\mathsf{Do}, \mathsf{String}_\sigma \colon \mathbf{PolyGraph} \to \mathbf{SymMonCat}_{\mathsf{Str}}$$

have been found to be left adjoints to the same forgetful functor. This implies they are isomorphic functors. □

4.6. Example: the XOR Variable Swap. Let us provide an example of the formal usage of both do-notation and string diagrams, by reasoning about a simple process. In imperative programming languages, swapping the value of two variables usually requires a third temporary variable. However, the XOR variable swap algorithm uses the properties of the exclusive-or operation (XOR, $\oplus$) to exchange variables without needing a temporary variable. Let $\mathsf{xor}(\mathsf{x}, \mathsf{y}) = (\mathsf{x} \oplus \mathsf{y}, \mathsf{y})$. The code is in Figure 16.

```
xorExchange(x,y):
  xor(x,y) → (x,y)
  xor(y,x) → (y,x)
  xor(x,y) → (x,y)
  return (x,y)
```

FIGURE 16. XOR variable exchange.

The property that makes this algorithm possible is the *nilpotency* of the XOR operation: $x \oplus x = 0$ for any n-bit word $x \in 2^n$. This means that we can prove the correctness of the XOR variable exchange in the abstract setting of nilpotent bialgebras. In fact, consider a polygraph $\mathcal{X}$ with a single object and the generators depicted in Figure 17.

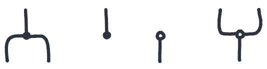


FIGURE 17. Signature for a bialgebra.

We now want to impose a set of equations, $E \subseteq \mathsf{String}(\mathcal{X}) \times \mathsf{String}(\mathcal{X})$, on top of this signature. This can also be done via the adjunction: the equations give two maps $E \to \mathsf{Forget}(\mathsf{String}(\mathcal{X}))$, or equivalently, $\mathsf{String}(E) \to \mathsf{String}(\mathcal{X})$. The coequalizer of the latter two describes the universal monoidal category with some generators and satisfying some equations. Back to our example, the theory of a nilpotent bialgebra satisfies the following equations in Figure 18.

Given any nilpotent bialgebra in any strict symmetric monoidal category, there exists a unique monoidal functor from the string diagrams quotiented by these equations to that signature.

Proposition 4.22. *Let a nilpotent bialgebra in a symmetric monoidal category. The XOR variable exchange algorithm is equal to the swap morphism.*

PROOF. In the theory of nilpotent bialgebras over a symmetric monoidal category, the following equation in Figure 19 holds.

The left hand side represents the XOR variable exchange while the right hand side represents swapping the contents of two variables. We have shown both are equal. □

4.7. Bibliography. Symmetric monoidal categories and their hexagon coherence equations were already stated by MacLane [Mac63, ML71], Bénabou

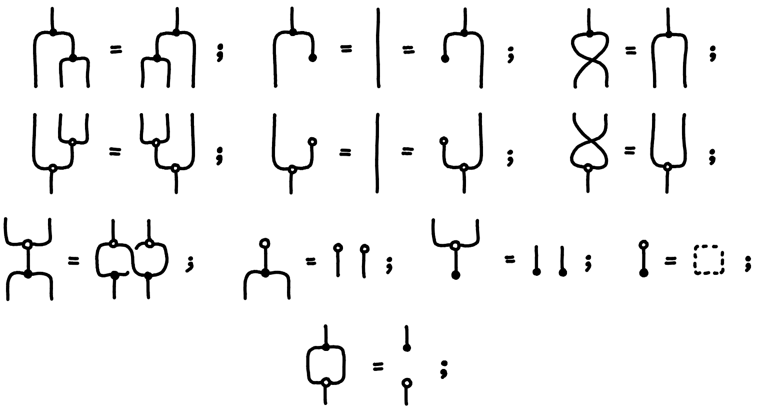


FIGURE 18. Theory of a nilpotent bialgebra.

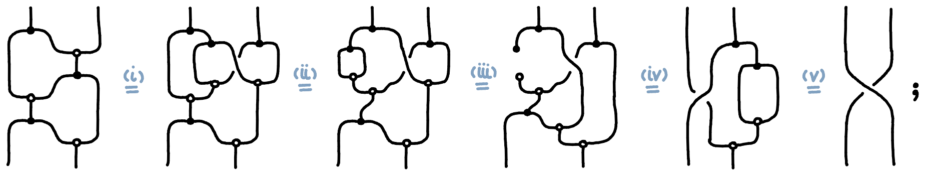


FIGURE 19. XOR variable exchange.

defined *commutations* on a monoidal category and an abstract notion of commutative monoid [Bén68]. String diagrams for symmetric monoidal categories are already described by Joyal and Street [JS91, Sel10]; we follow the representation in terms of hypergraphs by Bonchi, Gadducci, Kissinger, Sobocinski and Zanasi [BGK$^+$16]. The XOR example was known to Erbele [BE14] and Sobocinski. The idea of using a Rosetta stone to translate between categories and logics comes from Baez and Stay [BS10].

Do-notation comes from the Haskell programming language [HJW$^+$92], where it takes semantics in a strong promonad [Hug00, HJ06] (also known as an *arrow* [Pat03]). We have studied here an adaptation to the monoidal setting. Our presentation of do-notation follows the style of Shulman's categorical logic [Shu16].

5. Cartesianity: Determinism and Totality

Our process theories are, by default, *linear on resources*: every variable must be used exactly once. This may seem like a limitation, but it is a more general case that can be particularized into the classical case when necessary: we say that a process theory – a monoidal category – is *classical* or *cartesian* whenever it has processes representing copying and discarding and satisfying suitable axioms.

Non-classical theories can become so in two ways: either because they do not allow copying, or because they do not allow discarding. Theories without copying model stochasticity and non-determinism: running a computation twice is different from just running it once and assuming its result will be the same next time. Theories without discarding model partiality: even if we do not care about the result, we cannot assume anymore that any computation will succed.

5.1. Cartesian Monoidal Categories. Cartesian monoidal categories give a universal property to their tensor: the tensor of two objects, $A \times B$, is such that pair of maps to A and B are in precise correspondence to single map to $A \times B$. This universal property, in some sense, ensures that the tensor contains nothing more and nothing less than its two constituent parts; this is what characterizes classical theories.

Definition 5.1. *Cartesian monoidal categories* are monoidal categories, $(\mathbb{C}, \times, 1)$, such that

(1) each tensor, $A \times B$, is endowed with projections, $\pi_1 \colon A \times B \to A$ and $\pi_2 \colon A \times B \to B$, that make it a product: for each object $X \in \mathbb{C}_{obj}$ and any pair of morphisms, $f \colon X \to A$ and $g \colon X \to B$, there exists a unique $\langle f, g \rangle \colon X \to A \times B$ such that
$$\langle f, g \rangle \mathbin{⨟} \pi_1 = f \quad \text{and} \quad \langle f, g \rangle \mathbin{⨟} \pi_2 = g;$$
(2) the unit, 1, is terminal: for each object $X \in \mathbb{C}_{obj}$ there exists a unique morphism $\pi \colon X \to 1$.

Fox's theorem is a characterisation of classical theories, cartesian monoidal categories, in terms of the existence of a uniform cocommutative comonoid structure (copy and delete) on all objects of a monoidal category.

Theorem 5.2 (Fox, [Fox76]). *A symmetric monoidal category $(\mathbb{C}, \otimes, I)$ is cartesian monoidal if and only if every object $X \in \mathbb{C}$ has a commutative comonoid structure $(X, \varepsilon_X, \delta_X)$, every morphism of the category $f \colon X \to Y$ is a comonoid homomorphism, and this structure is uniform across the monoidal category, meaning that $\varepsilon_{X \otimes Y} = \varepsilon_X \otimes \varepsilon_Y$, that $\varepsilon_I = \mathrm{id}$, that $\delta_I = \mathrm{id}$ and that $\delta_{X \otimes Y} = (\delta_X \otimes \delta_Y) \mathbin{⨟} (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.*

Fox's characterization has a direct translation to string diagrams: the first conditions impose a natural commutative comonoid structure on each generator

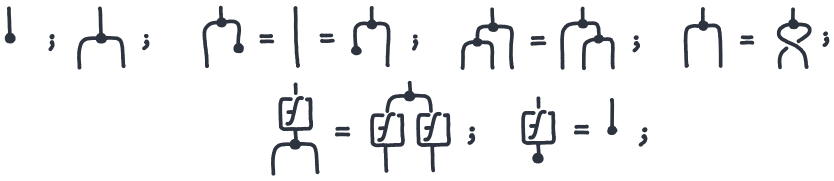


FIGURE 20. Theory of cartesian categories.

(Figure 20); the last conditions state that the structure on all the objects follows from that of the generators.

We can add a slight improvement. Most sources ask the comonoid structure in Fox's theorem (Theorem 5.2) to be cocommutative [Fox76, FS19]. However, cocommutativity and coassociativity of the comonoid structure are implied by the fact that the structure is uniform and natural. We present an original refined version of Fox's theorem.

THEOREM 5.3 (Refined Fox's theorem). *A symmetric monoidal category, $(\mathbb{C}, \otimes, I)$, is cartesian monoidal if and only if every object $X \in \mathbb{C}$ has a counital comagma structure $(X, \varepsilon_X, \delta_X)$, or $(X, \bullet_X, \curlywedge_X)$, every morphism of the category $f : X \to Y$ is a comagma homomorphism, and this structure is uniform across the monoidal category: meaning that $\varepsilon_{X \otimes Y} = \varepsilon_X \otimes \varepsilon_Y$, $\varepsilon_I = \mathrm{id}$, $\delta_I = \mathrm{id}$ and $\delta_{X \otimes Y} = (\delta_X \otimes \delta_Y); (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.*

PROOF. We prove that such a comagma structure is necessarily coassociative and cocommutative. Note that any comagma homomorphism $f : A \to B$ must satisfy $\delta_A \mathbin{⨾} (f \otimes f) = f \mathbin{⨾} \delta_B$. In particular, $\delta_X : X \to X \otimes X$ must itself be a comagma homomorphism (see Figure 25), meaning that

$$(1) \qquad \delta_X \mathbin{⨾} (\delta_X \otimes \delta_X) = \delta_X \mathbin{⨾} \delta_{X \otimes X} = \delta_X \mathbin{⨾} (\delta_X \otimes \delta_X) \mathbin{⨾} (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id}),$$

where the second equality follows by uniformity.

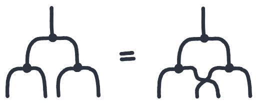


FIGURE 21. Comultiplication is a comagma homomorphism.

Now, we prove cocommutativity (Figure 22): composing both sides of Equation (1) with $(\varepsilon_X \otimes \mathrm{id} \otimes \mathrm{id} \otimes \varepsilon_X)$ discards the two external outputs and gives $\delta_X = \delta_X \mathbin{⨾} \sigma_X$. Then, we prove coassociativity (Figure 23): composing both sides

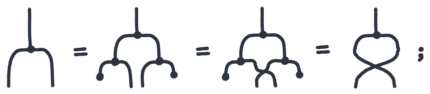


FIGURE 22. Cocommutativity

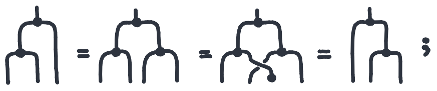


FIGURE 23. Coassociativity

of Equation (1) with $(\mathrm{id} \otimes \varepsilon_X \otimes \mathrm{id} \otimes \mathrm{id})$ discards one of the middle outputs and gives $\delta_X ⨟ (\mathrm{id} \otimes \delta_X) = \delta_X ⨟ (\delta_X \otimes \mathrm{id})$.

A coassociative and cocommutative comagma is a cocommutative comonoid. We can then apply the classical form of Fox's theorem (Theorem 5.2). □

5.2. Partial Markov Categories. If process theories were all cartesian, we could use the commutative comonoid structure on every object to simplify calculations. However, most of the theories that pose a challenge to computer science (like stochastic processes, partial processes, or quantum maps) are not cartesian. The rest of this thesis will not assume cartesianity: let us give a good example and motivation for doing this.

Cartesianity in a category with copy and discard processes can be divided in two concepts: *determinism* and *totality*. All pure functions, for instance, are deterministic and total, but stochastic functions are not deterministic (tossing a coin twice is different from tossing it once and copying the result twice), and partially computable functions are not total (because even if we do not care about the output, they could diverge and make the whole process wait).

Definition 5.4. In a symmetric monoidal category with uniform commutative comonoids, $(\mathbb{C}, \otimes, I, \curlywedge, \multimap, \sigma)$, a morphism $f \colon X \to Y$ is *deterministic* if it can be copied, $f ⨟ \curlywedge_Y = \curlywedge_X ⨟ (f \otimes f)$, and it is *total* (or *causal*) if it can be discarded, $f ⨟ \multimap_Y = \multimap_X$. Moreover, we say it is *quasitotal* (or *quasicausal*), if it can be copied on the side and discarded, $f = \curlywedge_X ⨟ ((f ⨟ \multimap_Y) \otimes f)$. See Figure 24.

Let us provide a single theory where these three assumptions fail: the theory of discrete partial Markov categories [LR23], which we will use to study partial stochastic functions. Apart from copy $(\curlywedge) \colon X \to X \otimes X$ and discard $(\multimap) \colon X \to I$, we also consider a comparator morphism $(\curlyvee) \colon X \otimes X \to X$. Copying, discarding and comparing interact as a *partial Frobenius algebra* [DLLNS21].

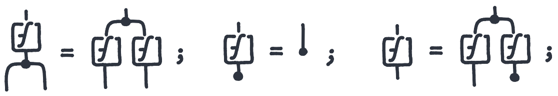


FIGURE 24. Deterministic, total, and quasitotal morphisms.

Definition 5.5. A *discrete partial Markov category* is is a symmetric monoidal category $(\mathbb{C}, \otimes, I)$ such that every object has a partial Frobenius monoid structure $(\curlywedge, \bullet, \curlyvee)$ that satisfies the axioms in Figure 25 and uniformity, meaning that

(1) $(\curlywedge)_{X \otimes Y} = (\curlywedge_X \otimes \curlywedge_Y) ⨾ (\mathrm{id}_X \otimes \sigma \otimes \mathrm{id}_Y)$ and $(\curlywedge)_I = \mathrm{id}$;
(2) $(\curlyvee)_{X \otimes Y} = (\mathrm{id}_X \otimes \sigma \otimes \mathrm{id}_Y) ⨾ (\curlyvee_X \otimes \curlyvee_Y)$ and $(\curlyvee)_I = \mathrm{id}$;
(3) $(\bullet)_{X \otimes Y} = (\bullet_X \otimes \bullet_Y)$ and $(\bullet)_I = \mathrm{id}$.

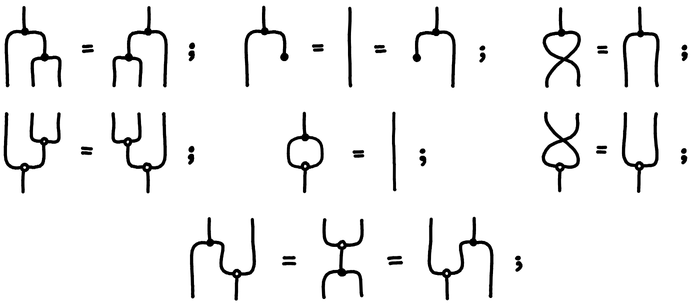


FIGURE 25. Theory of a partial Frobenius algebra.

Proposition 5.6. *A subdistribution on a set X is a function $d \colon X \to [0, 1]$ that is non-zero on a finite number of elements and that adds up to less or equal than one,*

$$\sum_{d(x)>0} d(x) \leq 1.$$

Subdistributions form a monad, and the Kleisi category of the subdistribution monad forms a discrete partial Markov category [LR23].

PROOF. Let us write $\mathbf{D}_{\leq 1}(X)$ for the set of subdistributions over a set. We claim that this extends to a monad in the category of sets and functions $\mathbf{D}_{\leq 1} \colon \mathbf{Set} \to \mathbf{Set}$. The multiplication $\mu \colon \mathbf{D}_{\leq 1}(\mathbf{D}_{\leq 1}(X)) \to \mathbf{D}_{\leq 1}(X)$ is defined by $\mu(\psi)(x) = \sum_{\psi(d)>0} d(x)$, and the unit $\eta \colon X \to \mathbf{D}_{\leq 1}(X)$ is defined by the

Dirac's delta, $\eta(x_0)(x) = [x = x_0]$, which is valued to 1 whenever $x = x_0$ and is valued to 0 otherwise.

The Kleisli category of this monad has morphisms the *partial stochastic channels* $f \colon X \to \mathbf{D}_{\leq 1}(Y)$. We write $f(y|x) \in [0,1]$ for the value of $f(x)$ on the input $y \in Y$, capturing the usual notation for conditionals in probability. Under this notation, composition on the Kleisli category becomes

$$(f \mathbin{⨾} g)(z|x) = \sum_{y \in Y} f(y|x) \cdot g(z|y).$$

While tensoring is $(f \otimes f')(y, y'|x, x') = f(y|x) \cdot f'(y'|x')$. The copy morphism is defined by $(\curlywedge)(x, y|z) = [x = y = z]$; the discard morphism is defined by (⫯)$(|x) = 1$; and the comparator is given by $(\curlyvee)(x|y, z) = [x = y = z]$. It is direct to check that these satisfy the axioms of a partial Frobenius algebra. □

Remark 5.7 (Effect algebras)**.** The set of partial stochastic channels between two sets, $X \to \mathbf{D}_{\leq 1}(Y)$, forms a particular algebraic structure known as an *effect module.*

An *effect algebra* [FB94, Jac15, vdW21] is a set E with a partial binary operation $(\oplus) \colon E \times E \to E$, a unary operation $(\bullet)^{\perp} \colon E \to E$, and a constant $0 \in E$. We write $x \perp y$ whenever $x \oplus y$ is well-defined and we write $1 = 0^{\perp}$. The effect algebra must satisfy

(1) Commutativity, $x \oplus y = y \oplus x$, where $x \perp y$ implies $y \perp x$.
(2) Unitality, $x \oplus 0 = x = 0 \oplus x$, where $x \perp 0$.
(3) Associativity, $x \oplus (y \oplus z) = (x \oplus y) \oplus z$, where having both $y \perp z$ and $x \perp (y \oplus z)$ implies both $x \perp y$ and $(x \oplus y) \perp z$.
(4) Complementarity, $x^{\perp}$ is unique such that $x^{\perp} \oplus x = 1$ and $x \perp x^{\perp}$.

An effect algebra homomorphism $f \colon E \to F$ must satisfy $f(1) = 1$, and $x \perp y$ must imply $f(x) \perp f(y)$, with $f(x \oplus y) = f(x) \oplus f(y)$. Effect algebras form a symmetric monoidal category.

The unit interval, $[0,1]$, forms an effect algebra with the binary sum (whenever it is contained on the interval), the unary complement $x^{\perp} = 1 - x$, and the zero; moreover, the unit interval $[0,1]$ with multiplication forms a monoid in the monoidal category of effect algebras.

Partial stochastic channels also form an effect algebra, with the same pointwise operations, but moreover, we can multiply them by a 'scalar' from the unit interval $[0,1]$: they form a module in the monoidal category of effect algebras. The structure of effect module interplays well with composition and tensoring of partial stochastic channels; so we will employ it later in Section 5.5.

Remark 5.8 (Bayesian inversions)**.** The *Bayesian inversion* of a stochastic channel $g \colon X \to Y$ with respect to a distribution $f \colon I \to X$ is the stochastic channel

$g_f^\dagger \colon Y \to X$ classically defined by

$$g_f^\dagger(x|y) = \frac{g(y|x) \cdot f(x)}{\sum_{x_\bullet \in X} g(y|x_\bullet) \cdot f(x_\bullet)}.$$

Bayesian inversions can be defined synthetically in any partial Markov category [CJ19]. The Bayesian inversion of a morphism $g \colon X \to Y$ with respect to $f \colon I \to X$ is a morphism $g_f^\dagger \colon Y \to X$ satisfying the equation in Figure 26, which translates to the above when the partial Markov category is that of subdistributions.

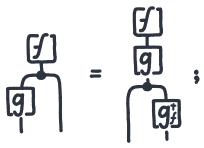


Figure 26. Bayesian inversion.

The definition of a *partial Markov category* includes asking a quasitotal Bayesian inversion for every morphism, with respect to any other morphism: this is the notion of *quasitotal conditional* [LR23], which we will not need in this thesis.

Finally, let us justify that this process theory is enough to capture many of the features of classical probability theory. We will prove a synthetic version of Bayes theorem using the syntax of discrete partial Markov categories. The classical Bayes theorem prescribes that, after observing the output of a prior distribution through a channel, we should update our posterior distribution to be the Bayesian inversion of the channel with respect to the prior distribution, evaluated on the observation.

Theorem 5.9 (Synthetic Bayes' Theorem). *In a discrete partial Markov category, observing a deterministic $x \colon I \to X$ from a prior distribution $f \colon I \to A$ through a channel $g \colon A \to X$ is the same, up to scalar, as the Bayesian inversion evaluated on the observation, $g_f^\dagger(x) \colon I \to A$.*

Proof. We employ string diagrams (Figure 27). Equalities follow from *(i)* the definition of Bayesian inversion, *(ii)* the partial Frobenius axioms, and *(iii)* determinism of y. □

5.3. Bibliography. Fox's theorem, in its original formulation, is the construction of a right adjoint to the forgetful functor from cartesian monoidal categories to Monoidal categories. This right adjoint is given by the category of

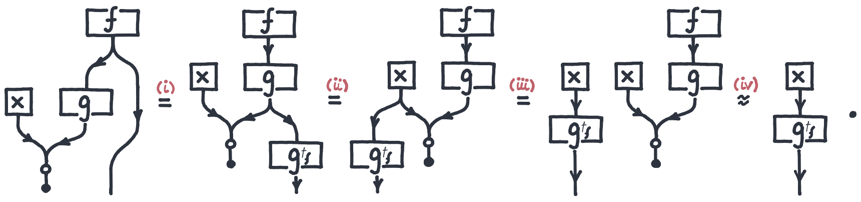


FIGURE 27. Bayes theorem.

cocommutative comonoids over a monoidal category [Fox76]. The version here presented is esentially equivalent; in fact, it is called "Fox's theorem" in the work of Bonchi, Seeber and Sobocinski [BSS18].

The categorical approach to probability theory based on Markov categories is due to Fritz [Fri20] and prior work of Cho and Jacobs [CJ19]. Multiple results of classical probability theory have been adapted to the framework of Markov categories by multiple authors [FR20, FPR21, FP19, FGP21]. Markov categories have been further applied for formalising Bayes networks and other kinds of probabilistic and causal reasoning in categorical terms [Fon13, JZ20, JKZ21]. Their partial counterpart and the application to decision theory was introduced in joint work with Di Lavore [LR23]. Effect algebras are due to Foulis and Bennett [FB94]; Jacobs employed them for a probabilistic categorical logic [Jac15]; van de Wetering [vdW21] characterized the unit interval in terms of effect algebras.

6. Premonoidal Categories

It might seem that monoidal categories are limited to pure imperative programming without effects. After all, the *interchange law* seems to imply that the order in which two independent are executed does not matter. This is true, but again, category theory has a solution for us: premonoidal categories.

Category theory has two successful applications that are rarely combined: monoidal string diagrams [JS91] and programming semantics [Mog91]. We use string diagrams to talk about quantum transformations [AC09], relational queries [BSS18], and even computability [Pav13]; at the same time, proof nets and the geometry of interaction [Gir89, BCST96] have been widely applied in computer science [AHS02, HMH14]. On the other hand, we traditionally use monads and comonads, Kleisli categories and premonoidal categories to explain effectful functional programming [Hug00, JHH09, Mog91, PT99, UV08]. Even if we traditionally employ Freyd categories with a cartesian base [Pow02], we can also consider non-cartesian Freyd categories [SL13], which we call *effectful categories*.

This section introduces premonoidal categories and effectful categories. The next section will study their string diagrams in terms of monoidal categories, reducing them to a particular consideration on top of the theory of monoidal categories.

6.1. Premonoidal Categories. Premonoidal categories are monoidal categories without the *interchange law*, $(f \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes g) \neq (\mathrm{id} \otimes g) ⨟ (f \otimes \mathrm{id})$. This means that we cannot tensor any two arbitrary morphisms, $(f \otimes g)$, without explicitly stating which one is to be composed first, $(f \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes g)$ or $(\mathrm{id} \otimes g) ⨟ (f \otimes \mathrm{id})$, and the two compositions are not equivalent (Figure 28).

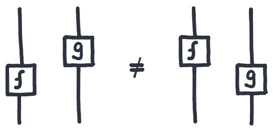


FIGURE 28. The interchange law does not hold in a premonoidal category.

In technical terms, the tensor of a premonoidal category $(\otimes) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is not a functor, but only what is called a *sesquifunctor*: independently functorial in each variable. Tensoring with any identity is itself a functor $(\bullet \otimes \mathrm{id}) \colon \mathbb{C} \to \mathbb{C}$, but there is no functor $(\bullet \otimes \bullet) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$.

A practical motivation for dropping the interchange law can be found when describing transformations that affect a global state. These effectful processes should not interchange in general, because the order in which we modify the global state is meaningful. For instance, in the Kleisli category of the *writer monad*, $(\Sigma^* \times \bullet) \colon \mathbf{Set} \to \mathbf{Set}$ for some alphabet $\Sigma \in \mathbf{Set}$, we can consider the function $\mathsf{print} \colon \Sigma^* \to \Sigma^* \times 1$. The order in which we "print" does matter (Figure 29).

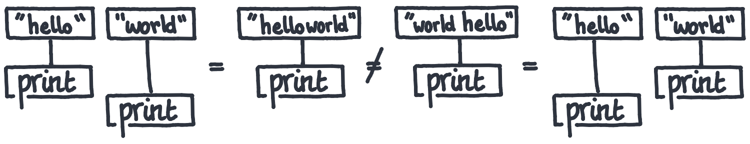


FIGURE 29. Writing does not interchange.

Not surprisingly, the paradigmatic examples of premonoidal categories are the Kleisli categories of Set-based monads $T \colon \mathbf{Set} \to \mathbf{Set}$ (more generally, of strong monads), which fail to be monoidal unless the monad itself is commutative [Gui80, PR97, PT99, Hed19]. Intuitively, the morphisms are "effectful", and these effects do not always commute.

However, we may still want to allow some morphisms to interchange. For instance, apart from asking the same associators and unitors of monoidal categories to exist, we ask them to be *central*: which means that they interchange with any other morphism. This notion of centrality forces us to write the definition of premonoidal category in two different steps: first, we introduce the minimal setting in which centrality can be considered (*binoidal categories* [PT99]) and then we use that setting to bootstrap the full definition of premonoidal category with central coherence morphisms.

Definition 6.1 (Binoidal category)**.** A *binoidal category* is a category $\mathbb{C}$ endowed with an object $I \in \mathbb{C}$ and an object $A \otimes B$ for each $A \in \mathbb{C}$ and $B \in \mathbb{C}$. There are functors $(A \otimes \bullet) \colon \mathbb{C} \to \mathbb{C}$, and $(\bullet \otimes B) \colon \mathbb{C} \to \mathbb{C}$ that coincide on $(A \otimes B)$. Note that $(\bullet \otimes \bullet)$ is not being defined as a functor.

Again, this means that we can tensor with identities (whiskering), functorially; but we cannot tensor two arbitrary morphisms: the interchange law stops being true in general. The *centre*, $\mathcal{Z}(\mathbb{C})$, is the wide subcategory of morphisms that do satisfy the interchange law with any other morphism. That is, $f \colon A \to B$ is *central* if, for each $g \colon A' \to B'$,

$$(f \otimes \mathrm{id}_{A'}) ⨾ (\mathrm{id}_B \otimes g) = (\mathrm{id}_A \otimes g) ⨾ (f \otimes \mathrm{id}_{B'}), \text{ and}$$
$$(\mathrm{id}_{A'} \otimes f) ⨾ (g \otimes \mathrm{id}_B) = (g \otimes \mathrm{id}_A) ⨾ (\mathrm{id}_{B'} \otimes f).$$

Definition 6.2. A *premonoidal category* is a noidal category $(\mathbb{C}, \otimes, I)$ together with the following coherence isomorphisms $\alpha_{A,B,C} \colon A \otimes (B \otimes C) \to (A \otimes B) \otimes C$,

$\rho_A \colon A \otimes I \to A$ and $\lambda_A \colon I \otimes A \to A$ which are central, natural *separately at each given component*, and satisfy the pentagon and triangle equations.

A premonoidal category is *strict* when these coherence morphisms are identities. A premonoidal category is moreover *symmetric* when it is endowed with a coherence isomorphism $\sigma_{A,B} \colon A \otimes B \to B \otimes A$ that is central and natural at each given component and satisfies the symmetry condition and hexagon equations.

Remark 6.3. The coherence theorem of monoidal categories still holds for premonoidal categories: every premonoidal is equivalent to a strict one. We will construct the free strict premonoidal category using string diagrams. However, the usual string diagrams for monoidal categories need to be restricted: in premonoidal categories, we cannot consider two morphisms in parallel unless any of the two is *central.*

6.2. Effectful and Freyd Categories. Premonoidal categories immediately present a problem: what are the premonoidal functors? If we want them to compose, they should preserve the centrality of the coherence morphisms (so that the central coherence morphisms of $F \mathbin{⨟} G$ are these of F after applying G), but naively asking them to preserve all central morphisms rules out important examples [SL13]. The solution is to explicitly choose some central morphisms that represent "pure" computations. These do not need to form the whole centre: it could be that some morphisms considered *effectful* just "happen" to fall in the centre of the category, while we do not ask our functors to preserve them. This is the well-studied notion of a *non-cartesian Freyd category*, which we shorten to *effectful monoidal category* or *effectful category.*

Effectful categories are premonoidal categories endowed with a chosen family of central morphisms. These central morphisms are called pure morphisms, contrasting with the general, non-central, morphisms that fall outside this family, which we call effectful.

Definition 6.4. An *effectful category* is an identity-on-objects functor $\mathbb{V} \to \mathbb{C}$ from a monoidal category $\mathbb{V}$ (the pure morphisms, or "values") to a premonoidal category $\mathbb{C}$ (the effectful morphisms, or "computations"), that strictly preserves all of the premonoidal structure and whose image is central. It is *strict* when both are. A *Freyd category* [Lev22] is an effectful category where the pure morphisms form a cartesian monoidal category.

Effectful categories solve the problem of defining premonoidal functors: a functor between effectful categories needs to preserve only the pure morphisms. We are not losing expressivity: premonoidal categories are effectful with their centre, $\mathcal{Z}(\mathbb{C}) \to \mathbb{C}$. From now on, we study effectful categories.

Definition 6.5 (Effectful functor)**.** Let $\mathbb{V} \to \mathbb{C}$ and $\mathbb{W} \to \mathbb{D}$ be effectful categories. An *effectful functor* is a quadruple $(F, F_0, \varepsilon, \mu)$ consisting of a functor

$F: \mathbb{C} \to \mathbb{D}$ and a functor $F_0: \mathbb{V} \to \mathbb{W}$ making the square commute, and two natural and pure isomorphisms $\varepsilon: I' \cong F(I)$ and $\mu: F(A \otimes B) \cong F(A) \otimes F(B)$ such that they make F_0 a monoidal functor. It is *strict* if these are identities.

When drawing string diagrams in an effectful category, we shall use two different colours to declare if we are depicting either a value or a computation (Figure 30).

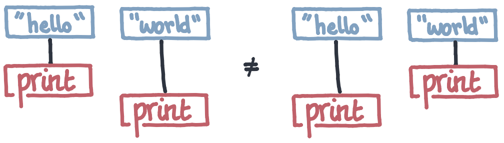


Figure 30. "Hello world" is not "world hello".

Here, the values "hello" and "world" satisfy the interchange law as in an ordinary monoidal category. However, the effectful computation "print" does not need to satisfy the interchange law. String diagrams like these can be found in the work of Alan Jeffrey [Jef97b]. Jeffrey presents a clever mechanism to graphically depict the failure of interchange: all effectful morphisms need to have a control wire as an input and output. This control wire needs to be passed around to all the computations in order, and it prevents them from interchanging.

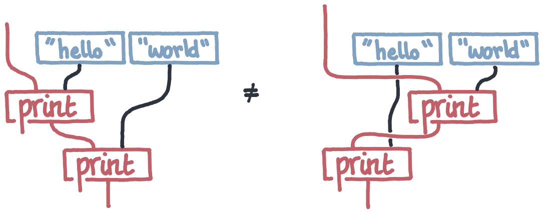


Figure 31. An extra wire prevents interchange.

Our interpretation of monoidal categories is as theories of resources. We can interpret premonoidal categories as monoidal categories with an extra resource – the "runtime" – that needs to be passed to all computations. The next section promotes Jeffrey's observation into a theorem.

Remark 6.6. After the next section, which reduces premonoidal categories to monoidal categories, the rest of this thesis deals mostly with monoidal categories. Why not study premonoidal categories instead of just monoidal categories? Premonoidal categories should be more general: they do not assume the interchange law, which is false in stateful systems. However, we give an argument for studying only monoidal categories in the next section: any premonoidal category can be reinterpreted as a monoidal category carrying an extra resource (an extra wire) representing the global state. We do not need to write a new theory of premonoidal categories: premonoidal categories are already monoidal categories, in which one wire is hidden. The following section makes this idea formal.

6.3. Bibliography. This chapter follows closely the first part of "Promonads and String Diagrams for Effectful Categories", by this author [Rom22].

Effectful categories are the monoidal counterpart of a well-known notion: that of "Freyd categories". The name "Freyd category" sometimes assumes cartesianity of the pure morphisms, but it is also used for the general case; choosing to call "effectful categories" to the general case and reserving the name "Freyd categories" for the cartesian ones avoids this clash of nomenclature. There exists also the more fine-grained notion of "Cartesian effect category" [DDR11], which generalizes Freyd categories and may further justify calling "effectful category" to the general case.

7. String Diagrams for Premonoidal Categories

Premonoidal categories give us a generalization of monoidal categories accounting for effectful computation but, at the same time, they do not need us to change our syntax yet. String diagrams for premonoidal categories can be reduced to string diagrams for monoidal categories: they rely on the fact that the morphisms of the monoidal category freely generated over a polygraph of generators are string diagrams on these generators, quotiented by topological deformations, as we saw in Section 1.3.

7.1. Effectful Polygraphs. We justify string diagrams for premonoidal categories by proving that the freely generated effectful category over a pair of polygraphs (for pure and effectful generators, respectively) can be constructed as the freely generated monoidal category over a particular polygraph that includes an extra wire. In the same sense that polygraphs are signatures for generating free monoidal categories, effectful polygraphs are signatures for generating free effectful categories.

Definition 7.1. An *effectful polygraph* is a pair of polygraphs $(\mathcal{V}, \mathcal{G})$ sharing the same objects, $\mathcal{V}_{\mathsf{obj}} = \mathcal{G}_{\mathsf{obj}}$. A morphism of effectful polygraphs $(u, f) \colon (\mathcal{V}, \mathcal{G}) \to (\mathcal{W}, \mathcal{H})$ is a pair of morphisms of polygraphs, $u \colon \mathcal{V} \to \mathcal{W}$ and $f \colon \mathcal{G} \to \mathcal{H}$, such that they coincide on objects, $f_{\mathsf{obj}} = u_{\mathsf{obj}}$.

7.2. Adding Runtime. Recall from Section 1.3 that there exists an adjunction between polygraphs and strict monoidal categories. Any monoidal category $\mathbb{C}$ can be seen as a polygraph, $\mathsf{Forget}(\mathbb{C})$, where the edges are morphisms

$$\mathsf{Forget}(\mathbb{C})(A_0, \dots, A_n; B_0, \dots, B_m) = \mathbb{C}(A_0 \otimes \dots \otimes A_n; B_0 \otimes \dots \otimes B_m),$$

and we forget about composition and tensoring. Given a polygraph $\mathcal{G}$, the free strict monoidal category, which we will now write as $\mathrm{Mon}(\mathcal{G}) = \mathsf{String}(\mathcal{G})$, is the strict monoidal category that has as morphisms the string diagrams over the generators of the polygraph.

We will construct a similar adjunction between effectful polygraphs and effectful categories. Let us start by formally adding the runtime to a free monoidal category.

Definition 7.2 (Runtime monoidal category)**.** Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph. Its *runtime monoidal category*, $\mathrm{Mon}_{\mathrm{Run}}(\mathcal{V}, \mathcal{G})$, is the monoidal category freely generated from adding an extra object – the runtime, R – to the input and output of every effectful generator in $\mathcal{G}$ (but not to those in $\mathcal{V}$), and letting that extra object be braided with respect to every other object of the category.

In other words, it is the monoidal category freely generated by the following polygraph, $\mathrm{Run}(\mathcal{V}, \mathcal{G})$, (Figure 32), assuming $A_0, \dots, A_n$ and $B_0, \dots, B_m$ are distinct from R

- $\mathsf{Run}(\mathcal{V},\mathcal{G})_{\text{obj}} = \mathcal{G}_{\text{obj}} + \{R\} = \mathcal{V}_{\text{obj}} + \{R\}$,
- $\mathsf{Run}(\mathcal{V},\mathcal{G})(R, A_0, \dots, A_n; R, B_0, \dots, B_n) = \mathcal{G}(A_0, \dots, A_n; B_0, \dots, B_n)$,
- $\mathsf{Run}(\mathcal{V},\mathcal{G})(A_0, \dots, A_n; B_0, \dots, B_n) = \mathcal{V}(A_0, \dots, A_n; B_0, \dots, B_n)$,
- $\mathsf{Run}(\mathcal{V},\mathcal{G})(R, A_0; A_0, R) = \mathsf{Run}(\mathcal{V},\mathcal{G})(A_0, R; R, A_0) = \{\sigma\}$,

with $\mathsf{Run}(\mathcal{V},\mathcal{G})$ empty in any other case, and quotiented by the braiding axioms for R (Figure 33).

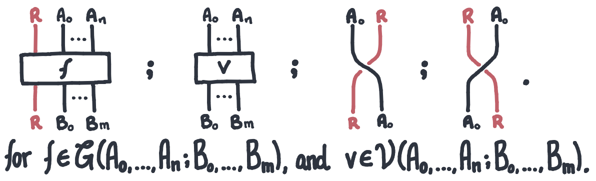


FIGURE 32. Generators for the runtime monoidal category.

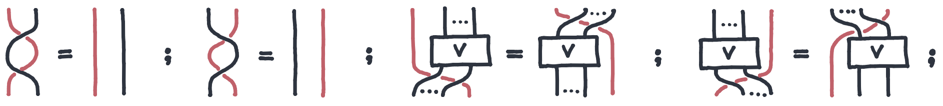


FIGURE 33. Axioms for the runtime monoidal category.

Somehow, we are asking the runtime R to be in the Drinfeld centre [DGNO10] of the monoidal category. The extra wire that R provides is only used to prevent interchange, and so it does not really matter where it is placed in the input and the output. We can choose to always place it on the left, for instance – and indeed we will be able to do so – but a better solution is to just consider objects "up to some runtime braidings". This is formalized by the notion of *braid clique*.

Definition 7.3 (Braid clique). Given any list of objects $A_0, \dots, A_n$ in $\mathcal{V}_{\text{obj}} = \mathcal{G}_{\text{obj}}$, we construct a *clique* [Tod10, Shu18] in the category $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V},\mathcal{G})$: we consider the objects, $A_0 \otimes \dots \otimes R_{(i)} \otimes \dots \otimes A_n$, created by inserting the runtime R in all of the possible $0 \leq i \leq n+1$ positions; and we consider the family of commuting isomorphisms constructed by braiding the runtime,

$$\sigma_{i,j} \colon A_0 \otimes \dots \otimes R_{(i)} \otimes \dots \otimes A_n \to A_0 \otimes \dots \otimes R_{(j)} \otimes \dots \otimes A_n.$$

We call this the *braid clique*, $\mathsf{Braid}_R(A_0, \dots, A_n)$, on that list.

Definition 7.4. A *braid clique morphism*,

$$f \colon \mathsf{Braid}_R(A_0, \dots, A_n) \to \mathsf{Braid}_R(B_0, \dots, B_m),$$

is a family of morphisms in the runtime monoidal category, $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V},\mathcal{G})$, from each of the objects of first clique to each of the objects of the second clique,

$$f_{ik} \colon A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R_{(k)} \otimes \ldots \otimes B_m,$$

that moreover commutes with all braiding isomorphisms, $f_{ij} ⨟ \sigma_{jk} = \sigma_{il} ⨟ f$.

A braid clique morphism $f\colon \mathrm{Braid}_R(A_0,\ldots,A_n) \to \mathrm{Braid}_R(B_0,\ldots,B_m)$ is fully determined by *any* of its components, by pre/post-composing it with braidings. In particular, a braid clique morphism is always fully determined by its leftmost component $f_{00}\colon R \otimes A_0 \otimes \ldots \otimes A_n \to R \otimes B_0 \otimes \ldots \otimes B_m$.

Lemma 7.5. *Let $(\mathcal{V},\mathcal{G})$ be an effectful polygraph. There exists a premonoidal category, $\mathsf{Eff}(\mathcal{V},\mathcal{G})$, that has objects the braid cliques, $\mathrm{Braid}_R(A_0,\ldots,A_n)$, in $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V},\mathcal{G})$, and as morphisms the braid clique morphisms between them. See Appendix.*

PROOF. First, let us give $\mathsf{Eff}(\mathcal{V},\mathcal{G})$ the structure of a category. The identity on $\mathrm{Braid}_R(A_0,\ldots,A_n)$ is the identity on $R \otimes A$. The composition of a morphism $R \otimes A \to R \otimes B$ with a morphism $R \otimes B \to R \otimes C$ is their plain composition in $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V},\mathcal{G})$.

Let us now check that it is moreover a premonoidal category. Tensoring of cliques is given by concatenation of lists, which coincides with the tensor in $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V},\mathcal{G})$. However, it is interesting to note that the tensor of morphisms cannot be defined in this way: a morphism $R \otimes A \to R \otimes B$ cannot be tensored with a morphism $R \otimes A' \to R \otimes B'$ to obtain a morphism $R \otimes A \otimes A' \to R \otimes B \otimes B'$.

Whiskering of a morphism $f\colon R \otimes A \to R \otimes B$ is defined with braidings in the left case, $R \otimes C \otimes A \to R \otimes C \otimes B$, and by plain whiskering in the right case, $R \otimes A \otimes C \to R \otimes B \otimes C$, as depicted in Figure 34. Finally, the associators and

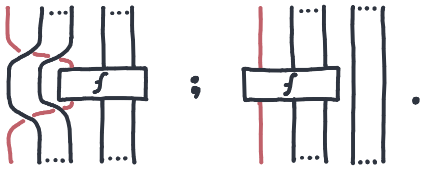


FIGURE 34. Whiskering in the runtime premonoidal category.

unitors are identities, which are always natural and central. □

Lemma 7.6. *Let $(\mathcal{V},\mathcal{G})$ be an effectful polygraph. There exists an identity-on-objects functor $\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V},\mathcal{G})$ that strictly preserves the premonoidal structure and whose image is central.*

PROOF. A morphism $v \in \mathsf{Mon}(\mathcal{V})(A, B)$ induces a morphism $(\mathrm{id}_R \otimes v) \in \mathsf{Mon}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(R \otimes A, R \otimes B)$, which can be read as a morphism of cliques $(\mathrm{id}_R \otimes v) \in \mathsf{Eff}(\mathcal{V}, \mathcal{G})(A, B)$. This is tensoring with an identity, which is indeed functorial.

Let us now show that this functor strictly preserves the premonoidal structure. The fact that it preserves right whiskerings is immediate. The fact that it preserves left whiskerings follows from the axioms of symmetry (Figure 35, left). Associators and unitors are identities, which are preserved by tensoring with an identity. Finally, we can check by string diagrams that the image of this functor

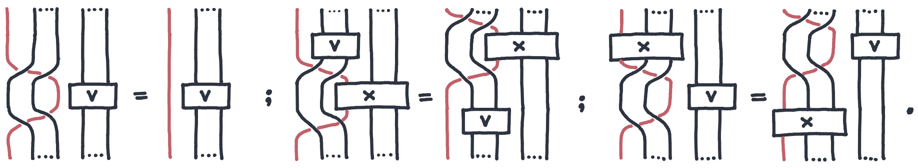


FIGURE 35. Preservation of whiskerings, and centrality.

is central, interchanging with any given $x \colon R \otimes C \to R \otimes D$ (Figure 35, center and right). □

Lemma 7.7. *Let $(\mathcal{V}, \mathcal{G})$ be an effectful polygraph and consider the effectful category determined by $\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G})$. Let $\mathbb{V} \to \mathbb{C}$ be a strict effectful category endowed with an effectful polygraph morphism $F \colon (\mathcal{V}, \mathcal{G}) \to \mathcal{U}(\mathbb{V}, \mathbb{C})$. There exists a unique strict effectful functor from $(\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G}))$ to $(\mathbb{V} \to \mathbb{C})$ commuting with F as an effectful polygraph morphism.*

PROOF. By freeness, there already exists a unique strict monoidal functor $H_0 \colon \mathsf{Mon}(\mathcal{V}) \to \mathbb{V}$ that sends any object $A \in \mathcal{V}_{\mathrm{obj}}$ to $F_{obj}(A)$. We will show there is a unique way to extend this functor together with the hypergraph assignment $\mathcal{G} \to \mathbb{C}$ into a functor $H \colon \mathsf{Eff}(\mathcal{V}, \mathcal{G}) \to \mathbb{C}$. Giving such a functor amounts to give some mapping of morphisms containing the runtime R in some position in their input and output,

$$f \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$$

to morphisms $H(f) \colon FA_0 \otimes \ldots \otimes FA_n \to FB_0 \otimes \ldots \otimes FB_n$ in $\mathbb{C}$, in a way that preserves composition, whiskerings, inclusions from $\mathsf{Mon}(\mathcal{V})$, and that is invariant to composition with braidings. In order to define this mapping, we will perform structural induction over the monoidal terms of the runtime monoidal category of the form $\mathsf{Mon}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(A_0 \otimes \ldots \otimes R^{(i)} \otimes \ldots \otimes A_n, R \otimes B_0 \otimes \ldots \otimes R^{(j)} \otimes \ldots \otimes B_m)$ and show that it is the only mapping with these properties (Figure 36).

Monoidal terms in a strict, freely presented, monoidal category are formed by identities (id), composition ($\fatsemi$), tensoring ($\otimes$), and some generators (in this case, in Figure 32). Monoidal terms are subject to *(i)* functoriality of the tensor, $\mathrm{id} \otimes \mathrm{id} = \mathrm{id}$ and $(f \fatsemi g) \otimes (h \fatsemi k) = (f \otimes h) \fatsemi (g \otimes k)$; *(ii)* associativity and unitality of the tensor, $f \otimes \mathrm{id}_I = f$ and $f \otimes (g \otimes h) = (f \otimes g) \otimes h$; *(iii)* the usual unitality, $f \fatsemi \mathrm{id} = f$ and $\mathrm{id} \fatsemi f = f$ and associativity $f \fatsemi (g \fatsemi h) = (f \fatsemi g) \fatsemi h$; *(iv)* the axioms of our presentation (in this case, in Figure 33).

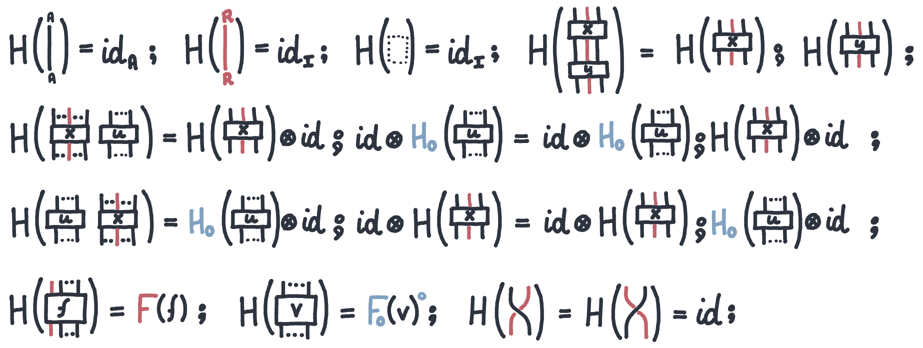


Figure 36. Assignment on morphisms, defined by structural induction on terms.

- If the term is an identity, it can be *(i)* an identity on an object $A \in (\mathcal{V}, \mathcal{G})_{\mathrm{obj}}$, in which case it must be mapped to the same identity by functoriality, $H(\mathrm{id}_A) = \mathrm{id}_A$; *(ii)* an identity on the runtime, in which case it must be mapped to the identity on the unit object, $H(\mathrm{id}_R) = \mathrm{id}_I$; or *(iii)* an identity on the unit object, in which case it must be mapped to the identity on the unit, $H(\mathrm{id}_I) = \mathrm{id}_I$.
- If the term is a composition, $(f \fatsemi g) \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to C_0 \otimes \ldots \otimes R \otimes \ldots \otimes C_k$, it must be along a boundary of the form $B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$: this is because every generator leaves the number of runtimes, R, invariant. Thus, each one of the components determines itself a braid clique morphism. We must preserve composition of braid clique morphisms, so we must map $H(f \fatsemi g) = H(f) \fatsemi H(g)$.
- If the term is a tensor of two terms, $(x \otimes u) \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_m$, then only one of them was a term taking R as input and output (without loss of generality, assume it to be the first one) and the other was not: again, by construction, there are no morphisms taking one R as input and producing none, or viceversa. We split this morphism into $x \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_{i-1} \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_{j-1}$ and $u \colon A_i \otimes \ldots \otimes A_n \to B_j \otimes \ldots \otimes B_m$.

  Again by structural induction, this time over terms $u \colon A_i \otimes \ldots \otimes A_n \to B_j \otimes \ldots \otimes B_m$, we know that the morphism must be either a generator in $\mathcal{V}(A_i, \ldots, A_n; B_j, \ldots, B_n)$ or a composition and tensoring of them. That is, u

is a morphism in the image of $\mathrm{Mon}(\mathcal{V})$, and it must be mapped according to the functor $H_0 \colon \mathrm{Mon}(\mathcal{V}) \to \mathbb{V}$.

By induction hypothesis, we know how to map the morphism $x \colon A_0 \otimes \ldots \otimes R \otimes \ldots \otimes A_{i-1} \to B_0 \otimes \ldots \otimes R \otimes \ldots \otimes B_{j-1}$. This means that, given any tensoring $x \otimes u$, we must map it to $H(x \otimes u) = (H(x) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes H_0(u)) = (\mathrm{id} \otimes H_0(u)) ⨾ (H(x) \otimes \mathrm{id})$, where $H_0(u)$ is central.

- If the string diagram consists of a single generator, $f \colon R \otimes A \to R \otimes B$, it can only come from a generator $f \in \mathrm{Run}(\mathcal{V}, \mathcal{G})(R, A_0, \ldots, A_n; R, B_0, \ldots, B_m) = \mathcal{G}(A_0, \ldots, A_n; B_0, \ldots, B_m)$, which must be mapped to $H(f) = F(f) \in \mathbb{C}(A_0 \otimes \ldots \otimes A_n, B_0 \otimes \ldots \otimes B_m)$. If the string diagram consists of a single braiding, it must be mapped to the identity, because the want the assignment to be invariant to braidings.

Now, we need to prove that this assignment is well-defined with respect to the axioms of these monoidal terms. Our reasoning follows Figure 37.

- The tensor is functorial. We know that $H(\mathrm{id} \otimes \mathrm{id}) = H(\mathrm{id})$, both are identities and that can be formally proven by induction on the number of wires. Now, for the interchange law, consider a quartet of morphisms that can be composed or tensored first and such that, without loss of generality, we assume the runtime to be on the left side. Then, we can use centrality to argue that

$$\begin{aligned} H((x \otimes u) ⨾ (y \otimes v)) &= (H(x) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes H_0(u)) ⨾ (H(y) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes H_0(v)) \\ &= ((H(x) ⨾ H(y)) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes (H_0(u) ⨾ H_0(v))) \\ &= H((x ⨾ y) \otimes (u ⨾ v)). \end{aligned}$$

- The tensor is monoidal. We know that $H(x \otimes \mathrm{id}_I) = (H(x) \otimes \mathrm{id}_I) ⨾ (\mathrm{id} \otimes \mathrm{id}_I) = H(x)$. Now, for associativity, consider a triple of morphisms that can be tensored in two ways and such that, without loss of generality, we assume the runtime to be on the left side. Then, we can use centrality to argue that

$$\begin{aligned} H((x \otimes u) \otimes v) &= (((H(x) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes H_0(u))) \otimes \mathrm{id}) ⨾ \mathrm{id} \otimes H_0(v) \\ &= (H(x) \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes H_0(u) \otimes H_0(v)) \\ &= H(x \otimes (u \otimes v)) \end{aligned}$$

- The terms form a category. And indeed, it is true by construction that $H(x ⨾ (y ⨾ z)) = H((x ⨾ y) ⨾ z)$ and also that $H(x ⨾ \mathrm{id}) = H(x)$ because H preserves composition.
- The runtime category enforces some axioms. The composition of two braidings is mapped to the identity by the fact that H preserves composition and sends both to the identity. Both sides of the braid naturality over a morphism v are mapped to $H_0(v)$; with the multiple braidings being mapped again to the identity.

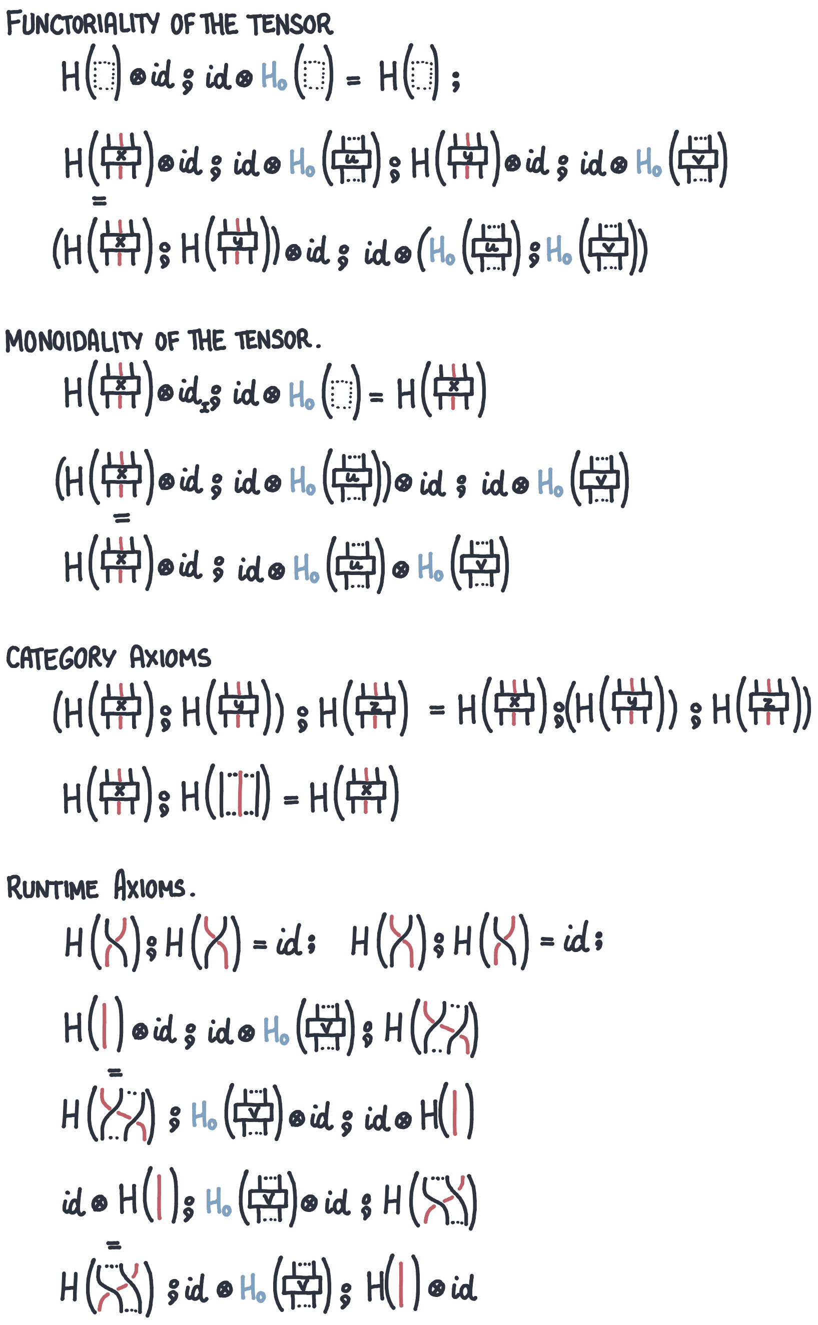


FIGURE 37. The assignment is well defined.

Thus, H is well-defined and it defines the only possible assignment and the only possible strict premonoidal functor. □

THEOREM 7.8 (Runtime as a resource). *The free strict effectful category over an effectful polygraph $(\mathcal{V}, \mathcal{G})$ is $\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G})$. Its morphisms $A \to B$ are in bijection with the morphisms $R \otimes A \to R \otimes B$ of the runtime monoidal category,*

$$\mathsf{Eff}(\mathcal{V}, \mathcal{G})(A, B) \cong \mathsf{Mon}_{\mathsf{Run}}(\mathcal{V}, \mathcal{G})(R \otimes A, R \otimes B).$$

PROOF. We must first show that $\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G})$ is an effectful category. The first step is to see that $\mathsf{Eff}(\mathcal{V}, \mathcal{G})$ forms a premonoidal category (Lemma 7.5). We already know that $\mathsf{Mon}(\mathcal{V})$ is a monoidal category: a strict, freely generated one. There exists an identity on objects functor, $\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G})$, that strictly preserves the premonoidal structure and centrality (Lemma 7.6).

Let us now show that it is the free one over the effectful polygraph $(\mathcal{V}, \mathcal{G})$. Let $\mathbb{V} \to \mathbb{C}$ be an effectful category, with an effectful polygraph map $F : (\mathcal{V}, \mathcal{G}) \to \mathcal{U}(\mathbb{V}, \mathbb{C})$. We can construct a unique effectful functor from $(\mathsf{Mon}(\mathcal{V}) \to \mathsf{Eff}(\mathcal{V}, \mathcal{G}))$ to $(\mathbb{V} \to \mathbb{C})$ giving its universal property (Lemma 7.7). □

Corollary 7.9 (String diagrams for effectful categories)**.** *We can use string diagrams for effectful categories, quotiented under the same isotopy as for monoidal categories, provided that we do represent the runtime as an extra wire that needs to be the input and output of every effectful morphism.*

7.3. Example: a Theory of Global State. Let us provide an example of reasoning using the string diagrams for premonoidal categories. Imperative programs are characterized by the presence of a global state that can be mutated. Reading or writing to this global state constitutes an effectful computation: the order of operations that affect some global state cannot be changed. Let us propose a simple theory of global state and let us show that it is enough to capture the phenomenon of *race conditions.*

Definition 7.10. The *theory of global state* is given by a single object X; two pure generators, $(\curlywedge) : X \to X \otimes X$ and $(\bullet) : X \to I$, allowing copy and discard; and two effectful generators, $\mathtt{put} : X \to I$ and $\mathtt{get} : I \to X$, quotiented by the equations in Figure 39.

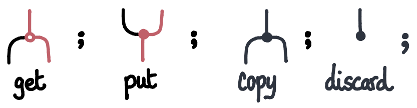


FIGURE 38. Generators of the theory of global state.

These two `put` and `get` generators, without extra axioms, are enough for capturing what happens when a process can *send* or *receive* resources; in further sections, we will develop this theory. Right now, we are only concerned with the theory of a single *global state*, accessed by a single process: we can impose some axioms that assert that the memory was not changed by anyone but this single process.

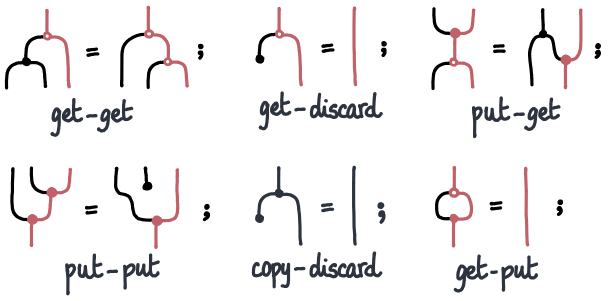


FIGURE 39. Axioms of the theory of global state.

The equations in Figure 39 say that: *(i)* reading the global state twice gets us the same result, *(ii)* reading the global state and discarding the result is the same as doing nothing, *(iii)* writing something to the global state and then reading it is the same as keeping a copy of it, *(iv)* writing twice to the global state keeps only the last thing that was written, *(v)* copying and discarding a copy is doing nothing, and *(vi)* reading something and immediately writing it to the global state is the same as doing nothing.

Proposition 7.11 (Race conditions)**.** *Concurrently mixing two processes that share a global state, f and g, can produce four possible results:* (i) *only the result of the first one is preserved,* (ii) *only the result of the second one is preserved, or* (iii,iv) *the composition of both is preserved, in any order, $f \mathbin{⨾} g$ or $g \mathbin{⨾} f$.*

PROOF. We work in the theory of global state adding two processes, $f \colon X \to X$ and $g \colon X \to X$, that can moreover be discarded, meaning $f \mathbin{⨾} \varepsilon = g \mathbin{⨾} \varepsilon = \varepsilon$. We employ the string diagrams of premonoidal categories. The first three diagrams in Figure 40 correspond to the first three cases, the last one is analogous to the third one. □

7.4. Bibliography. Alan Jeffrey pioneered a string diagrammatic representation of programs using an extra wire to represent runtime [Jef97a], all the credit for this idea should go there. Staton and Møgelberg [MS14] already showed how

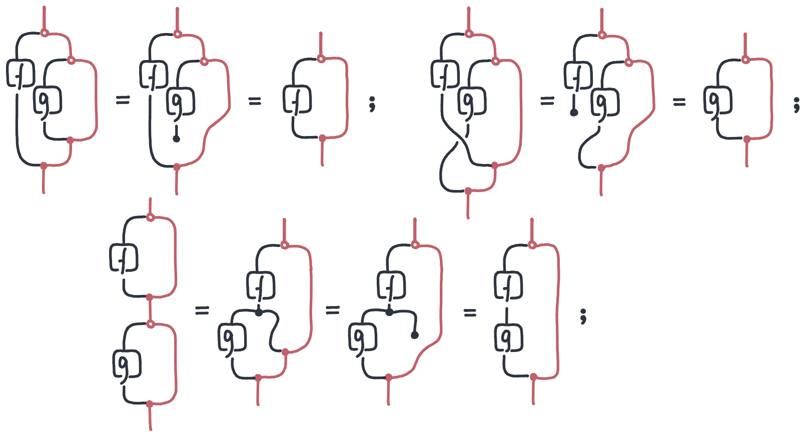


FIGURE 40. Race conditions in the theory of global state.

any premonoidal category could be reinterpreted as the Kleisli category of a state promonad.

We take these ideas a step further, showing that a particular syntax for monoidal categories can be used to talk about premonoidal categories as well. Our study may demystify premonoidal categories, or at least make them more accessible to a category theorist already interested in monoidal categories: premonoidal categories are simply monoidal categories with a hidden state object.

CHAPTER 2

Context Theory

Context Theory

Our goal in this thesis is to study how processes compose from their constituent parts while keeping for each one of these incomplete parts (each context) a meaning of its own: the meaning – the semantics – of the whole process is then determined by the semantics of each one of its parts, and how they compose. This is the principle of compositionality.

In the categorical framework, the structures that govern composition and decomposition are Lambek's multicategories: categories where morphisms have multiple inputs that get composed into a single output. We will decompose multicategories and compare them to monoidal categories using profunctors and dinaturality, which we claim to be the right mathematical tools to talk about abstract process composition in Section 1.

We give a short exposition of multicategories in Section 2, and then introduce a type of multicategory with the particular property that every transformation can be decomposed into smaller ones: *malleable multicategories*, which we prove equivalent to promonoidal categories in Section 3. Our main result in this section is a characterization of the multicategory that governs how morphisms compose in a category – the malleable multicategory of *spliced arrows* – as cofreely generated by the category, Section 4. This result is a variant on a recent result by Melliés and Zeilberger [MZ22]; and Chapter 3 will extend this theory for the first time to the setting of monoidal categories.

1. Profunctors and Coends

1.1. Profunctors. A profunctor from a category $\mathbb{A}$ to a category $\mathbb{B}$ is a functor $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathbf{Set}$ [Bén00]. Profunctors describe families of processes indexed functorially by the objects of two different categories. The canonical example of a profunctor is the one that returns the set of morphisms between two objects of the same category, $\mathbb{A}(\bullet;\bullet)\colon \mathbb{A}^{op} \times \mathbb{A} \to \mathbf{Set}$. Profunctors, however, do not need to be restricted to a single category: this makes them useful to study the relation between processes of different categories.

Categorically, profunctors can be seen as a categorification of the concept of *relations*, functions $A \times B \to 2$. Under this analogy, existential quantifiers correspond to *coends*. This section will first introduce profunctors (Definition 1.1), then a naturality relation for them (Definition 1.4) and, finally, their composition using coends (Sections 1.3 and 1.4). We connect them explicitly to process theories in Section 1.5.

Definition 1.1. A *profunctor* $(P, \prec, \succ)$ between two categories, $\mathbb{A}$ and $\mathbb{B}$, is a family of sets, $P(A, B)$, indexed by objects $A \in \mathbb{A}_{obj}$ and $B \in \mathbb{B}_{obj}$, and endowed with jointly functorial left and right actions of the morphisms of the two categories $\mathbb{A}$ and $\mathbb{B}$, respectively.

Explicitly, types of these actions are

$$(\succ)\colon \mathbb{A}(A';A) \times P(A';B) \to P(A;B)$$
$$(\prec)\colon P(A;B) \times \mathbb{B}(B;B') \to P(A;B')$$

These two actions must be compatible, $(f \succ p) \prec g = f \succ (p \prec g)$, they must preserve identities, $id \succ p = p$, and $p \prec id = p$, and they must preserve composition $(p \prec f) \prec g = p \prec (f \mathbin{⨟} g)$ and $f \succ (g \succ p) = (f \mathbin{⨟} g) \succ p$.

More succinctly, a profunctor $P\colon \mathbb{A} \to \mathbb{B}$ is the same as a functor $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathbf{Set}$. When presented as a family of sets with a pair of actions, profunctors have been sometimes called *bimodules*.

Definition 1.2 (Parallel composition)**.** Two profunctors $P\colon \mathbb{A}_1^{op} \times \mathbb{B}_1 \to \mathbf{Set}$ and $Q\colon \mathbb{A}_2^{op} \times \mathbb{B}_2 \to \mathbf{Set}$ compose *in parallel* into a profunctor $P \times Q\colon \mathbb{A}_1^{op} \times \mathbb{A}_2^{op} \times \mathbb{B}_1 \times \mathbb{B}_2 \to \mathbf{Set}$ defined by

$$(P \times Q)(A, A'; B, B') = P(A;B) \times Q(A';B').$$

Remark 1.3. We will consider profunctors between product categories explicitly: a *profunctor* $P\colon \mathbb{A}_0 \times \ldots \times \mathbb{A}_n \to \mathbb{B}_0 \times \ldots \times \mathbb{B}_m$ is a functor

$$P\colon \mathbb{A}_0^{op} \ldots \times \mathbb{A}_n^{op} \times \mathbb{B}_0 \times \ldots \times \mathbb{B}_m \to \mathbf{Set}.$$

For our purposes, a profunctor $P(A_0, \ldots, A_n; B_0, \ldots, B_m)$ is a family of processes indexed by contravariant inputs $A_0, \ldots, A_n$ and covariant outputs $B_0, \ldots, B_m$. The profunctor is endowed with jointly functorial left $(\succ_0, \ldots, \succ_n)$ and right

$(\prec_0, ..., \prec_m)$ actions of the morphisms of $\mathbb{A}_0, ..., \mathbb{A}_n$ and $\mathbb{B}_0, ..., \mathbb{B}_m$, respectively [Bén00, Lor21]. We will simply use $(\prec/\succ)$ without any subscript whenever the input/output is unique.

Composing profunctors *sequentially* is subtle: the same processes could arise as the composite of different pairs of processes, so we need to impose an equivalence relation. Imagine we try to connect two different processes:

$$p \in P(A_0, ..., A_n; B_0, \dots, B_m), \text{ and } q \in Q(C_0, ..., C_k; D_0, \dots, D_h);$$

and we have some morphism $f \colon B_i \to C_j$ that translates the i-th output port of p to the j-th input port of q. Let us write $({}_i|_j)$ for this connection operation. Note that we could connect them in two different ways: we could

(1) change *the output of the first process* $p \prec_i f$ before connecting both, thus obtaining $(p \prec_i f) \,{}_i|_j\, q$;
(2) or change *the input of the second process* $f \succ_j q$ before connecting both, thus obtaining $p \,{}_i|_j\, (f \succ_j q)$.

These are different descriptions, made up of two different components. However, they essentially describe the same process: they are *dinaturally equal.* Indeed, profunctors are canonically endowed with this notion of equivalence [Bén00, Lor21], precisely equating these two descriptions. Profunctors, and their elements, are thus composed *up to dinatural equivalence.*

1.2. Dinaturality and Composition. Dinaturality is a canonical notion of equivalence for profunctors: it arises naturally from the construction of the bicategory of profunctors, but it also has a good interpretation in terms of processess.

Definition 1.4 (Dinatural equivalence)**.** For any functor $P \colon \mathbb{C}^{op} \times \mathbb{C} \to \mathbf{Set}$, consider the set

$$S_P = \sum_{M \in \mathbb{C}} P(M; M).$$

Dinatural equivalence, $(\sim)$, on the set S_P is the smallest equivalence relation satisfying $(r \succ p) \sim (p \prec r)$ for each $p \in P(M; N)$ and each $r \in \mathbb{C}(N; M)$.

Coproducts quotiented by dinatural equivalence construct a particular form of colimit called a *coend.* Under the process interpretation of profunctors, taking a coend means *plugging an output to an input* of the same type.

Definition 1.5 (Coend)**.** Let $P \colon \mathbb{C}^{op} \times \mathbb{C} \to \mathbf{Set}$ be a functor. Its *coend* is the coproduct of $P(M, M)$ indexed by $M \in \mathbb{C}$, quotiented by dinatural equivalence.

$$\int^{M \in \mathbb{C}} P(M; M) := \left(\sum_{M \in \mathbb{C}} P(M; M) \middle/ \sim \right).$$

That is, the coend is the colimit of the diagram containing a *cospan* $P(M;M) \leftarrow P(M;N) \rightarrow P(N;N)$ for each $f\colon N \rightarrow M$.

Definition 1.6 (Sequential composition)**.** Two profunctors $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathbf{Set}$ and $Q\colon \mathbb{B}^{op} \times \mathbb{C} \to \mathbf{Set}$ compose sequentially into a profunctor $P \diamond Q\colon \mathbb{A}^{op} \times \mathbb{C} \to \mathbf{Set}$ defined by

$$(P \diamond Q)(A;C) = \int^{B \in \mathbb{B}} P(A;B) \times Q(B;C).$$

The hom-profunctor $\hom\colon \mathbb{A}^{op} \times \mathbb{A} \to \mathbf{Set}$ that returns the set of morphisms between two objects is the unit for sequential composition. Sequential composition is associative up to isomorphism.

1.3. Coend Calculus. *Coend calculus* is the name given to the algebraic manipulations of coends that prove isomorphisms or construct natural transformations between profunctors using the behaviour of *coends.* MacLane [ML71] and Loregian [Lor21] give presentations of coend calculus.

Proposition 1.7 (Yoneda reduction)**.** *Let $\mathbb{C}$ be any category and let $F\colon \mathbb{C} \to \mathbf{Set}$ be a functor; the following isomorphism holds for any given object $A \in \mathbb{C}_{obj}$.*

$$\int^{X \in \mathbb{C}} \mathbb{C}(X;A) \times FX \cong FA.$$

Following the analogy with classical analysis, the hom *profunctor works as a Dirac's delta.*

Proposition 1.8 (Fubini rule)**.** *Coends commute between them; that is, there exists a natural isomorphism*

$$\int^{X_1 \in \mathbb{C}} \int^{X_2 \in \mathbb{C}} P(X_1, X_2; X_1, X_2) \cong \int^{X_2 \in \mathbb{C}} \int^{X_1 \in \mathbb{C}} P(X_1, X_2; X_1, X_2).$$

In fact, they are both isomorphic to the coend over the product category,

$$\int^{(X_1,X_2) \in \mathbb{C} \times \mathbb{C}} P(X_1, X_2; X_1, X_2).$$

Following the analogy with classical analysis, coends follow the Fubini rule for integrals.

1.4. The Point of Coend Calculus. In the same way that regular logic links relations, a coend calculus expression is a list of profunctors linked by some objects that are bound to a coend. Usually, the isomorphisms that we construct are never made explicit, and it is difficult for the reader to compute the precise map we constructed.

Fortunately, this has a straightforward solution. We propose to *point* the coends: to write an profunctorial expression, P, together with the *generic element* it computes, P_p. An expression of pointed coend calculus is a coend bounding some objects and a series of *pointed profunctors.* For instance, we may write

$$\textstyle\int^{M,N} P(A;M,N)_f \times Q(M;B)_g \times \mathbb{C}(N;C)_h, \quad \text{instead of just}$$
$$\textstyle\int^{M,N} P(A;M,N) \times Q(M;B) \times \mathbb{C}(N;C).$$

Coends quotient expressions by dinaturality, meaning that any left action on the covariant occurrence of a bounded variable can be equivalently written as a right action on its contravariant occurrence. In terms of pointed profunctors, this means that

$$\int^{N} P(A;N)_{(f \prec h)} \times Q(N;B)_g = \int^{M} P(A;M)_f \times Q(M;B)_{(h \succ g)}.$$

Proposition 1.9. *Let $\mathbb{C}$ be a category and let $F \colon \mathbb{C}^{op} \to \mathbf{Set}$ and $G \colon \mathbb{C} \to \mathbf{Set}$ be a presheaf and a copresheaf, respectively. The following are natural isomorphisms of pointed profunctors,*

$$\begin{aligned}\int^{X} \mathbb{C}(X;A)_f \times F(X)_h \;&\cong\; F(A)_{(f \succ h)};\\ \int^{X} \mathbb{C}(A;X)_f \times G(X)_h \;&\cong\; G(A)_{(h \prec f)}.\end{aligned}$$

We call these isomorphisms the "pointed" Yoneda reductions.

Remark 1.10. Using pointed coends, any derivation does also include the computation of the isomorphism it induces. As an example, compare the following with the usual coend derivation of a cartesian lens [CEG⁺20],

Proposition 1.11. *In a cartesian monoidal category, the pairs of morphisms $\mathbb{C}(A;M \times X)$ and $\mathbb{C}(M \times Y;B)$, quotiented by dinaturality, are in bijective correspondence with the pairs of morphisms $\mathbb{C}(A;M)$ and $\mathbb{C}(M \times Y;B)$.*

Proof. A function is explicitly constructed by the following derivation.

$$\begin{aligned}&\int^{M} \mathbb{C}(A;M \times X)_f \times \mathbb{C}(M \times Y;B)_g\\ &\cong \quad (\ \textit{by the adjunction } \Delta \dashv \times)\\ &\int^{M} \mathbb{C}(A;M)_{(f ⨟ \pi_1)} \times \mathbb{C}(A;X)_{(f ⨟ \pi_2)} \times \mathbb{C}(M \times Y;B)_g\\ &\cong \quad (\ \textit{by pointed Yoneda lemma}\ )\\ &\mathbb{C}(A;X)_{(f ⨟ \pi_2)} \times \mathbb{C}(X \times Y;B)_{((f ⨟ \pi_1)\otimes id) ⨟ g}.\end{aligned}$$

The function mapping an equivalence class $[f,g]$ to $(f ⨟ \pi_2; (f ⨟ \pi_1) \otimes id)$ is a bijection because it has been constructed from composing bijections.

Indeed, in the first step, we have used that the adjunction $(\Delta \dashv \times)$ is given by postcomposition with projections and; in the second step, we use that the action on the last profunctor is defined as $h \succ g = (h \otimes id) \mathbin{⨟} g$. The bijection has been explicitly constructed as sending the pair $(f; g)$ to $(f \mathbin{⨟} \pi_2; ((f \mathbin{⨟} \pi_1) \otimes id) \mathbin{⨟} g)$. $\square$

1.5. Promonads. Promonads are to profunctors what monads are to functors: to quip, a promonad is just a monoid in the category of endoprofunctors. It may be then surprising to see that so little attention has been devoted to them, relative to their functorial counterparts. The main source of examples and focus of attention has been the semantics of programming languages [Hug00, Pat01, JHH09]. Strong monads are commonly used to give categorical semantics of effectful programs [Mog91], and the so-called *arrows* (or *strong promonads*) strictly generalize them: they coincide with our previous definition of effectful category [HJ06].

Part of the reason behind the relative unimportance given to promonads elsewhere may stem from precisely from that fact: promonads over a category can be shown in an elementary way to be equivalent to identity-on-objects functors from that category [Lor21]. The explicit proof is, however, difficult to find in the literature, and so we include it here (Theorem 1.14).

Under this interpretation, promonads are new morphisms for an old category. We can reinterpret the old morphisms into the new ones in a functorial way. The paradigmatic example is again that of Kleisli or cokleisli categories of strong monads and comonads. This structure is richer than it may sound, and we will explore it further during the rest of this text.

Definition 1.12. A *promonad* $(P, \star, ^\circ)$ over a category $\mathbb{C}$ is a profunctor $P\colon \mathbb{C}^{op} \times \mathbb{C} \to \mathbf{Set}$ together with natural transformations for inclusion $(^\circ)_{X,Y}\colon \mathbb{C}(X; Y) \to P(X; Y)$ and multiplication $(\star)_{X,Y}\colon P(X; Y) \times P(Y; Z) \to P(X; Z)$, and such that

i. the right action is premultiplication, $f^\circ \star p = f \succ p$;
ii. the left action is postmultiplication, $p \star f^\circ = p \prec f$;
iii. multiplication is dinatural, $p \star (f \succ q) = (p \prec f) \star q$;
iv. and multiplication is associative, $(p_1 \star p_2) \star p_3 = p_1 \star (p_2 \star p_3)$.

Equivalently, promonads are promonoids in the double category of categories, where the dinatural multiplication represents a transformation from the composition of the profunctor P with itself.

Lemma 1.13 (Kleisli category of a promonad)**.** *Every promonad* $(P, \star, ^\circ)$ *induces a category with the same objects as its base category, but with hom-sets given by* $P(\bullet, \bullet)$*, composition given by* $(\star)$ *and identities given by* (id°)*. This is called its* Kleisli category*,* $\mathsf{Kleisli}(P)$*. Moreover, there exists an identity-on-objects functor* $\mathbb{C} \to \mathsf{Kleisli}(P)$*, defined on morphisms by the unit of the promonad.*

The converse is also true: every category $\mathbb{C}$ with an identity-on-objects functor from some base category $\mathbb{V}$ arises as the Kleisli category of a promonad.

THEOREM 1.14. *Promonads over a category $\mathbb{C}$ correspond to identity-on-objects functors from the category $\mathbb{C}$. Given any identity-on-objects functor $i\colon \mathbb{C} \to \mathbb{D}$ there exists a unique promonad over $\mathbb{C}$ having $\mathbb{D}$ as its Kleisli category: the promonad given by the profunctor* $\hom_{\mathbb{D}}(i(\bullet), i(\bullet))$.

1.6. Bibliography. Coends, the Yoneda lemma, and their calculus, were introduced in MacLane's monograph [ML71]. A more modern presentation of coend calculus and its applications is in the work of Loregian [Lor21]. This author has also written on the importance of pointed profunctors for open diagrams [Rom20b] and collages [BR23].

2. Multicategories

2.1. Multicategories. Multicategories will provide an algebra for composing multiple pieces into one. A multicategory is like a category where every morphism has a list of inputs instead of a single one. A multicategory, $\mathbb{M}$, contains a set of objects, $\mathbb{M}_{obj}$, as a category does; but instead of a set of morphisms, $\mathbb{M}(X; Y)$, for every pair of objects $X, Y \in \mathbb{M}_{obj}$, it will have a set of *multimorphisms*,

$$\mathbb{M}(X_1, \dots, X_n; Y), \text{ for each list of objects } X_1, \dots, X_n, Y \in \mathbb{M}_{obj}.$$

As in sequent logic, it is easier to denote lists of objects by metavariables. For instance, we will use $\Gamma = X_1, \dots, X_n$ and write $\mathbb{M}(\Gamma; Y)$ for the set of multimorphisms $\mathbb{M}(X_1, \dots, X_n; Y)$.

Definition 2.1. A *multicategory*, $\mathbb{M}$, is a collection of objects, $\mathbb{M}_{obj}$, together with a collection of multimorphisms, $\mathbb{M}(\Gamma; Y)$, for each list of objects $\Gamma = X_0, \dots, X_n \in \mathbb{M}_{obj}$ and each object $Y \in \mathbb{M}_{obj}$.

For each object X, there must be an identity multimorphism, $\mathrm{id}_X \in \mathbb{M}(X; X)$. For each three lists of objects $\Gamma, \Gamma_1, \Gamma_2$ and each two objects Y and Z, there must exist a composition operation (we omit superscripts when clear from the context),

$$(\fatsemi)_i^{\Gamma_1,\Gamma_2} : \mathbb{M}(\Gamma; Y_i) \times \mathbb{M}(\Gamma_1, Y_i, \Gamma_2; Z) \to \mathbb{M}(\Gamma_1, \Gamma, \Gamma_2; Z).$$

Composition must be unital, meaning that $\mathrm{id}_X \fatsemi_X f = f$ and $f \fatsemi \mathrm{id}_Y = f$ every time that the equation is fomally well-typed. Composition must be also associative, meaning that $(h \fatsemi_X g) \fatsemi_Y f = h \fatsemi_X (g \fatsemi_Y f)$; and $g \fatsemi_Y (h \fatsemi_X f) = h \fatsemi_X (g \fatsemi_Y f)$ must hold whenever they are formally well-typed, see Figure 1.

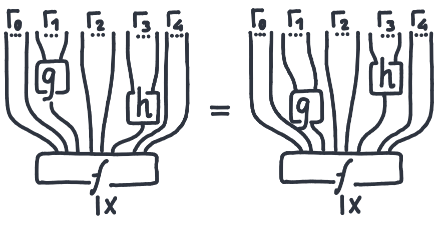


FIGURE 1. Associativity for a multicategory.

2.2. The Category of Multicategories. In the same way categories are the first step towards the theory of functors and natural transformations, multicategories are the first step towards the theory of multifunctors and multinatural transformations. In the same way the formal theory of categories is synthetised by the 2-category **Cat** of categories, functors and natural transformations; the study of multicategories is synthetised by the 2-category **MultiCat** of multicategories, multifunctors and multinatural transformations.

Definition 2.2. A *multifunctor* between two multicategories, $F\colon \mathbb{M} \to \mathbb{N}$, consists of an assignment on objects $F_{obj}\colon \mathbb{M} \to \mathbb{N}$ and an assignment on multimorphisms of any arity,

$$F_n\colon \mathbb{M}(X_1,\dots,X_n;Y) \to \mathbb{N}(F_{obj}X_1,\dots,F_{obj}X_n;F_{obj}Y),$$

that preserves identities, $F_1(\mathrm{id}_X) = \mathrm{id}_{F_{obj}(X)}$, and composition of multimorphisms, $F_{n+m-1}(f \mathbin{⨾}_Y g) = F_n(f) \mathbin{⨾}_{F_{obj}(Y)} F_m(g)$.

Definition 2.3. A multinatural transformation $\theta\colon F \to G$ between two multifunctors $F, G\colon \mathbb{M} \to \mathbb{N}$ is given by a family of multimorphisms $\theta_X \in \mathbb{N}(FX;GX)$ such that, for each multimorphism $f \in \mathbb{M}(X_1,\dots,X_n;Y)$, the following naturality condition holds

$$\theta_{X_1} \mathbin{⨾}_1 \dots \mathbin{⨾}_{n-1} \theta_{X_n} \mathbin{⨾} G(f) = F(f) \mathbin{⨾} \theta_Y .$$

Proposition 2.4. *Multicategories with multifunctors between them form a category,* **Mult***.*

2.3. Application: Shufflings. Let us exemplify multicategories with an example that will become increasingly relevant in this thesis. Shufflings are permutations that preserve the relative ordering of some blocks. We can always count shufflings combinatorially, but multicategories provide the extra structure that allows us to track how different shufflings compose.

Example 2.5. A *shuffling* is a permutation of the elements of multiple blocks that preserves their relative ordering. The multicategory of shufflings has objects the natural numbers and morphisms the shufflings, $\sigma \in \mathbf{Shuf}(p_0,\dots,p_n;q)$ that reorganize $p_0,\dots,p_n$ elements into $q = p_0+\dots+p_n$ without altering their internal ordering.

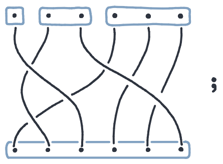


FIGURE 2. Example of a 1,2,3-shuffling.

More explicitly, the number of shufflings $\mathbf{Shuf}(p_0,\dots,p_n;q)$ is given by a multinomial coefficient whenever $q = p_0 + \dots + p_n$,

$$\#\mathbf{Shuf}(p_0,\dots,p_n;p_0+\dots+p_n) = \frac{(p_0+\dots+p_n)!}{p_0! \cdot \dots \cdot p_n!},$$

and it is zero in any other case.

Shufflings exhibit a particular property that motivates our next section: *malleability.* Any shuffling of p_0, p_1 and p_2 can be factored *uniquely* in two different forms: we can first shuffle p_0 and p_1 and then shuffle the result, $p_0 + p_1$, with p_2; or we can first shuffle p_1 and p_2, and then shuffle p_0 with the result, $p_1 + p_2$. For instance, any $1, 2, 3$-shuffling splits uniquely into a $2, 3$-shuffling followed by a $1, 5$-shuffling, but also uniquely into a $1, 2$-shuffling followed by a $3, 3$-shuffling.

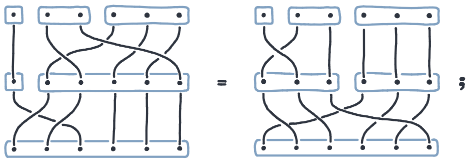


FIGURE 3. Two factorizations of the previous shuffling.

This is a global property: any shuffling can be uniquely factored into smallest shufflings, in whichever arrangement we pick. Any morphism of the multicategory **Shuf** can be factored into any possible shape, uniquely. We say that the multicategories that satisfy this property are *"malleable multicategories"*: shufflings form a malleable multicategory.

3. Malleable Multicategories

A malleable multicategory is a multicategory where each morphism can be morphed uniquely into any possible shape. This means that there exist unique factorizations of each morphism into each one of the possible shapes. Formally, we will define malleable multicategories to have an invertible composition, up to the morphisms of some underlying category.

Definition 3.1. The unary morphisms of a multicategory form a category [Shu16]. In other words, given a multicategory $\mathbb{M}$, the underlying category, $\mathbb{M}^u$, has the same objects as the multicategory, $\mathbb{M}^u_{obj} = \mathbb{M}_{obj}$, and morphisms defined from the unary multimorphisms of the multicategory, $\mathbb{M}^u(X; Y) = \mathbb{M}(X; Y)$. Composition and identities are exactly those of the multicategory.

Remark 3.2. The multimorphisms of a multicategory determine profunctors over the underlying category of the multicategory. The underlying category acts on the multimorphisms by composition,

$$(\succ) \colon \mathbb{M}^u(X; X') \times \mathbb{M}(\Gamma_1, X', \Gamma_2; Y) \to \mathbb{M}(\Gamma_1, X, \Gamma_2; Y),$$
$$(\prec) \colon \mathbb{M}(\Gamma; Y) \times \mathbb{M}^u(Y; Y') \to \mathbb{M}(\Gamma; Y').$$

In any multicategory, composition of multimorphisms is dinatural with respect to the underlying category. This follows from the associativity for multicategories,

$$(f \prec h) \mathbin{⨾}_{X_i} g = (f \mathbin{⨾}_{X_i} h) \mathbin{⨾}_{X_i'} g = f \mathbin{⨾}_{X_i} (h \mathbin{⨾}_{X_i'} g) = f \mathbin{⨾}_{X_i} (h \succ g).$$

As a consequence, composition is well-defined under dinaturality. We define dinatural composition to be composition lifted to the equivalence classes of the dinaturality equivalence relation, which are written as a coend,

$$(⨾)\colon \left(\int^{Y \in \mathbb{M}} \mathbb{M}(\Gamma; Y) \times \mathbb{M}(\Gamma_0, Y, \Gamma_1; Z)\right) \to \mathbb{M}(\Gamma_0, \Gamma, \Gamma_1; Z).$$

Definition 3.3. A *malleable multicategory* is a multicategory where dinatural composition is invertible.

Proposition 3.4. *Malleable multicategories with multifunctors between them form a category,* **mMult**. *This is a wide subcategory of the category of multicategories.*

Remark 3.5. If a multicategory is malleable, we can reconstruct it up to isomorphism from its binary and nullary maps. When defining a malleable multicategory, it is usually easier to provide its binary, unary and nullary maps, and deduce from those the rest of the structure. The situation is similar in monoidal categories: we do not need to provide the n-ary tensor in order to define a monoidal category, we only provide the binary and unary tensors.

This suggests that we will really work with a *biased* version of malleable multicategories, one that privileges the binary and nullary tensors over the others. Biased malleable multicategories are better known as *promonoidal categories.*

3.1. Promonoidal Categories. In the same sense that multicategories provide an algebra for the composition of multiple pieces into one, promonoidal categories provide an algebra for the *coherent composition* of multiple pieces into one. A category $\mathbb{C}$ contains sets of *morphisms*, $\mathbb{C}(X; Y)$. In the same way, a promonoidal category $\mathbb{V}$ contains sets of *joints*, $\mathbb{V}(X_0 \triangleleft X_1; Y)$, *morphisms*, $\mathbb{V}(X; Y)$, and *units*, $\mathbb{V}(N; X)$, where N is the virtual tensor unit. Joints, $\mathbb{V}(X_0 \triangleleft X_1; Y)$, represent a way of joining objects of type X_0 and X_1 into an objects of type Y. Morphisms, $\mathbb{V}(X; Y)$, as in any category, are transformations of X into Y. Units, $\mathbb{V}(N; Y)$, are the atomic pieces of type Y.

These compositions must now be coherent. For instance, imagine we want to join X_0, X_1 and X_2 into Y. Joining X_0 and X_1 into something (•), and then joining that something (•) and X_2 into Y, *should be doable in essentially the same ways* as joining X_1 and X_2 into something (•), and then joining X_0 and that something (•) into Y. Formally, we are saying that,

$$\int^{U} \mathbb{V}(X_0 \triangleleft X_1; U) \times \mathbb{V}(U \triangleleft X_2; Y) \cong \int^{V} \mathbb{V}(X_1 \triangleleft X_2; V) \times \mathbb{V}(X_0 \triangleleft V; Y),$$

and, in fact, we usually just write $\mathbb{V}(X_0 \triangleleft X_1 \triangleleft X_2; Y)$ for the set of such decompositions, even when it is only defined up to isomorphism.

Definition 3.6. Promonoidal categories are the 2-monoids of the monoidal bicategory of profunctors, which is equivalent to the following definition. A *promonoidal category* is a category $\mathbb{V}(\bullet;\bullet)$ endowed with two profunctors

$$\mathbb{V}(\bullet \triangleleft \bullet; \bullet) \colon \mathbb{V} \times \mathbb{V} \to \mathbb{V}, \text{ and } \mathbb{V}(\mathsf{N}; \bullet) \colon 1 \to \mathbb{V}.$$

Equivalently, these are functors

$$\mathbb{V}(\bullet \triangleleft \bullet; \bullet) \colon \mathbb{V}^{op} \times \mathbb{V} \times \mathbb{V} \to \mathbf{Set}, \text{ and } \mathbb{V}(\mathsf{N}; \bullet) \colon \mathbb{V}^{op} \to \mathbf{Set}.$$

Moreover, promonoidal categories must be endowed with the following natural isomorphisms,

$$\begin{aligned}\mathbb{V}(X_0 \triangleleft X_1; \bullet) \diamond \mathbb{V}(\bullet \triangleleft X_2; Y) &\cong \mathbb{V}(X_1 \triangleleft X_2; \bullet) \diamond \mathbb{V}(X_0 \triangleleft \bullet; Y);\\ \mathbb{V}(\mathsf{N}; \bullet) \diamond \mathbb{V}(\bullet \triangleleft X; Y) &\cong \mathbb{V}(X; Y);\\ \mathbb{V}(\mathsf{N}; \bullet) \diamond \mathbb{V}(X \triangleleft \bullet; Y) &\cong \mathbb{V}(X; Y);\end{aligned}$$

called α, λ, ρ, respectively, and asked to satisfy the pentagon and triangle coherence equations, $\alpha \,⨾\, \alpha = (\alpha \diamond \mathrm{id}) \,⨾\, \alpha \,⨾\, (\mathrm{id} \diamond \alpha)$, and $(\rho \diamond \mathrm{id}) = \alpha \,⨾\, (\lambda \diamond \mathrm{id})$.

Definition 3.7 (Promonoidal functor)**.** Let $\mathbb{V}$ and $\mathbb{W}$ be two promonoidal categories. A *promonoidal functor* $F \colon \mathbb{V} \to \mathbb{W}$ is a functor between the two categories together with natural transformations

$$F_{\triangleleft} \colon \mathbb{V}(X_0 \triangleleft X_1; Y) \to \mathbb{W}(FX_0 \triangleleft FX_1; FY), \quad \text{and} \quad F_{\mathsf{N}} \colon \mathbb{V}(\mathsf{N}; X) \to \mathbb{W}(\mathsf{N}; Y),$$

that satisfy $\lambda ⨾ F_{map} = (F_{\triangleleft} \times F_N) ⨾ \lambda$, $\rho ⨾ F_{map} = (F_{\triangleleft} \times F_N) ⨾ \rho$, and $\alpha ⨾ (F_{\triangleleft} \times F_{\triangleleft}) ⨾ i = (F_{\triangleleft} \times F_{\triangleleft}) ⨾ i ⨾ \alpha$.

Proposition 3.8. *Promonoidal categories with promonoidal functors between them form a category,* **Prom***.*

3.2. Promonoidal Categories are Malleable Multicategories. In this section, we show that the category of promonoidal categories is equivalent to that of malleable multicategories. In this sense, the study of malleable multicategories is the study of promonoidal categories.

Definition 3.9 (Underlying malleable multicategory)**.** Let $\mathbb{V}$ be a promonoidal category. There is a malleable multicategory, $\mathbb{V}^m$, that has the same objects but multimorphisms defined by the elements of the promonoidal category. By induction, we define

$$\begin{aligned}&\mathbb{V}^m(X_0, X_1, \Gamma; Y) = \textstyle\int^V \mathbb{V}(X_0 \triangleleft X_1; V) \times \mathbb{V}^m(V, \Gamma; Y),\\ &\mathbb{V}^m(X; Y) = \mathbb{V}(X; Y),\\ &\mathbb{V}^m(; Y) = \mathbb{V}(\mathsf{N}; Y).\end{aligned}$$

In other words, the multimorphisms are elements of the left-biased tree reductions of the promonoidal category, seen as a 2-monoid. Dinatural composition is then defined to be the unique map relating two tree expressions in a 2-monoid, which exists uniquely by coherence,

$$(\mathrm{coh})\colon \left(\int^{X\in\mathbb{V}}\mathbb{V}(\Gamma;X)\times\mathbb{V}^m(\Gamma_0,X,\Gamma_1;Y)\right)\to\mathbb{V}^m(\Gamma_0,\Gamma,\Gamma_1;X).$$

Coherence maps are isomorphisms, and so dinatural composition is invertible, making the multicategory coherent. By coherence for pseudomonoids, composition must satisfy associativity and unitality.

Proposition 3.10. *The category of promonoidal categories and the category of malleable multicategories are equivalent with the functor $(\bullet)^m\colon$ **Prom** $\to$ **mMult** induced by the construction of the underlying malleable multicategory of a promonoidal category. See a polycategorical analogue at Proposition 2.16.*

Proof. First, let us show that a promonoidal functor, $F\colon\mathbb{V}\to\mathbb{W}$, induces a multifunctor, $F^m\colon\mathbb{V}^m\to\mathbb{W}^m$ between the underlying multicategories. On objects, we define it to be the same, $F^m_{obj}=F_{obj}$. On multimorphisms, we define the binary, unary and nullary cases using the promonoidal transformations:

$$F^u_0=F_N,\ \text{with } F^u_1=F_{map}\text{ and } F^u_2=F_{\triangleleft}.$$

Then, we extend this definition to the n-ary case by induction, using $F^u_n=F_{\triangleleft}\times F^u_{n-1}$. We now verify that this assignment is functorial: the only remarkable case is that of checking that the inductive case preserves composition.

$$\begin{aligned}(F\mathbin{⨾}G)^u_n &\overset{(i)}{=}(F\mathbin{⨾}G)_{\triangleleft}\times(F\mathbin{⨾}G)^u_{n-1}\overset{(ii)}{=}(F_{\triangleleft}\mathbin{⨾}G_{\triangleleft})\times(F^u_{n-1}\mathbin{⨾}G^u_{n-1})\\ &\overset{(iii)}{=}(F_{\triangleleft}\times F^u_{n-1})\times(G_{\triangleleft}\times G^u_{n-1})\overset{(iv)}{=}F^u_n\mathbin{⨾}G^u_n.\end{aligned}$$

This equation holds because *(i)* of the inductive definition, *(ii)* the composition of promonoidal functors and the inductive hypothesis, *(iii)* the interchange law for functions, *(iv)* and the inductive definition.

We will now show that this is a fully faithful functor. It is hopefully clear that it is faithful because $F^u=G^u$ directly implies $F_{obj}=G_{obj}$, $F_{\triangleleft}=G_{\triangleleft}$, $F_N=G_N$ and $F_{map}=G_{map}$. Let us show that it is also full. Let $G\colon\mathbb{V}_u\to\mathbb{W}_u$ be a multifunctor between malleable multicategories. We will construct a promonoidal functor $G^\star$ such that $G^{\star u}=G$. We start by defining that $G^\star_N=G_0$, that $G^\star_{map}=G_1$ and that $G^\star_{\triangleleft}=G_2$. Now, we need to prove that $(G^{\star u})_n=G_n$. This is definitionally true for binary, unary and nullary multimorphisms; then, by induction,

$$G_n\overset{(i)}{=}\mathrm{decomp}\mathbin{⨾}(G_2\times G_n)\overset{(ii)}{=}\mathrm{decomp}\mathbin{⨾}(G^{\star u}_2\times G^{\star u}_n)\overset{(iii)}{=}G^{\star u}_n.$$

Here, we use *(i)* malleability, *(ii)* the induction hypothesis, and *(iii)* the definition of the underlying multifunctor. We have shown that we have a fully faithful functor.

Finally, we will show that $U \colon \mathbf{Prom} \to \mathbf{mMult}$ is essentially surjective. Given any malleable multicategory $\mathbb{M}$, we can define $M^\flat$ to be the promonoidal category with the same objects and only binary, unary and nullary morphisms. We now note that there is an isomorphism, $\mathbb{M} \cong \mathbb{M}^{\flat u}$, that is the identity on objects. It is defined to use the invertible dinatural composition that exists by malleability,

$$\begin{aligned} \mathbb{M}^u_\flat(X_1, \dots, X_n; Y) &= \int^U \mathbb{M}(X_0, X_1; U) \times \mathbb{M}(U, X_2, \dots, X_n; Y) \\ &\cong \mathbb{M}(X_1, \dots, X_n; Y). \end{aligned}$$

This makes the functor fully faithful and essentially surjective, defining an equivalence of categories. □

3.3. Bibliography. What makes monoidal tensors universal? Products and coproducts have a universal property, but that is the exception and not the rule. Hermida's work [Her00] explains that tensors are universal because they represent a relevant multimap structure, a *multicategory.* For instance, the monoidal category of vector spaces with their tensor product represents functions linear in each variable: a linear function $A \otimes B \to C$ is the same as a multilinear function $A, B \to C$. Indeed, Lambek [Lam69] first introduced multicategories as the underlying structure that unified Gentzen's sequents and multilinear maps.

4. The Splice-Contour Adjunction

4.1. Contour of a multicategory. This last section characterizes the cofree malleable multicategory of *spliced arrows*, which governs how incomplete morphisms can be nested. Any multicategory freely generates another category, its *contour* [MZ22]. This can be interpreted as the category that tracks the processes of decomposition that the multicategory describes. The construction is particularly pleasant from the geometric point of view: it takes its name from the fact that it can be constructed by following the contour of the shape of the decomposition (Figure 4).

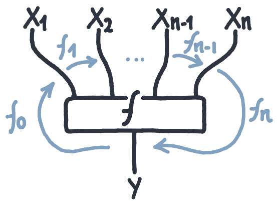


FIGURE 4. Contour of a multimorphism.

Definition 4.1. Let $\mathbb{M}$ be a multicategory. Its contour, $\mathsf{Contour}(\mathbb{M})$, is the category presented by two polarized objects, X° and $X^\bullet$, for each object $X \in \mathbb{M}_{obj}$;

(1) for each multimorphism $f \in \mathbb{M}(X_1, \dots, X_n; Y)$, the following generators,

$$f_0 \colon Y^\circ \to X_1^\circ; f_1 \colon X_1^\bullet \to X_2^\circ; \quad \dots; \quad f_{n-1} \colon X_{n-1}^\bullet \to X_n^\circ; f_n \colon X_n^\bullet \to Y^\bullet;$$

with only an $f_0 \colon Y^\circ \to Y^\bullet$ for the case $n = 0$;

(2) requiring contour to preserve identities, $(\mathrm{id}_X)_0 = \mathrm{id}_{X^\circ}$ and $(\mathrm{id}_X)_1 = \mathrm{id}_{X^\bullet}$;

(3) and requiring contour to preserve compositions, meaning that for each $f \in \mathbb{M}(X_1, \dots, X_n; Y_i)$ and each $g \in \mathbb{M}(Y_1, \dots, Y_m; Z)$, the contour of their composition is defined by the following five cases

$$(f \fatsemi_{X_i} g)_j = \begin{cases} g_j & \text{when } j < i, \\ g_i \fatsemi f_0 & \text{when } j = i, \\ f_{j-i} & \text{when } i < j < i + n, \\ f_n \fatsemi g_{i+1} & \text{when } j = i + n, \\ g_{j-n+1} & \text{when } i + n < j < n + m, \end{cases}$$

with the special case $(f \fatsemi_{X_i} g)_i = g_i \fatsemi f_0 \fatsemi g_{i+1}$ whenever $n = 0$.

A recent article by Melliès and Zeilberger [MZ22] develops the notion of a context-free grammar over a category as a multicategorical functor to the multicategory of *spliced arrows*. The multicategory of spliced arrows is a universal construction over a category that produces a multicategory of "contexts" over the category.

4.2. Spliced Arrows. The multicatgeory of spliced arrows is formed by arrows containing *blanks* or *holes* that could be filled to constuct a morphism. This multicategory gives an algebraic theory of context for the category.

Definition 4.2 (Spliced arrows)**.** Let $\mathbb{C}$ be a category. The multicategory of *spliced arrows* has objects pairs of objects in $\mathbb{C}$, and the multimorphisms are given by sequences of arrows in $\mathbb{C}$ separated by n gaps

$$\mathsf{Splice}\mathbb{C}\left({}^{X_1}_{Y_1}, \dots, {}^{X_n}_{Y_n}; {}^{X}_{Y}\right) = \mathbb{C}(X; X_1) \times \prod_{k=1}^{n-1} \mathbb{C}(Y_k, X_{k+1}) \times \mathbb{C}(Y_n; Y);$$

which we write as $f_0 \,⨾\, \square \,⨾\, \dots \,⨾\, \square \,⨾\, f_n$. That is, the sequence goes from X to Y, with holes typed by $\{X_i \to Y_i\}_{0<i\leq n}$. Composition is defined by substitution

$$\begin{aligned}(f_0 ⨾ \square ⨾ \dots ⨾ \square ⨾ f_n) \succ_i (g_0 ⨾ \square ⨾ \dots ⨾ \square ⨾ g_m) = \\ (g_0 ⨾ \square ⨾ \dots ⨾ g_i ⨾ f_0 ⨾ \square ⨾ \dots ⨾ \square ⨾ f_n ⨾ g_{i+1} ⨾ \dots ⨾ \square ⨾ g_m),\end{aligned}$$

and the identity morphism in $\binom{X}{Y}$ is $id_X ⨾ \square ⨾ id_Y$.

Proposition 4.3. *The multicategory of spliced arrows is a malleable multicategory.*

Proof. We will show that dinatural composition is invertible by exhibiting an inverse to the composition operation, up to dinaturality. Consider a spliced arrow $h_0 ⨾ \square \cdots \square ⨾ h_{n+m-1}$ with $(n+m+1)$ holes; we can decompose n of its holes at position i with the following operation.

$$\begin{aligned}&\mathrm{decomp}^n_i(h_0 ⨾ \square \cdots \square ⨾ h_{n+m-1}) \\ &\quad = (h_0 ⨾ \square \cdots h_i ⨾ \square ⨾ h_{i+n} \cdots \square ⨾ h_{n+m-1}) \mid (\mathrm{id} ⨾ \square ⨾ h_{i+1} \cdots h_{i+n-1} ⨾ \square ⨾ \mathrm{id}) \\ &\quad = (h_0 ⨾ \square \cdots \mathrm{id} ⨾ \square ⨾ \mathrm{id} \cdots \square ⨾ h_{n+m-1}) \mid (h_i ⨾ \square ⨾ h_{i+1} \cdots h_{i+n-1} ⨾ \square ⨾ h_{i+n}).\end{aligned}$$

This operation is an inverse to dinatural composition. It follows by construction that $(\succ)_i ⨾ \mathrm{decomp}_i = \mathrm{id}$, and we now check that $\mathrm{decomp}_i ⨾ (\succ)_i$ is an identity up to dinaturality. Let $f_0 ⨾ \square \cdots \square ⨾ f_n$ and $g_0 ⨾ \square \cdots \square ⨾ g_m$ be two spliced arrows,

$$\begin{aligned}&\mathrm{decomp}^n_i((f_0 ⨾ \square \dots \square ⨾ f_n) \succ_i (g_0 ⨾ \square \dots \square ⨾ g_m)) \\ &= \mathrm{decomp}^n_i(g_0 ⨾ \square \dots ⨾ g_i ⨾ f_0 ⨾ \square ⨾ \dots \square ⨾ f_n ⨾ g_{i+1} ⨾ \square \cdots \square ⨾ g_m)) \\ &= (g_0 ⨾ \dots ⨾ g_i ⨾ f_0 ⨾ \square ⨾ f_n ⨾ g_{i+1} ⨾ \dots ⨾ g_m) \mid (\mathrm{id} ⨾ \square ⨾ f_1 ⨾ \dots ⨾ f_{n-1} ⨾ \square ⨾ \mathrm{id}) \\ &= (g_0 ⨾ \dots ⨾ g_i ⨾ \square ⨾ g_{i+1} ⨾ \dots ⨾ g_m) \mid (f_0 ⨾ \square ⨾ f_1 ⨾ \dots ⨾ f_{n-1} ⨾ \square ⨾ f_n).\end{aligned}$$

We have shown that dinatural composition is invertible. □

4.3. Splice-Contour Adjunction. A first explicit account of splice-contour adjunction is due to Melliès and Zeilberger [MZ22]. In later joint work with Earnshaw and Hefford [EHR23], we showed that this adjunction produced not only multicategories but malleable multicategories.

THEOREM 4.4. *Splice is right adjoint to contour.*

PROOF. Let $\mathbb{M}$ be a multicategory. We will show that any multifunctor to a spliced arrow multicategory, $F\colon \mathbb{M} \to \mathsf{Splice}(\mathbb{C})$, factors through a canonical multifunctor $T\colon \mathbb{M} \to \mathsf{Splice}(\mathsf{Contour}(\mathbb{M}))$ that is followed by a unique functor $F^{\sharp}\colon \mathsf{Contour}(\mathbb{M}) \to \mathbb{C}$. First, we construct $T\colon \mathbb{M} \to \mathsf{Splice}(\mathsf{Contour}(\mathbb{M}))$, the multifunctor that sends any object X to the pair of polarized objects $\binom{X^{\circ}}{X^{\bullet}}$; and that sends any multimap $f \in \mathbb{M}(X_0,\dots,X_n;Y)$ to the spliced arrow

$$f_0 \mathbin{⨾} \square \mathbin{⨾} \dots \mathbin{⨾} \square \mathbin{⨾} f_n \in \mathsf{Splice}(\mathbb{C}) \begin{pmatrix} X_0^{\circ} \\ X_0^{\bullet} \end{pmatrix}, \dots, \begin{pmatrix} X_n^{\circ} \\ X_n^{\bullet} \end{pmatrix}; \begin{pmatrix} Y^{\circ} \\ Y^{\bullet} \end{pmatrix}.$$

We check now that T is indeed a multifunctor: by construction, it sends $T(f \mathbin{⨾}_{X_i} g) = f \succ_i g$ and it sends $T(\mathrm{id}_X) = (\mathrm{id}_{X^{\circ}} \mathbin{⨾} \square \mathbin{⨾} \mathrm{id}_{X^{\bullet}})$.

We will now show that there exists a unique functor $F^{\sharp}\colon \mathsf{Contour}(\mathbb{M}) \to \mathbb{C}$ factoring the multifunctor F, such that $F = T \mathbin{⨾} \mathsf{Splice}(F^{\sharp})$. The contour is a category presented by some generators and equations: to define a functor from it, it suffices to define it on the generators and show that it preserves the equations of the presentation. We do so next. Consider the objects, for each $X \in \mathbb{M}_{obj}$, assume $F(X) = \binom{A}{B}$. We must have that

$$\mathsf{Splice}(F^{\sharp})(T(X)) = \mathsf{Splice}(F^{\sharp})\begin{pmatrix} X^{\circ} \\ X^{\bullet} \end{pmatrix} = \begin{pmatrix} F^{\sharp}X^{\circ} \\ F^{\sharp}X^{\bullet} \end{pmatrix} = \begin{pmatrix} A \\ B \end{pmatrix},$$

which forces $F^{\sharp}(X^{\circ}) = A$ and $F^{\sharp}(X^{\bullet}) = B$. Consider now the morphisms, for each $f \in \mathbb{M}(X_0,\dots,X_n;Y)$. We must have that

$$\mathsf{Splice}(F^{\sharp})(T(f)) = \mathsf{Splice}(F^{\sharp})(f_0 \mathbin{⨾} \dots \mathbin{⨾} f_n) = F^{\sharp}f_0 \mathbin{⨾} \square \mathbin{⨾} \dots \mathbin{⨾} \square \mathbin{⨾} F^{\sharp}f_n,$$

which forces $F^{\sharp}(f_i) = F(f)_i$. This uniquely determines the value of $F^{\sharp}$ in all of the morphisms of the contour. We must finally check that that $F^{\sharp}$ satisfies the equations. We first notice that, by functoriality of F, we have

$$F^{\sharp}((f \mathbin{⨾}_{X_i} g)_j) = F(f \mathbin{⨾}_{X_i} g)_j = (F(f) \mathbin{⨾}_{FX_i} F(g))_j,$$

and, using this and the previous $F^{\sharp}(f_i) = F(f)_i$, we simply check the five cases of contour composition,

$$F^{\sharp}((f \mathbin{⨾}_{X_i} g)_j) = \left\{ \begin{array}{l} F(g)_j \\ F(g)_i \mathbin{⨾} F(f)_0 \\ F(f)_{j-i} \\ F(f)_n \mathbin{⨾} F(g)_{i+1} \\ F(g_{j-n+1}) \end{array} \right\} = \left\{ \begin{array}{l} F^{\sharp}(g_j) \\ F^{\sharp}(g_i) \mathbin{⨾} F^{\sharp}(f_0) \\ F^{\sharp}(f)_{j-i} \\ F^{\sharp}(f)_n \mathbin{⨾} F^{\sharp}(g)_{i+1} \\ F^{\sharp}(g)_{j-n+1} \end{array} \right\}.$$

This proves the existence of $F^\sharp$, but it also proves that it is the unique possible functor such that $F = T \fatsemi \mathsf{Splice}(F^\sharp)$. $\square$

4.4. Promonoidal Splice-Contour. We have commented on how any malleable multicategory induces a promonoidal category. The malleable multicategory of spliced arrows induces a promonoidal category of spliced arrows. This promonoidal category is precisely the one that arises from the adjunction between any category and its opposite category in the monoidal bicategory of profunctors.

Remark 4.5. The promonoidal splice could be seen as a particular case of the more general multicategorical splice. However, we will see that it is usually better behaved: technically, it is the 2-monoid arising from the 2-duality of a category with its opposite category, $\mathbb{C} \dashv \mathbb{C}^{op}$, in the monoidal bicategory of profunctors. We will not use this particular fact too much, but it will inspire its generalization. In the next chapter, we repeat a *monoidal* version of the splice-contour adjunction that, by default, uses only the malleable version.

Proposition 4.6 (Promonoidal spliced arrows). *Let $\mathbb{C}$ be a category. The promonoidal category of* spliced arrows, $\mathsf{Splice}\mathbb{C}$, *has as objects pairs of objects of $\mathbb{C}$. It uses the following profunctors to define morphisms, splits and units.*

(1) $\mathsf{Splice}(\mathbb{C})\left({X \atop Y}; {A \atop B}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y, B)$;
(2) $\mathsf{Splice}(\mathbb{C})\left({X \atop Y} \triangleleft {X' \atop Y'}; {A \atop B}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; B)$;
(3) $\mathsf{Splice}(\mathbb{C})\left(N; {A \atop B}\right) = \mathbb{C}(A; B)$.

In other words, morphisms are *pairs of arrows* $f \colon A \to X$ and $g \colon Y \to B$. Splits are *triples of arrows* $f \colon A \to X$, $g \colon Y \to X'$ and $h \colon Y' \to B$. Units are simply *arrows* $f \colon A \to B$. We use the following notation for

(1) morphisms, $(f \fatsemi \square \fatsemi g) \colon \left({X \atop Y}\right) \to \left({A \atop B}\right)$;
(2) joins, $(f \fatsemi \square \fatsemi g \fatsemi \square \fatsemi h) \colon \left({X \atop Y}\right) \triangleleft \left({X' \atop Y'}\right) \to \left({A \atop B}\right)$;
(3) and units, $f \colon \mathsf{N} \to \left({A \atop B}\right)$.

Definition 4.7. The *contour* of a promonoidal category $\mathbb{P}$ is the category $\mathsf{Contour}(\mathbb{P})$ presented by two polarized objects, X° and $X^\bullet$, for each object $X \in \mathbb{P}_{obj}$; and generated by the arrows that arise from *contouring* the decompositions of the promonoidal category.

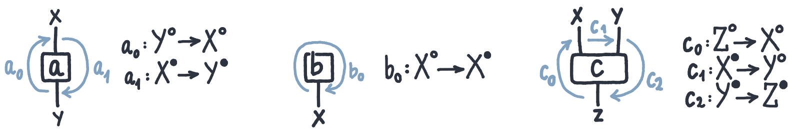


FIGURE 5. Contour of a promonoidal category.

Specifically, the contour is the category presented by the following generators, as depicted in Figure 5:

(1) $a_0 \colon Y^\circ \to X^\circ$ and $a_1 \colon X^\bullet \to Y^\bullet$, for each morphism $a \in \mathbb{P}(X; Y)$;
(2) $b_0 \colon X^\circ \to X^\bullet$, for each unit $b \in \mathbb{P}(N; X)$;
(3) a triple of generators, $c_0 \colon Z^\circ \to X^\circ$, $c_1 \colon X^\bullet \to Y^\circ$ and $c_2 \colon Y^\bullet \to Z^\bullet$, for each split $c \in \mathbb{P}(X \triangleleft Y; Z)$.

We impose several equations over these generators, all depicted in Figure 6. The equations come from the decomposition of categories and they form the theory of contour.

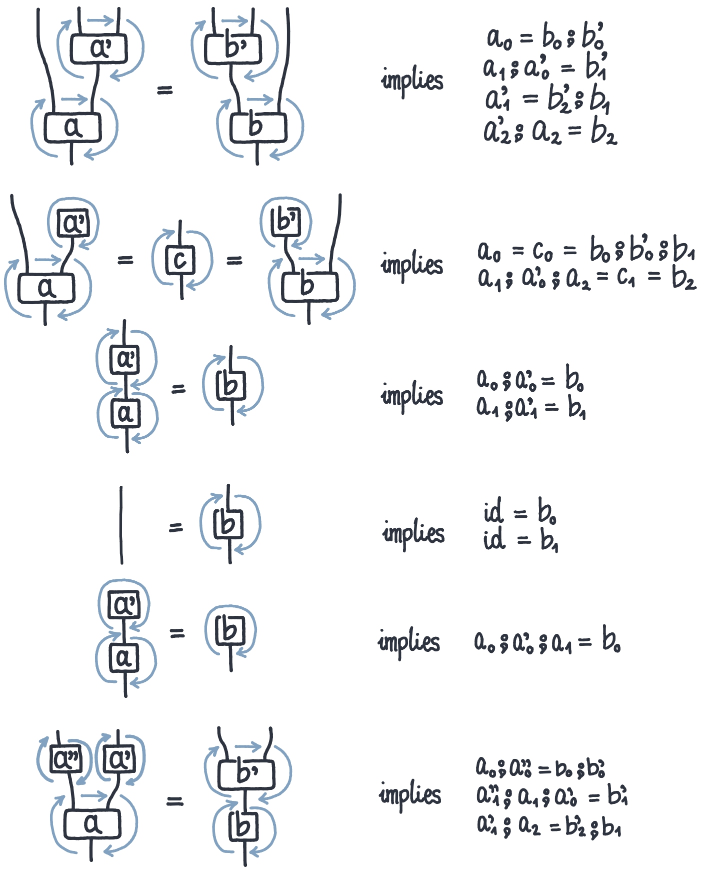


Figure 6. Theory of contour.

CHAPTER 3

Monoidal Context Theory

Monoidal Context Theory

This section develops a theory of context, or incomplete parts, for arbitrary monoidal categories. In the same way that the theory of context for categories required malleable multicategories or promonoidal categories; the theory of context for monoidal categories will require duoidal categories and produoidal categories.

Duoidal categories combine a sequential tensor ($\triangleleft$) with a parallel tensor ($\otimes$); and it is well known that they can be used for process description [GF16, SS22]. However, lifting context theory to monoidal categories will come with a few technical surprises that we develop; the most important one is a normalization monad in the category of produoidal categories: this becomes a crucial step in creating a theory of monoidal context that allows incomplete morphisms to take any shape. The morphisms of the produoidal category of contexts have been called *lenses* and *combs* in the literature, and we characterize them by a universal property.

We revisit the literature on duoidal categories and normalization in Sections 1 and 2. Produoidal categories and their splice-contour adjunction are introduced in Section 3. We take a technical aside in Section 4 to construct the normalization monad, and we immediately use it in Section 5.

1. Duoidal categories

1.1. Duoidal Categories. Duoidal categories result from the interaction of two monoidal categories. By the Eckmann-Hilton argument, each time we have two monoids $(*, \circ)$ such that one is a monoid homomorphism over the other, $(a \circ b) * (c \circ d) = (a * c) \circ (b * d)$, we know that both monoids coincide in a single commutative monoid.

However, an extra dimension helps us side-step the Eckmann-Hilton argument. If, instead of equalities or isomorphisms, we use directed morphisms, both monoids (which now may become 2-monoids) do not necessarily coincide, and the resulting structure is that of a duoidal category.

Definition 1.1 (Duoidal category)**.** A *duoidal category* [AM10] is a category $\mathbb{C}$ with two monoidal structures, $(\mathbb{C}, \otimes, I, \alpha, \lambda, \rho)$ and $(\mathbb{C}, \triangleleft, N, \beta, \kappa, \nu)$ such that the latter distribute over the former. In other words, it is endowed with a duoidal tensor, $(\triangleleft) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, together with natural distributors

$$\begin{aligned}
&\psi_2 \colon (X \triangleleft Z) \otimes (Y \triangleleft W) \to (X \otimes Y) \triangleleft (Z \otimes W),\\
&\psi_0 \colon I \to I \triangleleft I,\\
&\varphi_2 \colon N \otimes N \to N, \quad \text{and}\\
&\varphi_0 \colon I \to N,
\end{aligned}$$

satisfying the following coherence equations (Appendix 1, Figures 1 to 5). A duoidal category is *strict* when both of its monoidal structures are.

Remark 1.2. In other words, the duoidal tensor and unit are lax monoidal functors for the first monoidal structure, which means that the laxators must satisfy the following equations.

(1) $(\psi_2 \otimes id) \fatsemi \psi_2 \fatsemi (\alpha \triangleleft \alpha) = \alpha \fatsemi (id \otimes \psi_2) \fatsemi \psi_2$, for the associator;
(2) $(\psi_0 \otimes id) \fatsemi \psi_2 \fatsemi (\lambda \triangleleft \lambda) = \lambda$, for the left unitor; and
(3) $(id \otimes \psi_0) \fatsemi \psi_2 \fatsemi (\rho \triangleleft \rho) = \rho$, for the right unitor;
(4) $\alpha \fatsemi (id \otimes \varphi_2) \fatsemi \varphi_2 = (\varphi_2 \otimes id) \fatsemi \varphi_2$, for the associator;
(5) $(\varphi_0 \otimes id) \fatsemi \varphi_2 = \lambda$, for the left unitor; and
(6) $(id \otimes \varphi_0) \fatsemi \varphi_2 = \rho$, for the right unitor.

1.2. Communication via Duoidals. The operations of a posetal duoidal structure can be interpreted as speaking about the communication of processes [SS22]. Let $(\otimes, i)$ and $(\triangleleft, n)$ form a duoidal structure on a poset. We read the elements of this poset as being processes and we interpret

(1) $x \otimes y$ as "x and y happen together, in parallel and independently";
(2) $x \triangleleft y$ as "y happens after x, and may depend on it";
(3) i as a process that "interrupts communication";
(4) n as a process that "does nothing";
(5) $x \to y$ as "channels of x are included in channels of y".

Under this interpretation, the rules of a duoidal category say that

(1) $(x \triangleleft y) \otimes (z \triangleleft w) \to (x \otimes z) \triangleleft (y \otimes w)$, adds intermediate communication;
(2) $i \to i \triangleleft i$, allows to interrupt an interrupted process;
(3) $n \otimes n \to n$, simplifies parallelism that does nothing; and
(4) $i \to n$, allows new communications.

We can already picture this interpretation in terms of *communication diagrams.* Processes are boxes, and wires represent the information flow: a path from a process to another one means that the first can communicate to the second (Figure 1).

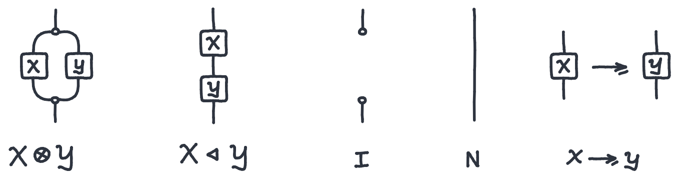


Figure 1. Communication diagrams.

The axioms of a duoidal category impose the following transformations of communication diagrams (Figure 2).

We can take these diagrams seriously: they are string diagrams of a monoidal bicategory where processes are endocells. The only structure that we ask of the single object generator is that of an *adjoint monoid.*

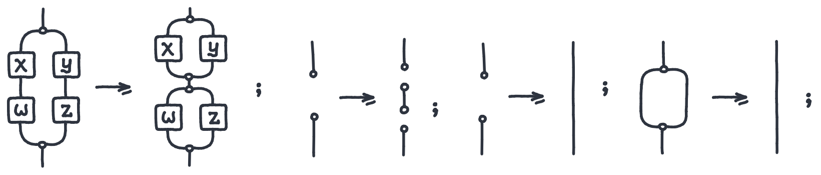


Figure 2. Communication diagram axioms.

1.3. Duoidals via adjoint monoids. Let $(\mathbb{C}, \otimes, I)$ be a monoidal category; let $(A, \curlyvee, ⫯)$ be a monoid and let $(B, \curlywedge, ⫰)$ be a comonoid. The set of morphisms $\mathbb{C}(A; B)$ forms a monoid with the operation of *convolution*, $f * g = \curlywedge \,⨟\, (f \otimes g) \,⨟\, \curlyvee$ and the biunit, $e = ⫰ \,⨟\, ⫯$, as in Figure 3.

Convolution and composition interact duoidally whenever the monoid and the comonoid are adjoint to each other. This is the notion of an *adjoint monoid.*

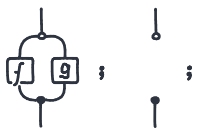


FIGURE 3. Convolution monoid.

Definition 1.3. An *adjoint monoid*, in a monoidal bicategory $\mathbb{B}$, is an object endowed with both 2-monoid and 2-comonoid structure (A, ⋎, ⫯, ⫰, ⋏), such that the multiplication is adjoint to the comultiplication (⋎ $\dashv$ ⋏) and the unit is adjoint to the counit (⫯ $\dashv$ ⫰).

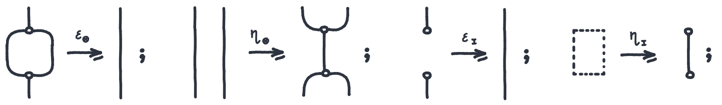


FIGURE 4. Adjoint monoid.

This means that there exist 2-cells, $\varepsilon_\otimes \colon$ ⋏ ⨾ ⋎ $\to$ id and $\eta_\otimes \colon$ id $\to$ ⋏ ⨾ ⋎, witnessing the adjunction (⋎ $\dashv$ ⋏); and that there exist 2-cells, $\varepsilon_I \colon$ ⫰ ⨾ ⫯ $\to$ id and $\eta_I \colon$ id $\to$ ⫰ ⨾ ⫯ witnessing the adjunction (⫯ $\dashv$ ⫰), as in Figure 4.

THEOREM 1.4 (Garner, López Franco [GF16]). *The endocells of an adjoint 2-monoid form a duoidal category with convolution and composition.*

PROOF. Let (A, ⋎, ⫯, ⫰, ⋏) be an adjoint monoid, and let $X, Y \in \mathbb{B}(A; A)$ be endocells. We define the sequential tensor as the composition, $X \triangleleft Y = X \mathbin{;} Y$, with the unit being the identity, $N = \mathbf{I}$. We define the parallel tensor as the convolution, $X \otimes Y =$ ⋏ ⨾ $(X \boxtimes Y)$ ⨾ ⋎, with the unit being the pair of monoid units, $I =$ ⫰⨾⫯. The duoidal interchangers are constructed out of the 2-cells of the adjoint monoid, taking Figure 2 seriously as the string diagrams of a monoidal bicategory.

(1) The first interchanger, $(X \triangleleft W) \otimes (Y \triangleleft Z) \to (X \otimes Y) \triangleleft (W \otimes Z)$, is constructed from the tensor adjunction unit, $\eta_\otimes \colon$ id $\to$ ⋎ ⨾ ⋏;
(2) the second interchanger, $I \to I \triangleleft I$, is constructed from the unit of the unit adjunction, $\eta_I \colon$ id $\to$ ⫯ ⨾ ⫰;
(3) the third interchanger, $I \to N$, is constructed from the counit of the unit adjunction, $\varepsilon_I \colon$ ⫰ ⨾ ⫯ $\to$ id; and
(4) the fourth interchanger, $N \otimes N \to N$, is constructed from the tensor adjunction counit, $\varepsilon_\otimes \colon$ ⋏ ⨾ ⋎ $\to$ id.

Finally, we need to check that all of the structure diagrams commute. This is usually left to the reader [GF16]; we can visualize the equations as surface diagrams. $\square$

Remark 1.5 (Day convolution)**.** A particular case is the convolution of two parallel profunctors $P, Q \colon \mathbb{C} \to \mathbb{D}$ between monoidal categories. A monoidal category determines an adjoint monoid in the monoidal bicategory of profunctors. Convolution is the operation that constructs a profunctor $P * Q \colon \mathbb{C} \to \mathbb{D}$ defined by

$$(P * Q)(A, B) = \int^{X,X',Y,Y'} \mathbb{C}(A, X \otimes Y) \times P(X, X') \times Q(Y, Y') \times \mathbb{D}(X' \otimes Y', B).$$

Whenever we particularize to presheaves over a monoidal category $\mathbb{C}$, we recover *Day convolution* of presheaves.

$$(F * G)(A) = \int^{X,Y} \mathbb{C}(A, X \otimes Y) \times F(X) \times G(Y).$$

In particular, the endoprofunctors over any monoidal category form a duoidal category, this is the duoidal category that we study in Section 3.

1.4. Be Careful with Duoidal Coherence. Monoidal categories possess a coherence theorem that determines that any two parallel morphisms constructed out of structure isomorphisms commute. In contrast, duoidal categories do not satisfy that same statement. This causes some confusion around coherence for duoidal categories. I bring an example of how this confusion may arise, hoping that it will help the interested reader and that it may further justify the importance of expository category theory.

We could be tempted to provide an alternative definition of duoidal categories that avoids asking for a bunch of commutative diagrams by simply asking that any formal such diagram commutes. In fact, this may possible for the physical duoidal categories studied by Spivak and Shapiro [SS22], who comment that

> *alternatively, duoidal categories can be defined by the two monoidal structures along with the generating structure maps [...] (4) natural in a, b, c, d which satisfy equations guaranteeing that any two structure maps built from those in (4) between the same two expressions in $y_\otimes, y_\triangleleft, \otimes, \triangleleft$ are equal.*

One of the first, most complete and comprehensive accounts of duoidal categories is the monograph by Aguiar and Mahajan [AM10]. It includes a passing comment that could suggest that this version can be proven correct. It says that

> *"[...] if two morphisms $A \to B$ are constructed out of the structure maps in $\mathbb{C}$ (including the structure constraints of the monoidal categories $(\mathbb{C}, \diamond, I)$ and $(\mathbb{C}, \star, J)$), then they coincide."*

However, intepreted literally and strictly, this turns out to not be true. Two parallel morphisms constructed out of the structure maps of a duoidal category do not need to coincide.

Proposition 1.6. *There exist two different maps of type $I \triangleleft I \to I$ constructed out of the structure maps of a duoidal category.*

Proof. We can consider two maps of type $I \triangleleft I \to I$, depending on which of the two parallel units we decide to convert to a sequential unit using the laxators. Explicitly, we are saying that $(I \triangleleft \varphi_0) \mathbin{;} \rho_\triangleleft$ and $(\varphi_0 \triangleleft I) \mathbin{;} \lambda_\triangleleft$ do not coincide; and the more intuitive string diagrams for bicategories for adjoint monoids confirm this (Figure 5). We will construct an explicit example of this phenomenon.

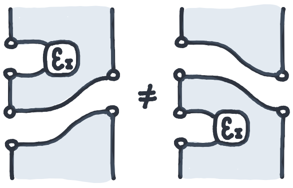


Figure 5. Bicategorical string diagrams for the two coherence maps.

Consider the duoidal category of endoprofunctors over a monoidal category. This is one of the first examples of duoidal category described by Street [Str12]; it is also described by Garner and López Franco [GF16], even when the axioms are not explicitly checked in print. In this category of endoprofunctors over $\mathbb{C}$, parallel tensor is the profunctor $I(X;Y) = \mathbb{C}(X;I) \times \mathbb{C}(I;Y)$, and sequencing two of them gives

$$(I \triangleleft I)(X;Y) = \hom(X;I) \times \hom(I;I) \times \hom(I;Y).$$

In this case, the two maps send the triple (f,a,g) to $(f \mathbin{;} a, g)$ and $(f, a \mathbin{;} g)$, respectively. However, these two pairs do not need to be equal if $a \in \hom(I;I)$ is a non-identity morphism. □

Example 1.7 (Graded spaces)**.** We look for a more classical source of examples in the theory of *graded spaces.* Let $(\mathbb{V}, \otimes, I)$ be a monoidal category with coproducts that are preserved by the tensor; let $(G, +, 0)$ be a commutative monoid. We say that the functor category $[G, \mathbb{V}]$ is the category of *G-graded $\mathbb{V}$-spaces.* This category has a rich structure; we highlight two of its tensor products: the *pointwise* or *Hadamard* tensor product

$$(V \otimes W)_n = V_n \otimes W_n, \text{ for each } n \in G, \text{ with unit } \mathbb{I}_n = I;$$

and the *convolution* or *Cauchy* tensor product

$$(V \bullet W)_n = \sum\nolimits_{k+m=n} V_k \otimes W_m, \text{ with unit } \mathbf{1}_n = \mathbf{0} \text{ except for } \mathbf{1}_0 = I.$$

These two tensors interact in a duoidal category with a laxator as follows; see for instance the work of López Franco and Vasilakopoulou [FV20].

$$\sum_{k+m=n} V_k \otimes W_k \otimes U_m \otimes Z_m \to \left(\sum_{k_1+m_1=n_1} V_{k_1} \otimes U_{m_1} \right) \otimes \left(\sum_{k_2+m_2=n_2} W_{k_2} \otimes Z_{m_2} \right).$$

Proposition 1.8. *Dually, there exist two different maps of type $J \to J \otimes J$ constructed out of the structure maps of a duoidal category.*

Proof. This follows from the previous Proposition 1.6, by considering the opposite duoidal category. However, let us comment a second example [AM]. Consider the duoidal category of graded spaces over a monoid G. The two maps, $\mathbb{I} \to \mathbb{I} \otimes 1 \to \mathbb{I} \otimes \mathbb{I}$ and $\mathbb{I} \to 1 \otimes \mathbb{I} \to \mathbb{I} \otimes \mathbb{I}$, correspond to inclusions of the vector space graded by $g \in G$ into the summand indexed by $(g, 0)$ or $(0, g)$, respectively; these are different in general. $\square$

In fact, the stronger statement of coherence does not seem to be used explicitly in any of these two texts, and the definition of duoidal categories as completely coherent strucutres is not usually found in the literature. Most authors, like Aguiar and Mahajan [AM10], and Garner and López Franco [GF16], revert to the definition of duoidal category as a 2-monoid in the monoidal bicategory of monoidal categories.

Aguiar and Mahajan [AM10] do actually point out that the expected coherence theorem should follow from the coherence theorem for lax monoidal functors. The confusion can arise if we do not realize that this coherence theorem does not actually prove that any two parallel maps coincide: in particular, coherence for lax monoidal functors does not prove that the two maps $F(I) \otimes F(I) \to F(I)$ coincide. In this case, however, the problem is better known – it is mentioned by Malkiewich and Ponto [MP21], who cite a short mention in the original proof by Lewis [Lew06] and Kelly and Laplaza [KL80].

1.5. Bibliography. I thank Marcelo Aguiar and Swapneel Mahajan [AM] for their generosity, helping me confirm the problem and specially for providing the second counterexample; I thank Brandon Shapiro and David Spivak for helping me follow this same idea on their work. I thank Matt Earnshaw for sharing his knowledge of the literature on duoidal categories.

2. Normal Duoidal Categories

Duoidal categories seem to contain too much structure: of course, we want to split things in two different ways, sequentially ($\triangleleft$) and in parallel ($\otimes$); but that does not necessarily mean that we want to keep track of two different types of units, parallel (I) and sequential (N). The atomic components of our decomposition algebra could be the same, without having to care if they are *atomic for sequential composition* or *atomic for parallel composition*; when this is the case, we talk of *normal duoidal categories.*

Definition 2.1. A *normal duoidal category* is a duoidal category in which the map $\varphi_0 \colon I \to N$ is an isomorphism.

While duoidal categories are useful to track communication between processes; symmetric normal duoidal categories track *dependencies* – whether a process' input depends on the output of another – structuring a dependency poset. This idea is explored by Garner and López Franco [GF16] and Spivak and Shapiro [SS22], and it has a formal counterpart in Theorem 2.9.

Most duoidal categories we have seen so far – and particularly those arising from adjoint monoids – have two different units. There exists a well-known abstract procedure that, starting from some duoidal category, constructs a new duoidal category that is normal: both units are identified. This procedure is known as *normalization*, and it can only be applied to duoidal categories with certain coequalizers preserved by the tensor.

2.1. Normalization of duoidal categories.

Garner and López Franco construct the normalization of a well-behaved duoidal category, using a new duoidal category of *bimodules* [GF16].

Remark 2.2. Let M be a bimonoid in the duoidal category $(\mathbb{V}, \otimes, I, \triangleleft, N)$, with maps $e \colon I \to M$ and $m \colon M \otimes M \to M$; and with maps $u \colon M \to N$ and $d \colon M \to M \triangleleft M$. Consider now the category of $M^{\otimes}$-bimodules. This category has a monoidal structure lifted from $(\mathbb{V}, \triangleleft, N)$:

(1) the unit, N, has a bimodule structure with

$$M \otimes N \otimes M \xrightarrow{u \otimes \mathrm{id} \otimes u} N \otimes N \otimes N \longrightarrow N;$$

(2) the sequencing of two $M^{\otimes}$-bimodules is a $M^{\otimes}$-bimodule with

$$\begin{aligned} &M \otimes (A \triangleleft B) \otimes M \\ &\quad \to (M \triangleleft M) \otimes (A \triangleleft B) \otimes (M \triangleleft M) \\ &\quad \to (M \otimes A \otimes M) \triangleleft (M \otimes B \otimes M) \to A \triangleleft B. \end{aligned}$$

Moreover, whenever $\mathbb{V}$ admits reflexive coequalizers preserved by ($\otimes$), the category of $M^{\otimes}$-bimodules is monoidal with the tensor of bimodules: the coequalizer

$$A \otimes M \otimes B \rightrightarrows A \otimes B \twoheadrightarrow A \otimes_M B.$$

In this case $(\mathbf{Bimod}^{\otimes}_M, \otimes_M, M, \triangleleft, N)$ is a duoidal category.

THEOREM 2.3 (Normalization of a duoidal, [GF16]). *Let $(\mathbb{V}, \otimes, I, \triangleleft, N)$ be a duoidal category with reflexive coequalizers preserved by $(\otimes)$. The category of N-bimodules is then a normal duoidal category,*

$$\mathcal{N}(\mathbb{V}) = (\mathbf{Bimod}^{\otimes}_N, \otimes_N, N, \triangleleft, N).$$

We call this category the normalization *of the duoidal category $\mathbb{V}$.*

2.2. Physical duoidal categories. The interaction of dependent and independent composition of normal duoidal categories is a recurrent idea in physical models: categorical models of spacetime exhibit this structure [HK22, SS22]; but it is also exhibited by parallel and sequentially composing programs [HS23]; or more simply, by the category of partially ordered sets.

In most of these cases, the normal duoidal category has an extra property: the parallel tensor $(\otimes)$ is symmetric. This is what motivates the name *physical duoidal category* for the $\otimes$-symmetric normal duoidal categories.

Definition 2.4. A *physical duoidal category* is a normal duoidal category endowed with a symmetric monoidal category structure for its parallel tensor.

Posets are a canonical example of a physical duoidal category: in fact, it is known that a subcategory of the category of posets and poset inclusions forms the free physical duoidal category over a generator. In that precise sense, duoidal expressions are dependency tracking posets.

Definition 2.5. The category of *poset shapes*, **PosetSh**, is the skeleton of the category of finite posets with bijective-on-objects monotone functions. Objects are isomorphism classes of finite posets, and morphisms are inclusions.

Proposition 2.6. Poset shapes *form a physical duoidal category.*

PROOF. The sequential tensor is constructed by sequentially joining the posets. Let $(P, \leq_P)$ and $(Q, \leq_Q)$ be two posets; their sequentiation, $P \triangleleft Q$, is a poset that contains a copy of P, a copy of Q, and an edge $p_i \leq q_j$ for each $p_i \in P$ and $q_j \in Q$; that is,

$$P \triangleleft Q = (P + Q, \leq_P + \leq_Q + \{p_i \leq q_j \mid p_i \in P, q_j \in Q\}).$$

The parallel tensor, $P \otimes Q$, is defined to be the disjoint union of posets, $P \otimes Q = (P + Q, \leq_P + \leq_Q)$, which defines a symmetric monoidal structure. The empty poset is the unit for both sequential and parallel tensoring, making **PosetSh** a physical duoidal category. $\square$

The category of poset shapes is not posetal: there are, for instance, two possible inclusions of the discrete two-element poset into itself. This prompts

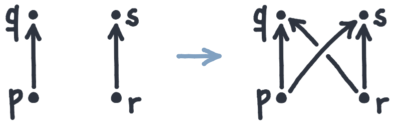


FIGURE 6. Poset inclusion.

us to label the nodes to indicate inclusions, as in Figure 6, but we work up to relabelling, or α-equivalence.

What makes this physical duoidal category particularly relevant is that it contains the free physical duoidal category over a generator. Every formal normal duoidal expression constructs a poset: we simply substitute each variable by the singleton poset and we interpret the expression in the duoidal category of poset shapes. Every formal structure map between normal duoidal expressions corresponds to an inclusion; for instance, Figure 6 documents the structure map $(p \triangleleft q) \otimes (r \triangleleft s) \to (p \otimes r) \triangleleft (q \otimes s)$.

Formalizing this result needs a bit of care, though: while all formal physical duoidal expressions correspond to posets, not every poset shape corresponds to a physical duoidal expression. The poset shapes that arise from applying duoidal operations to the singleton poset are called *expressible*, and we have a characterization result for them.

Definition 2.7. Expressible poset shapes are those inductively constructed from

(1) the empty poset, N;
(2) the singleton poset, $\{A\}$;
(3) the union of posets, $P \otimes Q = (P + Q, \leq_P + \leq_Q)$; and
(4) the sequencing of posets,
$$P \triangleleft Q = (P + Q, \leq_P + \leq_Q + \{p_i \leq q_j \mid p_i \in P, q_j \in Q\}).$$

Expressible poset shapes form a full duoidal subcategory of the physical duoidal category of poset shapes, **ExprSh**.

Proposition 2.8 (Grabowski, [Gra81])**.** *Not every poset shape is expressible. In fact, a poset shape is not expressible if and only if it admits an inclusion of the* Z-poset shape *defined by Figure 7.*

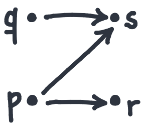


FIGURE 7. The Z poset shape.

Theorem 2.9 (Shapiro and Spivak, [SS22]). *The physical duoidal category* **ExprSh** *of expressible poset shapes is the free physical duoidal category on a single object. There exists exactly one structure map between any two objects of the free physical duoidal category for each inclusion of their associated expressible posets.*

Remark 2.10 (Coherence for normal duoidal categories)**.** Coherence for duoidal categories needs some care: not any two morphisms between distinctly-typed expressions in the free duoidal category are equal (Proposition 1.6). However, the previous theorem implies that the same statement is true for normal duoidal categories.

Corollary 2.11. *Any two morphisms between distinctly typed expressions in the free duoidal category over a set of objects are equal.*

2.3. Physical Lax Tensor of a Physical Duoidal Category. Let us recap our interpretation of physical duoidal categories: they track an underlying poset of dependencies. The sequential tensor, $X \triangleleft Y$, says that X occurs before Y, but $Y \triangleleft X$ says that Y occurs before X; consequently, it is not symmetric. The parallel tensor, $X \otimes Y$, states that both X and Y occur independently. This final section of our introduction to physical duoidal categories shows what happens when we want to consider both X and Y but we do not know at all how they interact: the tensor that tracks this case is a derived operation, the *physical tensor*, $X \boxtimes Y$.

The physical tensor simply says that both occur at some point: it does not impose independence, but it does not impose any particular dependency either. The physical tensor $X \boxtimes Y$ says that X may occur before Y, or Y before X, or both in parallel and in that case it does not matter how we regard the dependency. This is a tool that we will employ later to discuss a version of monoidal context that does not track dependency: *wiring diagrams* ([Spi13], Conjecture 5.11).

Remark 2.12. The binary physical tensor, $X \boxtimes Y$, is easy to define: it is the pushout of the two structure maps $X \otimes Y \to X \triangleleft Y$ and $X \otimes Y \to Y \triangleleft X$. However, unlike most tensors, its n-ary version cannot be deduced from its binary and nullary versions; the physical tensor is only a lax tensor.

Definition 2.13 (Leinster [Lei04])**.** A *lax monoidal category* is a category $\mathbb{C}$ endowed with a family of lax tensor n-fold tensor functors $(\boxtimes) \colon \mathbb{C}^n \to \mathbb{C}$ – written as $X_1 \boxtimes \ldots \boxtimes X_n$, with the 0-ary case E – and a family of associator natural transformations that unbias the application of the lax tensor,

$$\alpha \colon \boxtimes_{i=0}^{n} \left(\boxtimes_{j=0}^{k_i} X_j^i \right) \to X_1^1 \boxtimes \ldots \boxtimes X_{k_1}^1 \boxtimes \ldots \boxtimes X_1^n \boxtimes \ldots \boxtimes X_{k_n}^n,$$

such that all formally well-typed equations hold.

Definition 2.14. Let $(\mathbb{C}, \otimes, I, \triangleleft, N)$ be a physical duoidal category. The *physical tensor*, $(\boxtimes)$, is an additional lax monoidal tensor, defined as the glueing of the sequential tensor $(\triangleleft)$ along the parallel tensor $(\otimes)$; that is, it is the pushout on the following family of structure maps, indexed by permutations

$$l_\sigma \colon X_1 \otimes \ldots \otimes X_n \to X_{\sigma 1} \triangleleft \ldots \triangleleft X_{\sigma n}, \text{ for } \sigma \in P(n).$$

Remark 2.15. This only forms a lax tensor for a good reason. Consider the simpler case of three elements, $X \boxtimes Y \boxtimes Z$. This expression allows all of the possible six permutations to occur: *(i)* $X \triangleleft Y \triangleleft Z$; *(ii)* $X \triangleleft Z \triangleleft Y$; *(iii)* $Y \triangleleft X \triangleleft Z$; *(iv)* $Y \triangleleft Z \triangleleft X$; *(v)* $Z \triangleleft X \triangleleft Y$; and *(vi)* $Z \triangleleft Y \triangleleft X$. However, when we consider $(X \boxtimes Y) \boxtimes Z$, we are only allowing a certain subset of these cases to occur. Namely, only those where Z does not happen between X and Y: *(i)* $X \triangleleft Y \triangleleft Z$; *(ii)* $Y \triangleleft X \triangleleft Z$; *(iii)* $Z \triangleleft X \triangleleft Y$; and *(iv)* $Z \triangleleft Y \triangleleft X$. This is why the physical tensor is only lax. We have two inclusions: $(X \boxtimes Y) \boxtimes Z \to X \boxtimes Y \boxtimes Z$ and $X \boxtimes (Y \boxtimes Z) \to X \boxtimes Y \boxtimes Z$, but these are not isomorphisms.

Proposition 2.16. *The physical tensor defines a lax monoidal structure. Given any physical duoidal category* $(\mathbb{C}, \otimes, \triangleleft, N)$, *the physical tensor defines a lax monoidal category* $(\mathbb{C}, \boxtimes, N)$.

PROOF. The definition of the laxator follows from the universal property of the pushout; the coherence equations hold by uniqueness of the maps constructed out of this universal property. □

2.4. Bibliography. The original monograph on duoidal categories is due to Aguiar and Mahajan [AM10] – duoidal categories were originally known as *"2-monoidal categories"*; Street first described multiple examples that we recall [Str12], and Garner and López Franco mention for the first time the connection to adjoint monoids [GF16]. The reason duoidal categories do not have a correspondence in lower dimensional algebra is the Eckmann-Hilton argument [EH61, Theorem 1.12].

Physical duoidal categories follow the definition and nomenclature of Spivak and Shapiro [SS22]; their work makes the case for interpreting them as expressing dependencies between processes and argues initiality of the category of expressible posets. It seems originally due to Grabowski [Gra81] that expressible posets are precisely those not containing a Z, and Gischer recognized the lax interchange of normal duoidal categories as subsumption of posets [Gis88]. Even if the physical lax tensor does not seem to appear in the related literature, its definition and its consideration as a lax tensor follow the work of Leinster [Lei04] and the abstraction of commutativity by Garner and López Franco [GF16]. I thank Matt Earnshaw for multiple pointers to the literature.

3. Produoidal Decomposition of Monoidal Categories

3.1. Produoidal categories. Produoidal categories, first defined by Booker and Street [BS13], provide an algebraic structure for the interaction of sequential and parallel decomposition. A produoidal category $\mathbb{V}$ not only contains *morphisms*, $\mathbb{V}(X;Y)$, as in a category, but also *sequential joints*, $\mathbb{V}(X_0 \triangleleft X_1; Y)$, and *sequential units*, $\mathbb{V}(N;X)$, provided by a promonoidal structure; and *parallel joints*, $\mathbb{V}(X_0 \otimes X_1; Y)$ and *parallel units*, $\mathbb{V}(I;X)$, provided by another promonoidal structure.

These splits must be coherent. For instance, imagine we want to join X_0, X_1 and X_2 (sequentially) into Y. Joining X_0 and X_1 into something $(\bullet)$, and then joining that something with X_2 to produce Y *should be doable in essentially the same ways* as joining X_1 and X_2 into something $(\bullet)$, and then joining that something with X_0 to produce Y. Formally, we are saying that

$$\mathbb{V}(X_0 \triangleleft X_1; \bullet) \diamond \mathbb{V}(\bullet \triangleleft X_2; Y) \cong \mathbb{V}(X_1 \triangleleft X_2; \bullet) \diamond \mathbb{V}(X_0 \triangleleft \bullet; Y),$$

and, in fact, we just write $\mathbb{V}(X_0 \triangleleft X_1 \triangleleft X_2; Y)$ for the set of such transformations. This is precisely what we ask for in a promonoidal structure.

Definition 3.1 (Produoidal category)**.** A *produoidal category* is a category $\mathbb{V}$ endowed with two promonoidal structures,

$$\mathbb{V}(\bullet \otimes \bullet; \bullet) \colon \mathbb{V} \times \mathbb{V} \to \mathbb{V}, \text{ and } \mathbb{V}(I;\bullet) \colon 1 \to \mathbb{V},$$
$$\mathbb{V}(\bullet \triangleleft \bullet; \bullet) \colon \mathbb{V} \times \mathbb{V} \to \mathbb{V}, \text{ and } \mathbb{V}(N;\bullet) \colon 1 \to \mathbb{V},$$

such that one laxly distributes over the other. This is to say that it is endowed with the following natural *lax interchangers*:

(1) $\psi_2 \colon \mathbb{V}((X \triangleleft Y) \otimes (Z \triangleleft W); \bullet) \to \mathbb{V}((X \otimes Z) \triangleleft (Y \otimes W); \bullet)$,
(2) $\psi_0 \colon \mathbb{V}(I; \bullet) \to \mathbb{V}(I \triangleleft I; \bullet)$,
(3) $\varphi_2 \colon \mathbb{V}(N \otimes N; \bullet) \to \mathbb{V}(N; \bullet)$, and
(4) $\varphi_0 \colon \mathbb{V}(I; \bullet) \to \mathbb{V}(N; \bullet)$.

Interchangers, together with unitors and associators, must satisfy coherence conditions (see Appendix 1). We denote by **ProDuo** the category of produoidal categories and produoidal functors.

Proposition 3.2. *Let $\mathbb{V}$ be a produoidal category, then its category of copresheaves,* $[\mathbb{V}, \mathbf{Set}]$*, is a duoidal category.*

Remark 3.3 (Nesting profunctorial structures)**.** Notation for nesting functorial structures, say $(\triangleleft)$ and $(\otimes)$, is straightforward: we use expressions like $(X_1 \otimes Y_1) \triangleleft (X_2 \otimes Y_2)$ without a second thought. Nesting the profunctorial (or *virtual*) structures $(\triangleleft)$ and $(\otimes)$ is more subtle: defining $\mathbb{V}(X \otimes Y; \bullet)$ and $\mathbb{V}(X \triangleleft Y; \bullet)$ for each pair of objects X and Y does not itself define what something like $\mathbb{V}((X_1 \otimes Y_1) \triangleleft (X_2 \otimes Y_2); \bullet)$ means. Recall that, in the profunctorial case, $X_1 \triangleleft Y_1$

and $X_1 \otimes Y_1$ are not objects themselves: they are just names for the profunctors $\mathbb{V}(X_1 \lhd Y_1; \bullet)$ and $\mathbb{V}(X_1 \otimes Y_1; \bullet)$, which are not *representable.*

Instead, when we write $\mathbb{V}((X_1 \otimes Y_1) \lhd (X_2 \otimes Y_2); \bullet)$, we formally mean the composition of profunctors $\mathbb{V}(X_1 \otimes Y_1; \bullet_1) \diamond \mathbb{V}(X_2 \otimes Y_2; \bullet_2) \diamond \mathbb{V}(\bullet_1 \lhd \bullet_2; \bullet)$. By convention, nesting profunctorial structures means profunctor composition in this text.

Remark 3.4. Should we reverse the direction of the interchangers? Depending on the author, promonoidal categories and produoidal categories are reversed. It seems that both conventions have their advantages. The one we follow here [DPS05] makes intuitive sense: it follows the multicategorical and operadic point of view – multiple ingredients produce a result. The opposite one [EHR23] gets the interchangers to be those of a duoidal category and it becomes clear that there is a correspondence between produoidal categories and closed duoidal categories on preshaves.

3.2. Monoidal Contour of a Produoidal Category. Any produoidal category freely generates a monoidal category, its *monoidal contour.* Contours form a monoidal category of paths around the decomposition trees of the produoidal category. Contours follow a pleasant geometric pattern where we follow the shape of the decomposition, both in the parallel and sequential dimensions, to construct both sequential and parallel compositions for a monoidal category.

Definition 3.5 (Monoidal contour)**.** The *contour* of a produoidal category $\mathbb{B}$ is the monoidal category $\mathsf{mContour}(\mathbb{B})$ presented by two polarized objects, X° and $X^\bullet$, for each object $X \in \mathbb{B}_{\mathrm{obj}}$; and generated by arrows that arise from *contouring* both sequential and parallel decompositions of the promonoidal category.

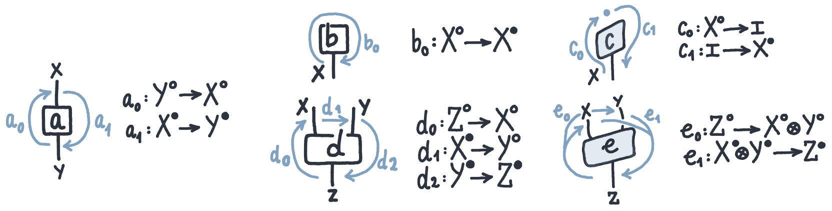


FIGURE 8. Generators of the monoidal category of contours.

Specifically, monoidal contour is the monoidal category presented by the following generators in Figure 8:

(1) $a_0 \colon Y^\circ \to X^\circ$ and $a_1 \colon X^\bullet \to Y^\bullet$, for each morphism $a \colon X \to Y$;
(2) $b_0 \colon X^\circ \to X^\bullet$, for each sequential unit, $b \colon \mathsf{N} \to X$;
(3) $c_0 \colon X^\circ \to I$ and $c_1 \colon I \to X^\bullet$, for each parallel unit, $c \colon I \to X$;

(4) a triple of generators $d_0 \colon Z^\circ \to X^\circ$, $d_1 \colon X^\bullet \to Y^\circ$ and $d_2 \colon Y^\bullet \to Z^\bullet$, for each sequential join $d \colon X \triangleleft Y \to Z$; and
(5) a pair of generators $e_0 \colon Z^\circ \to X^\circ \otimes Y^\circ$ and $e_1 \colon X^\bullet \otimes Y^\bullet \to Z^\bullet$ for each parallel join, $e \colon X \otimes Y \to Z$.

We impose all the equations of the theory of contour. Additionally, we also impose all of the equations depicted in Figure 9. Together, these form the theory of monoidal contour, which adds to the theory of sequential contour a new monoidal dimension.

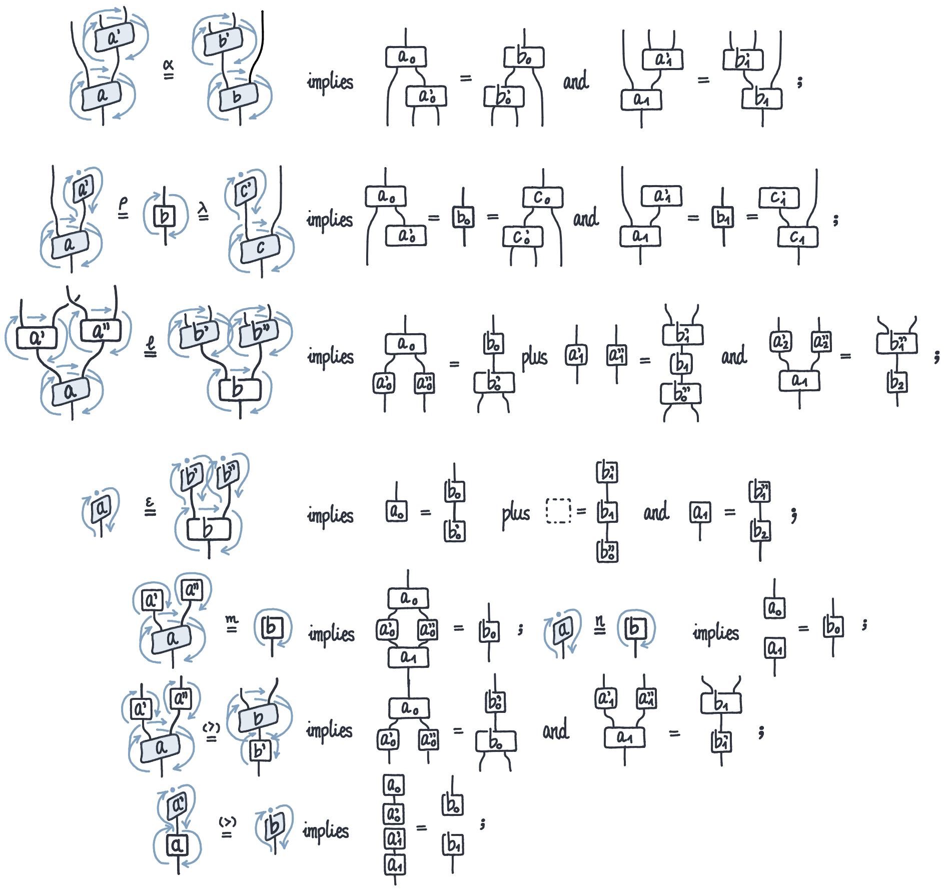


FIGURE 9. Extra equations for the theory of monoidal contour.

Proposition 3.6. *Monoidal contour extends to a functor*

$$\mathsf{mContour} : \mathbf{ProDuo} \to \mathbf{MonCat}.$$

Proof. Definition 3.5 defines how the functor acts on objects. We define the action on produoidal functors, the morphisms of the category of produoidal categories. Given a produoidal functor, $F : \mathbb{V} \to \mathbb{W}$, let us define the strict monoidal functor $\mathsf{mContour}(F) : \mathsf{mContour}(\mathbb{V}) \to \mathsf{mContour}(\mathbb{W})$ by the following morphism of generators:

1. objects X° and $X^\bullet$ are mapped to $F(X)^\circ$ and $F(X)^\bullet$;
2. for each $a \colon X \to Y$, the morphisms $a_0 : X^\circ \to X^\circ, a_1 : X^\bullet \to Y^\bullet$ are mapped to $F(a)_0$ and $F(a)_1$;
3. for each $b \colon I \to X$, both $b_0 : X^\circ \to I$ and $b_1 : I \to X^\bullet$ are mapped to $F_I(b)_0$ and $F_I(b)_1$;
4. for each $c \colon \mathsf{N} \to X$, the morphism $c_0 : X^\circ \to X^\bullet$ is mapped to $F_{\mathsf{N}}(c)_0$;
5. for each $d \colon X \triangleleft Y \to Z$, the morphisms $d_0 : Z^\circ \to X^\circ$, $d_1 : X^\bullet \to Y^\circ$ and $d_2 : Y^\bullet \to Z^\bullet$ are mapped to $F_\triangleleft(d)_0$, $F_\triangleleft(d)_1$ and $F_\triangleleft(d)_2$;
6. for each $e \colon X \triangleleft Y \to Z$, the morphisms $e_0 : Z^\circ \to X^\circ \otimes Y^\circ$, and $e_1 : X^\bullet \otimes Y^\bullet \to Z^\bullet$ are mapped to $F_\triangleleft(e)_0$, and $F_\triangleleft(e)_1$.

To show that this defines a morphism of presentations, we need to prove that the assignment of generators preserves the equations of the theory of contour, in Definition 4.1. Because $F : \mathbb{V} \to \mathbb{W}$ is a produoidal functor, the images of the generators do satisfy all of the contour equations of the target category. As a consequence, this assignment extends to a strict monoidal functor.

Finally, when $\mathrm{id}_{\mathbb{V}} : \mathbb{V} \to \mathbb{V}$ is an identity, the resulting functor is an identity because it is the identity on generators. Let $G : \mathbb{U} \to \mathbb{V}$ be another produoidal functor, then $\mathsf{mContour}(G \mathbin{⨾} F) = \mathsf{mContour}(G) \mathbin{⨾} \mathsf{mContour}(F)$ follows from the composition of produoidal functors. □

3.3. Produoidal Splice of a Monoidal Category. We want to go the other way around: given a monoidal category, what is the produoidal category that tracks the decomposition of arrows in that monoidal category? This subsection finds a right adjoint to the monoidal contour construction: the produoidal category of *spliced monoidal arrows.*

Definition 3.7. Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. Its produoidal category of *spliced monoidal arrows*, $\mathsf{mSplice}(\mathbb{C})$, has objects formed by pairs, $\mathsf{mSplice}(\mathbb{C})_{obj} = (\mathbb{C}^{op} \times \mathbb{C})_{obj}$, and is defined by the following profunctors, depicted in Figure 10.

1. $\mathsf{mSplice}(\mathbb{C})\ (\tfrac{X}{Y}; \tfrac{A}{B}) = \mathbb{C}(A; X) \times \mathbb{C}(Y, B)$,
2. $\mathsf{mSplice}(\mathbb{C})(\tfrac{X}{Y} \triangleleft \tfrac{X'}{Y'}; \tfrac{A}{B}) = \mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; B)$;
3. $\mathsf{mSplice}(\mathbb{C})(\tfrac{X}{Y} \otimes \tfrac{X'}{Y'}; \tfrac{A}{B}) = \mathbb{C}(A; X \otimes X') \times \mathbb{C}(Y \otimes Y'; B)$;
4. $\mathsf{mSplice}(\mathbb{C})(\mathsf{N}; \tfrac{A}{B};) = \mathbb{C}(A; B)$;
5. $\mathsf{mSplice}(\mathbb{C})(I; \tfrac{A}{B}) = \mathbb{C}(A; I) \times \mathbb{C}(I; B)$.

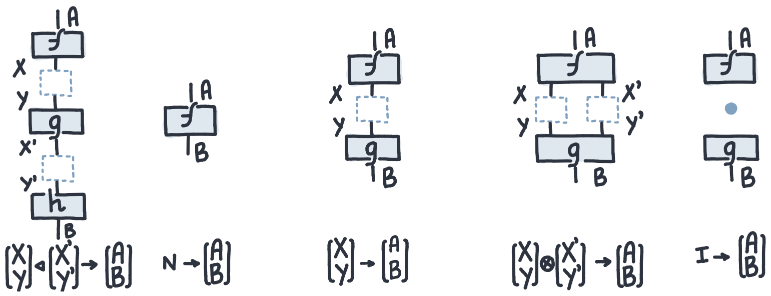


Figure 10. Spliced monoidal arrows.

Proposition 3.8. *Spliced monoidal arrows indeed form a produoidal category.*

Proof sketch. The complete proof constructs all of the necessary natural isomorphisms using coend calculus. We refer to joint work of this author with Earnshaw and Hefford, where the equations are proven in full detail [EHR23]. □

Remark 3.9. The produoidal algebra of spliced arrows is a natural construction: abstractly, we know that there exists a duoidal structure on the endomodules of any monoidal category [Day70, Str12] – monoidal spliced arrows form its explicitly constructed produoidal counterpart. What may be more surprising is that spliced arrows have themselves a universal property as part of an adjunction.

Theorem 3.10. *Spliced monoidal arrows form a produoidal category with their sequential and parallel splits, units, and suitable coherence morphisms and laxators. Spliced monoidal arrows extend to a functor* $\mathsf{mSplice} : \mathbf{MonCat} \to \mathbf{ProDuo}$. *The monoidal contour and the produoidal splice are left and right adjoints to each other, respectively.*

Proof. Monoidal contour $\mathsf{mContour}(\mathbb{B})$ is presented by generators and equations: to specify a strict monoidal functor $\mathsf{mContour}(\mathbb{B}) \to \mathbb{M}$, it is enough to specify images of the generators and then prove that they satisfy the equations.

Let $(\mathbb{M}, \otimes_M, I_M)$ be a monoidal category. Then a strict monoidal functor $\mathsf{mContour}(\mathbb{B}) \to \mathbb{M}$ amounts to the following data satisfying some extra conditions.

(1) For each object $X \in \mathbb{B}_{\mathrm{obj}}$, a pair of objects $X^\circ, X^\bullet \in \mathbb{M}_{\mathrm{obj}}$;
(2) for each element $f \colon \mathsf{N} \to X$, a morphism $f_0 \colon X^\circ \to X^\bullet$;
(3) for each unit $f \colon I \to X$, a choice of $f_0 \colon X^\circ \to I$ and $f_1 \colon I \to X^\bullet$;
(4) for each morphism $f \colon X \to Y$, a choice of $f_0 \colon Y^\circ \to X^\circ$ and $f_1 \colon X^\bullet \to Y^\bullet$;
(5) for each sequential split $f \colon X \triangleleft Y \to Z$, a choice of morphisms $f_0 \colon Z^\circ \to X^\circ$, plus $f_1 \colon X^\bullet \to Y^\circ$ and $f_2 \colon Y^\bullet \to Z^\bullet$;

(6) for each parallel split $f\colon X \otimes Y \to Z$, a choice of morphisms $f_0\colon Z^\circ \to X^\circ \otimes Y^\circ$ and $f_1\colon X^\bullet \otimes Y^\bullet \to Z^\bullet$.

In order to construct a well-defined strict monoidal functor, the previous assignments must satisfy the following conditions for each one of the two promonoidal categories:

(1) $\alpha(a \fatsemi_1 b) = (c \fatsemi_2 d)$ in the promonoidal category implies $a_0 \fatsemi (b_0 \otimes \mathrm{id}) = c_0 \fatsemi (\mathrm{id} \otimes d_0)$ and $(b_1 \otimes \mathrm{id}) \fatsemi a_1 = (\mathrm{id} \otimes d_1) \fatsemi c_1$ in the monoidal category;
(2) $\lambda(a \fatsemi_1 b) = c = \rho(d \fatsemi_2 e)$ in the promonoidal category implies $a_0 \fatsemi (b_0 \otimes \mathrm{id}) = c_0 = d_0 \fatsemi (\mathrm{id} \otimes e_0)$ and $(b_1 \otimes \mathrm{id}) \fatsemi a_1 = c_1 = (\mathrm{id} \otimes e_1) \fatsemi d_1$ in the monoidal category;

Moreover, they must also satisfy the following conditions for the produoidal category.

(1) $\psi_2(a \mid b \mid c) = (d \mid e \mid f)$ in the promonoidal category implies $a_0 \fatsemi (b_0 \otimes c_0) = d_0 \fatsemi e_0, b_1 \otimes c_1 = e_1 \fatsemi d_1 \fatsemi f_0$ and $(b_2 \otimes c_2) \fatsemi a_1 = f_1 \fatsemi d_2$ in the monoidal category;
(2) $\psi_0(a) = (b \mid c \mid d)$ in the promonoidal category implies $a_0 = b_0 \fatsemi c_0$, $\mathrm{id} = c_1 \fatsemi b_1 \fatsemi d_0$, and $a_1 = d_1 \fatsemi b_2$ in the monoidal category;
(3) $\varphi_2(a \mid b \mid c) = d$ in the promonoidal category implies $a_0 \fatsemi (b_0 \otimes c_0) \fatsemi a_1 = d_0$ in the monoidal category;
(4) $\varphi_0(a) = b$ in the promonoidal category implies $a_0 \fatsemi a_1 = b_0$ in the monoidal category.

On the other hand, a produoidal functor $\mathbb{B} \to \mathsf{mSplice}(\mathbb{M})$, also amounts to the following data. We will state it in multiple points and finally confirm that each one of these points has a correspondence on the first part of the proof, finishing the definition.

(1) For each object $X \in \mathbb{B}_{\mathrm{obj}}$, an object $(X^\circ, X^\bullet) \in \mathsf{mSplice}(\mathbb{M})_{\mathrm{obj}}$;
(2) for each element $f\colon \mathsf{N} \to X$, a morphism $f_0\colon \mathsf{N} \to \binom{X^\circ}{X^\bullet}$;
(3) for each unit, $f\colon I \to X$, a unit $\langle f_0 \parallel f_1 \rangle\colon I \to \binom{X^\circ}{X^\bullet}$;
(4) for each morphism $f\colon X \to Y$, a splice $\langle f_0 \fatsemi \square \fatsemi f_1 \rangle\colon \binom{X^\circ}{X^\bullet} \to \binom{Y^\circ}{Y^\bullet}$;
(5) for each seq. join $f\colon X \triangleleft Y \to Z$, a spliced arrow
$$\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2 \rangle\colon \binom{X^\circ}{X^\bullet} \triangleleft \binom{Y^\circ}{Y^\bullet} \to \binom{Z^\circ}{Z^\bullet};$$
(6) for each par. join $f\colon X \otimes Y \to Z$, a spliced arrow
$$\langle f_0 \fatsemi \square \otimes \square \fatsemi f_1 \rangle\colon \binom{X^\circ}{X^\bullet} \otimes \binom{Y^\circ}{Y^\bullet} \to \binom{Z^\circ}{Z^\bullet}.$$

The following conditions must hold for each one of the promonoidal categories. These correspond definitionally to the conditions for the two promonoidal structures we imposed before.

(1) $\alpha(a \mid b) = (c \mid d)$ in the produoidal category implies $\alpha(Fa \mid Fb) = (Fc \mid Fd)$ for the spliced arrows;

(2) $\lambda(a \mid b) = c = \rho(d \mid e)$ in the produoidal category implies $\lambda(Fa \mid Fb) = Fc = \rho(Fd \mid Fe)$ for the spliced arrows.

Finally, all the following conditions must also hold for the produoidal category. These are definitionally equal to the conditions for the produoidal category we imposed before.

(1) $\psi_2(a \mid b \mid c) = (d \mid e \mid f)$ in the produoidal category implies
$$\psi_2(Fa \mid Fb \mid Fc) = (Fd \mid Fe \mid Ff);$$
(2) $\psi_0(a) = (b \mid c \mid d)$ in the produoidal category implies
$$\psi_0(Fa) = (Fa \mid Fc \mid Fd);$$
(3) $\varphi_2(a \mid b \mid c) = d$ in the produoidal category implies
$$\varphi_2(Fa \mid Fb \mid Fc) = Fd;$$
(4) $\varphi_0(a) = b$ in the produoidal category implies $\varphi_0(Fa) = Fb$.

Each of these points is exactly equal by definition to the relative point in the first part of the proof: this establishes the desired adjunction. In other words, the definition of monoidal splice is precisely the one that makes this proof hold definitionally – even when we gave multiple different characterizations of it. □

3.4. A Representable Parallel Structure. A produoidal category has two tensors, and neither is, in principle, representable. However, the cofree produoidal category over a category we have just constructed happens also to have a representable tensor, $(\otimes)$: spliced monoidal arrows form a monoidal category.

Remark 3.11. This means $\mathsf{mSplice}(\mathbb{C})$ has the structure of a *virtual duoidal category* [Shu17] or *monoidal multicategory*, defined by Aguiar, Haim and López Franco [AHLF18] as a pseudomonoid in the cartesian monoidal 2-category of multicategories.

Proposition 3.12. *Parallel joins and parallel units of spliced monoidal arrows are representable profunctors. Explicitly,*
$$\mathsf{mSplice}(\mathbb{C})\left(\tbinom{X}{Y} \otimes \tbinom{X'}{Y'}; \tbinom{A}{B}\right) \cong \mathsf{mSplice}(\mathbb{C})\left(\tbinom{X \otimes X'}{Y \otimes Y'}; \tbinom{A}{B}\right), \text{ and}$$
$$\mathsf{mSplice}(\mathbb{C})\left(I; \tbinom{A}{B}\right) \cong \mathsf{mSplice}(\mathbb{C})\left(\tbinom{I}{I}; \tbinom{A}{B}\right).$$

In fact, these sets are equal by definition. However, we argue that there is a reason to work in the full generality of produoidal categories: produoidal categories can always be *normalized.*

Remark 3.13. Normalization is a procedure to mix both tensors of a duoidal category, $(\otimes)$ and $(\triangleleft)$, but not every duoidal category has a normalization [GF16]. It is folklore that one loses nothing by regarding non-representable produoidal structures as representable *duoidal structures on presheaves*, dismissing that they are moreover *closed* [Day70]; thus, one would expect only some produoidal categories to be normalizable – after all, only some duoidal categories are. Against folklore, we prove that every produoidal category, representable or not, has a *universal normalization*, a normal produoidal category which may be again representable or not.

3.5. Bibliography. Motivated by language theory and the representation theorem of Chomsky and Schützenberger, Melliès and Zeilberger [MZ22] were the first to present the multicategorical *splice-contour* adjunction. We are indebted to their exposition, which we extend to the promonoidal and produoidal cases. Our contribution is to show how monoidal contexts arise from an extended produoidal splice-contour adjunction; unifying these two threads.

Street already noted that the endoprofunctors of a monoidal category had a duoidal structure [Str12]; Pastro and Street described a promonoidal structure on lenses [PS07] and Garner and López-Franco contributed a partial normalization procedure for duoidal categories [GF16]. We build on top of this literature, putting it together, spelling out existence proofs, popularizing its produoidal counterpart and providing multiple new results and constructions that were previously missing (e.g. Theorems 3.10, 4.6 and 5.3).

This section takes its main ideas from joint work of this author with Matt Earnshaw and James Hefford [EHR23]. Earnshaw and Sobociński [ES22] have described a syntactic congruence on formal languages of string diagrams using monoidal contexts.

4. Interlude: Produoidal Normalization

4.1. Normal Produoidal Categories. Produoidal categories seem to contain too much structure: of course, we want to split things in two different ways, sequentially ($\triangleleft$) and in parallel ($\otimes$); but that does not necessarily mean that we want to keep track of two different types of units, parallel (I) and sequential (N). The atomic components of our decomposition algebra should be the same, without having to care if they are *atomic for sequential composition* or *atomic for parallel composition.*

Remark 4.1. The monoidal spliced arrows we just introduced are a perfect example: if we simply want a hole in a string diagram, the type it may take depends on the wires we want to leave to each side (see Figure 35). We would prefer to construct a new category – a Kleisli category on top of this one – where the bureaucracy of the units was already handled for us.

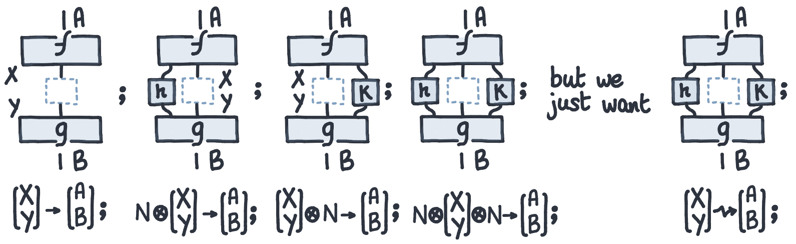


FIGURE 11. Multiple units complicate types.

4.2. The Normalization Monad. Fortunately, there exists an abstract procedure that, starting from any produoidal category, constructs a new produoidal category where both units are identified. This procedure is known as *normalization*, and the resulting produoidal categories are called *normal.*

Definition 4.2 (Normal produoidal category)**.** A *normal produoidal category* is a produoidal category where the unit interchanger $n \colon \mathbb{V}(I; \bullet) \to \mathbb{V}(\mathsf{N}; \bullet)$ is an isomorphism.

Normal produoidal categories form a category with produoidal functors between them, **nProDuo**. As a consequence, it is endowed with a fully faithful forgetful functor $\mathcal{U} \colon$ **nProDuo** $\to$ **ProDuo**.

THEOREM 4.3. *Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a produoidal category. The profunctor*

$$\mathsf{Nor}(\mathbb{V})(\bullet;\bullet) = \mathbb{V}(\mathsf{N} \otimes \bullet \otimes \mathsf{N}; \bullet)$$

forms a promonad. Moreover, the Kleisli category of this promonad is a normal produoidal category with the following profunctors.

(1) $\mathsf{Nor}(\mathbb{V})(X \otimes_{\mathsf{N}} Y; Z) = \mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N} \otimes Y \otimes \mathsf{N}; Z)$*;*
(2) $\mathsf{Nor}(\mathbb{V})(X \triangleleft_{\mathsf{N}} Y; Z) = \mathbb{V}((\mathsf{N} \otimes X \otimes \mathsf{N}) \triangleleft (\mathsf{N} \otimes Y \otimes \mathsf{N}); Z)$*; and*
(3) $\mathsf{Nor}(\mathbb{V})(I_{\mathsf{N}}; X) = \mathsf{Nor}(\mathbb{V})(\mathsf{N}_{\mathsf{N}}; X) = \mathbb{V}(\mathsf{N}; X)$.

Proof. Let us prove that $\mathsf{Nor}(\mathbb{V})$ is a promonad. We now define the multiplication and unit for the promonad, $\mathsf{Nor}(\mathbb{V})$. They are constructed out of laxators of the produoidal category $\mathbb{V}$ and Yoneda isomorphisms; thus, they must be associative and unital by coherence. The unit is defined by *(i)* unitality of $\mathbb{V}$, *(ii)* the laxator of $\mathbb{V}$, and *(iii)* by definition of $\mathsf{Nor}(\mathbb{V})$.

$$\mathbb{V}(X;Y) \cong \mathbb{V}(I \otimes X \otimes I; Y) \to \mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N}; Y) \cong \mathsf{Nor}(\mathbb{V})(X;Y).$$

Multiplication is constructed as follows, using *(i)* the definition of $\mathsf{Nor}(\mathbb{V})$, *(ii)* Yoneda reduction, *(iii)* the laxators of $\mathbb{V}$, and *(iv)* the definition of $\mathsf{Nor}(\mathbb{V})$.

$$\begin{aligned}\int^{Y \in \mathbb{V}} \mathsf{Nor}(\mathbb{V})(X;Y) \times \mathsf{Nor}(\mathbb{V})(Y;Z) &= \int^{Y \in \mathbb{V}} \mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N}; Y) \times \mathbb{V}(\mathsf{N} \otimes Y \otimes \mathsf{N}; Z) \\ &\cong \mathbb{V}(\mathsf{N} \otimes \mathsf{N} \otimes X \otimes \mathsf{N} \otimes \mathsf{N}; Y) \to \mathbb{V}(\mathsf{N} \otimes Y \otimes \mathsf{N}; Z) = \mathsf{Nor}(\mathbb{V})(X;Z).\end{aligned}$$

Associativity and unitality follow from those of the produoidal category.

The second part of this proof will show that $\mathsf{Nor}(\mathbb{V})$ is indeed a produoidal category: we will construct its associators, unitors and laxators from those of $\mathbb{V}$ and Yoneda isomorphisms. The right unitor is constructed as follows; the left unitor is constructed in a similar way: *(i)* by definition of $\mathsf{Nor}(\mathbb{V})$, *(ii)* by associativity, *(iii)* by definition, *(iv)* by Yoneda reduction, *(v)* by definition, and *(vi)* by unitality.

$$\begin{aligned}&\int^{M \in \mathsf{Nor}(\mathbb{V})} \mathsf{Nor}(\mathbb{V})(\mathsf{N}; M) \times \mathsf{Nor}(\mathbb{V})(X \otimes_{\mathsf{N}} M; Y) && = \\ &\int^{M \in \mathsf{Nor}(\mathbb{V})} \mathsf{Nor}(\mathbb{V})(\mathsf{N}; M) \times \mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N} \otimes M \otimes \mathsf{N}; Y) && \cong \\ &\int^{M \in \mathsf{Nor}(\mathbb{V}), P \in \mathbb{V}} \mathsf{Nor}(\mathbb{V})(\mathsf{N}; M) \times \mathbb{V}(\mathsf{N} \otimes M \otimes \mathsf{N}; P) \times \mathbb{V}(\mathsf{N} \otimes X \otimes P; Y) && = \\ &\int^{M \in \mathsf{Nor}(\mathbb{V}), P \in \mathbb{V}} \mathsf{Nor}(\mathbb{V})(\mathsf{N}; M) \times \mathsf{Nor}(\mathbb{V})(M; P) \times \mathbb{V}(\mathsf{N} \otimes X \otimes P; Y) && \cong \\ &\int^{P \in \mathbb{V}} \mathsf{Nor}(\mathbb{V})(\mathsf{N}; P) \times \mathbb{V}(\mathsf{N} \otimes X \otimes P; Y) && = \\ &\int^{P \in \mathbb{V}} \mathbb{V}(\mathsf{N}; P) \times \mathbb{V}(\mathsf{N} \otimes X \otimes P; Y) && \cong \\ &\mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N}; Y).\end{aligned}$$

Let us now construct the associator in two steps: we will show that both sides of the following equation

$$\begin{aligned}&\int^{P \in \mathsf{Nor}(\mathbb{V})} \mathsf{Nor}(\mathbb{V})(Y \otimes_{\mathsf{N}} Z; P) \times \mathsf{Nor}(\mathbb{V})(X \otimes_{\mathsf{N}} P; A) \cong \\ &\int^{M \in \mathsf{Nor}(\mathbb{V})} \mathsf{Nor}(\mathbb{V})(X \otimes_{\mathsf{N}} Y; M) \times \mathsf{Nor}(\mathbb{V})(M \otimes_{\mathsf{N}} Z; A)\end{aligned}$$

are isomorphic to $\mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N} \otimes Y \otimes \mathsf{N} \otimes Z \otimes \mathsf{N}; A)$. The first side is isomorphic by *(i)* definition of $\mathsf{Nor}(\mathbb{V})$, *(ii)* associativity of $\mathbb{V}$, *(iii)* definition of $\mathsf{Nor}(\mathbb{V})$, *(iv)*

Yoneda reduction, *(v)* definition of $\mathsf{Nor}(\mathbb{V})$, and *(vi)* associativity. The second side is analogous.

$$\begin{aligned}
&\textstyle\int^{M\in\mathsf{Nor}(\mathbb{V})}\mathsf{Nor}(\mathbb{V})(Y\otimes_{\mathsf{N}} Z;M)\times\mathsf{Nor}(\mathbb{V})(X\otimes_{\mathsf{N}} M;A) &&=\\
&\textstyle\int^{M\in\mathsf{Nor}(\mathbb{V})}\mathsf{Nor}(\mathbb{V})(Y\otimes_{\mathsf{N}} Z;M)\times\mathbb{V}(\mathsf{N}\otimes X\otimes\mathsf{N}\otimes M\otimes\mathsf{N};A) &&\cong\\
&\textstyle\int^{M\in\mathsf{Nor}(\mathbb{V}),P\in\mathbb{V}}\mathsf{Nor}(\mathbb{V})(Y\otimes_{\mathsf{N}} Z;M)\times\mathbb{V}(\mathsf{N}\otimes M\otimes\mathsf{N};P)\times\mathbb{V}(\mathsf{N}\otimes X\otimes P;A) &&=\\
&\textstyle\int^{M\in\mathsf{Nor}(\mathbb{V}),P\in\mathbb{V}}\mathsf{Nor}(\mathbb{V})(Y\otimes_{\mathsf{N}} Z;M)\times\mathsf{Nor}(\mathbb{V})(M;P)\times\mathbb{V}(\mathsf{N}\otimes X\otimes P;A) &&\cong\\
&\textstyle\int^{P\in\mathbb{V}}\mathsf{Nor}(\mathbb{V})(Y\otimes_{\mathsf{N}} Z;P)\times\mathbb{V}(\mathsf{N}\otimes X\otimes P;A)\times &&=\\
&\textstyle\int^{P\in\mathbb{V}}\mathbb{V}(\mathsf{N}\otimes Y\otimes\mathsf{N}\otimes Z\otimes\mathsf{N};P)\times\mathbb{V}(\mathsf{N}\otimes X\otimes P;A) &&\cong\\
&\mathbb{V}(\mathsf{N}\otimes X\otimes\mathsf{N}\otimes Y\otimes\mathsf{N}\otimes Z\otimes\mathsf{N};A).
\end{aligned}$$

Finally, let us construct the four interchangers that define the produoidal category. Three of them are immediate: they are either identities or unitors: $\mathsf{Nor}(\mathbb{V})(I_{\mathsf{N}},A)\cong\mathsf{Nor}(\mathbb{V})(I_{\mathsf{N}}\triangleleft I_{\mathsf{N}};A)$ is the first, $\mathsf{Nor}(\mathbb{V})(\mathsf{N}\otimes\mathsf{N};A)\cong\mathsf{Nor}(\mathbb{V})(\mathsf{N};A)$ is the second, and $\mathsf{Nor}(\mathbb{V})(I_{\mathsf{N}};A)\cong\mathsf{Nor}(\mathbb{V})(\mathsf{N}_{\mathsf{N}};A)$ is the last one. We note here that this produoidal category is natural because of these isomorphisms. Thus, we only need to construct the first interchanger morphism of a produoidal category,

$$\mathsf{Nor}(\mathbb{V})((X_1\triangleleft Y_1)\otimes(X_2\triangleleft Y_2);A)\longrightarrow\mathsf{Nor}(\mathbb{V})((X_1\otimes X_2)\triangleleft(Y_1\otimes Y_2);A),$$

which is defined by the following reasoning. Here, we abbreviate $\mathsf{N}\otimes X\otimes\mathsf{N}$ by $X^{\otimes\mathsf{N}}$, and we apply *(i)* the definition of the tensors of $\mathsf{Nor}(\mathbb{V})$, *(ii)* the interchanger of $\mathbb{V}$, *(iii)* unitality in $\mathbb{V}$, and *(iv)* the definition of tensors of $\mathsf{Nor}(\mathbb{V})$.

$$\begin{aligned}
&\mathsf{Nor}(\mathbb{V})((X_1\triangleleft_{\mathsf{N}} Y_1)\otimes_{\mathsf{N}}(X_2\triangleleft_{\mathsf{N}} Y_2);A) &&=\\
&\mathbb{V}(\mathsf{N}\otimes(X_1^{\otimes\mathsf{N}}\triangleleft Y_1^{\otimes\mathsf{N}})\otimes\mathsf{N}\otimes(X_2^{\otimes\mathsf{N}}\triangleleft Y_2^{\otimes\mathsf{N}})\otimes\mathsf{N};A) &&\to\\
&\mathbb{V}((\mathsf{N}\otimes X_1^{\otimes\mathsf{N}}\otimes\mathsf{N}\otimes Y_1^{\otimes\mathsf{N}}\otimes\mathsf{N})\triangleleft(\mathsf{N}\otimes X_2^{\otimes\mathsf{N}}\otimes\mathsf{N}\otimes Y_2^{\otimes\mathsf{N}}\otimes\mathsf{N});A) &&\to\\
&\mathbb{V}((\mathsf{N}\otimes X_1\otimes\mathsf{N}\otimes Y_1\otimes\mathsf{N})^{\otimes\mathsf{N}}\triangleleft(\mathsf{N}\otimes X_2\otimes\mathsf{N}\otimes Y_2\otimes\mathsf{N})^{\otimes\mathsf{N}};A) &&\to\\
&\mathsf{Nor}(\mathbb{V})((X_1\otimes_{\mathsf{N}} Y_1)\triangleleft_{\mathsf{N}}(X_2\otimes_{\mathsf{N}} Y_2);A).
\end{aligned}$$

The structure equations of the laxators follow from those of the base category $\mathbb{V}$. This finishes the construction of a produoidal category on the Kleisli category of the promonad. □

Lemma 4.4. *Normalization extends to an idempotent monad.*

PROOF. The first part of the proof will show that $\mathsf{Nor}(\mathbb{V})$ is the free normal produoidal category over $\mathbb{V}$ by constructing the a monad structure on top of the functor $\mathsf{Nor}\colon\mathbf{nProDuo}\to\mathbf{nProDuo}$. Let us construct the unit and the multiplication of the monad. The unit $\eta_{\mathbb{V}}:\mathbb{V}\to\mathsf{Nor}(\mathbb{V})$ is defined as the identity-on-objects functor associated to the promonad; it and acts on morphisms by the unit of the promonad. Let us show that this is a produoidal functor by constructing the following components; all of them use the map $I \to \mathsf{N}$ from the base produoidal category,

(1) $\eta_{\otimes} \colon \mathbb{V}(X \otimes Y; A) \to \mathbb{V}(I \otimes X \otimes I \otimes Y \otimes I; A) \to \mathbb{V}(\mathsf{N} \otimes X \otimes \mathsf{N} \otimes Y \otimes \mathsf{N}; A)$;
(2) $\eta_I : \mathbb{V}(I; A) \to \mathbb{V}(\mathsf{N}; A)$;
(3) $\eta_{\triangleleft} : \mathbb{V}(X \triangleleft Y; A) \to \mathbb{V}((I \otimes X \otimes I) \triangleleft (I \otimes Y \otimes I); A) \to \mathbb{V}((\mathsf{N} \otimes X \otimes \mathsf{N}) \triangleleft (\mathsf{N} \otimes Y \otimes \mathsf{N}); A)$;
(4) $\eta_{\mathsf{N}} : \mathbb{V}(\mathsf{N}; A) \to \mathbb{V}(\mathsf{N}; A)$ is simply an identity.

these preserve laxators and coherence maps since they are constructed only from laxators and coherence maps.

Let us construct now the multiplication of the monad, $\mu_{\mathbb{V}} : \mathsf{Nor}(\mathsf{Nor}(\mathbb{V})) \to \mathsf{Nor}(\mathbb{V})$, and show that it is an isomorphism, making it an idempotent monad. The underlying functor is identity on objects, and it acts on morphisms by the normality of the already normalized produoidal category,

$$\mathsf{Nor}(\mathsf{Nor}(\mathbb{V}))(X; Y) = \mathsf{Nor}(\mathbb{V})(\mathsf{N} \otimes_{\mathsf{N}} X \otimes_{\mathsf{N}} \mathsf{N}; Y) \cong \mathsf{Nor}(\mathbb{V})(X; Y).$$

The following components make this functor a produoidal functor, they are constructed again from the normality of the already normalized produoidal category:

(1) $\mu_{\otimes} : \mathsf{Nor}(\mathsf{Nor}(\mathbb{V}))(X \otimes_{\mathsf{NN}} Y; A) = \mathsf{Nor}(\mathbb{V})(\mathsf{N} \otimes_{\mathsf{N}} X \otimes_{\mathsf{N}} \mathsf{N} \otimes_{\mathsf{N}} Y \otimes_{\mathsf{N}} \mathsf{N}; A) \cong \mathsf{Nor}(\mathbb{V})(X \otimes_N Y; A)$;
(2) $\mu_{\triangleleft} : \mathsf{Nor}(\mathsf{Nor}(\mathbb{V}))(X \triangleleft_{\mathsf{NN}} Y; A) = \mathsf{Nor}(\mathbb{V})((\mathsf{N} \otimes_{\mathsf{N}} X \otimes_{\mathsf{N}} \mathsf{N}) \triangleleft_{\mathsf{N}} (\mathsf{N} \otimes_{\mathsf{N}} Y \otimes_{\mathsf{N}} \mathsf{N}); A) \cong \mathsf{Nor}(\mathbb{V})(X \triangleleft_{\mathsf{N}} Y; A)$;
(3) $\mu_I : \mathsf{Nor}(\mathsf{Nor}(\mathbb{V}))(\mathsf{N}; A) = \mathsf{Nor}(\mathbb{V})(\mathsf{N}; A)$;
(4) $\mu_{\mathsf{N}} : \mathsf{Nor}(\mathsf{Nor}(\mathbb{V}))(\mathsf{N}; A) = \mathsf{Nor}(\mathbb{V})(\mathsf{N}; A)$,

Finally we verify the monad laws. $\eta_{\mathcal{N}\mathbb{V}} ⨾ \mu_{\mathbb{V}}$ is an identity-on-objects; on morphisms, it applies left and right unitors followed by their inverses; as a consequence, its underlying functor is the identity. The components of the natural transformations are also identities, since the interchanger $I \to \mathsf{N}$ is an identity for the normalized produoidal category $\mathsf{Nor}(\mathbb{V})$; they are otherwise composed of unitors followed by their inverses. The last step is to check the associativity of the monad, $\mu_{\mathsf{Nor}(\mathbb{V})} ⨾ \mu_{\mathbb{V}}$ and $\mathsf{Nor}(\mu_{\mathbb{V}}) ⨾ \mu_{\mathbb{V}}$; this is simply the identity on objects, so we simply apply left and right unitors twice on morphisms and their components. □

Lemma 4.5. *A produoidal category $\mathbb{V}$ has exactly one algebra structure for the normalization monad when it is normal, and none otherwise.*

Proof. Let $(F, F_{\otimes}, F_I, F_{\mathsf{N}}, F_{\mathsf{N}}) \colon \mathsf{Nor}(\mathbb{V}) \to \mathbb{V}$ be an algebra. This means that the following commutative diagrams with the unit and multiplication of the normalization monad must commute.

$$\begin{tikzcd}
\mathbb{V} \arrow[r, "\eta"] \arrow[dr, no head, "\mathrm{id}"'] & \mathsf{Nor}(\mathbb{V}) \arrow[d, "F"] \\
& \mathbb{V}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Nor}(\mathsf{Nor}(\mathbb{V})) \arrow[r, "\mu"] \arrow[d, "\mathsf{Nor}(F)"'] & \mathsf{Nor}(\mathbb{V}) \arrow[d, "F"] \\
\mathsf{Nor}(\mathbb{V}) \arrow[r, "F"'] & \mathbb{V}
\end{tikzcd}$$

Now, consider how the interchanger $\psi_0 \colon \mathbb{V}(I; \bullet) \to \mathbb{V}(\mathsf{N}; \bullet)$ is transported by these maps.

$$\begin{tikzcd}
& & \mathbb{V}(\mathsf{N};\bullet) \arrow[d, "\mathrm{id}"] \arrow[drr, "F_I"] & & \\
\mathbb{V}(I;\bullet) \arrow[urr, "\eta_I"] \arrow[d, "\psi_0"'] \arrow[rrrr, bend right, "\mathrm{id}"] & & \mathbb{V}(\mathsf{N};\bullet) \arrow[drr, "F_N"] & & \mathbb{V}(I;\bullet) \arrow[d, "\psi_0"] \\
\mathbb{V}(\mathsf{N};\bullet) \arrow[urr, "\mathrm{id}"] \arrow[rrrr, bend right, "\mathrm{id}"'] & & & & \mathbb{V}(\mathsf{N};\bullet)
\end{tikzcd}$$

We conclude that $\eta_I = \psi_0$, but also that $F_{\mathsf{N}} = \mathrm{id}$. As a consequence, ψ_0 is invertible and F_I must be its inverse. We have shown that any produoidal category that is an algebra for the normalization monad must be normal.

We will now show that this already determines all of the functor F. We know that $\eta_\otimes, \eta_\triangleleft, \eta$ are isomorphisms because they are constructed from the unitors, associators, and the laxator ψ_0, which is an isomorphism in this case. This determines that $F_\otimes, F_\triangleleft, F$ must be their inverses. By construction, these satisfy all structure equations. □

THEOREM 4.6 (Free normal produoidal). *Normalization determines an adjunction between produoidal categories and normal produoidal categories,*

$$\mathsf{Nor} \colon \mathbf{ProDuo} \rightleftharpoons \mathbf{nProDuo} \colon \mathsf{Forget}.$$

That is, $\mathsf{Nor}(\mathbb{V})$ *is the free normal produoidal category over* $\mathbb{V}$*.*

PROOF. We know that the algebras for the normalization monad are exactly the normal produoidal categories (Lemma 4.5). We also know that the normalization monad is idempotent (Theorem 4.6). This implies that the forgetful functor from its category of algebras is fully faithful, and thus, the algebra morphisms are exactly the produoidal functors. As a consequence, the canonical adjunction to the category of algebras of the monad is exactly an adjunction to the category of normal produoidal categories. □

Remark 4.7. Garner and López Franco [GF16] introduced a partial normalization procedure for duoidal categories. We contribute a general normalization procedure for produoidal categories and we characterize it universally. Produoidal normalization behaves slightly better than duoidal normalization: it always succeeds, and we prove that it forms an idempotent monad (Theorem 4.6). The technical reason for this improvement is that the original duoidal normalization required the existence of certain coequalizers in $\mathbb{V}$; produoidal normalization uses coequalizers in **Set**.

In the previous Section 3.3, we constructed the produoidal category of spliced monoidal arrows, which distinguishes between morphisms and morphisms with a hole in the monoidal unit. This is because the latter hole splits the morphism in two parts. Normalization equates both; it sews these two parts. In Section 5, we explicitly construct monoidal contexts, the normalization of spliced monoidal arrows. Before that, let us also introduce the symmetric version of normalization.

4.3. Symmetric Normalization. Normalization is a generic procedure that applies to any produoidal category, it does not matter if the parallel join ($\otimes$) is symmetric or not. However, when $\otimes$ happens to be symmetric, we can also apply a more specialized normalization procedure: *symmetric normalization.*

Definition 4.8 (Symmetric produoidal category)**.** A *symmetric produoidal category* is a produoidal category $\mathbb{V}_{\triangleleft,N,\otimes,I}$ endowed with a natural isomorphism $\sigma\colon \mathbb{V}(X \otimes Y; Z) \cong \mathbb{V}(Y \otimes X; Z)$ satisfying the symmetry and hexagon equations.

Theorem 4.9. *Let $\mathbb{V}$ be a symmetric produoidal category. The profunctor*

$$\mathsf{sNor}(\mathbb{V})(\bullet;\bullet) = \mathbb{V}(N \otimes \bullet;\bullet)$$

forms a promonad. Moreover, the Kleisli category of this promonad is a normal symmetric produoidal category with the following profunctors.

1. $\mathsf{sNor}(\mathbb{V})(X \otimes_N Y; Z) = \mathbb{V}(N \otimes X \otimes Y; Z)$*;*
2. $\mathsf{sNor}(\mathbb{V})(X \triangleleft_{\mathsf{N}} Y; Z) = \mathbb{V}((N \otimes X) \triangleleft (N \otimes Y); Z)$*; and*
3. $\mathsf{sNor}(\mathbb{V})(I_{\mathsf{N}}; X) = \mathsf{sNor}(\mathbb{V})(\mathsf{N}_{\mathsf{N}}; X) = \mathbb{V}(\mathsf{N}; X)$*.*

Theorem 4.10. *Normalization determines an adjunction between symmetric produoidal and normal symmetric produoidal categories,*

$$\mathsf{sNor}\colon \mathbf{symProDuo} \rightleftharpoons \mathbf{nSymProDuo}\colon \mathcal{U}.$$

That is, $\mathsf{sNor}(\mathbb{V})$ is the free normal symmetric produoidal category over $\mathbb{V}$.

4.4. Bibliography. Garner and López-Franco contributed a partial normalization procedure for duoidal categories [GF16], all of the credit for this elegant idea goes there.

We contribute its produoidal counterpart. The reader could think that this is an automatic process: extending an argument by Day [Day70], produoidal categories could be understood as closed duoidal categories in some sense. However, we show that there are some technical differences that make this case important: in the work of Garner and López-Franco, not every duoidal category has a normalization, and this prevents us from constructing a monad. We prove that produoidal normalization is always possible and defines an idempotent monad.

5. Monoidal Lenses

Monoidal lenses – the name we give to monoidal contexts [PGW17, Ril18, CEG+20] – formalize the notion of an incomplete morphism in a monoidal category. The category of monoidal lenses will have a rich algebraic structure: we shall be able to still compose contexts sequentially and in parallel and, at the same time, we shall be able to fill a context using another monoidal context. Perhaps surprisingly, then, the category of monoidal lenses is not even monoidal.

We justify this apparent contradiction in terms of profunctorial structure: the category is not monoidal, but it does have two promonoidal structures that precisely represent sequential and parallel composition. These structures form a normal produoidal category. In fact, we show it to be the normalization of the produoidal category of spliced monoidal arrows. This section constructs explicitly this normal produoidal category of monoidal lenses.

5.1. The Category of Monoidal Lenses. A monoidal lens – an element of type $\binom{X}{Y} \to \binom{A}{B}$ – represents a process from A to B with a hole admitting a process from X to Y. In this sense, monoidal lenses are similar to spliced monoidal arrows. The difference with spliced monoidal arrows is that monoidal lenses allow for communication to happen to the left and to the right of this hole.

Definition 5.1 (Monoidal lens)**.** Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. *Monoidal lenses* are the elements of the profunctor

$$\mathsf{mLens}\,(\tbinom{X}{Y}; \tbinom{A}{B}) = \int^{M_1,M_2} \mathbb{C}(A; M_1 \otimes X \otimes M_2) \times \mathbb{C}(M_1 \otimes Y \otimes M_2; B).$$

In other words, a *monoidal lens* from A to B, *with a hole* from X to Y, is an equivalence class consisting of a pair of objects $M, N \in \mathbb{C}_{\mathrm{obj}}$ and a pair of morphisms $f \in \mathbb{C}(A; M \otimes X \otimes N)$ and $g \in \mathbb{C}(M \otimes Y \otimes N; B)$, quotiented by dinaturality of M and N.

Definition 5.2. Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. Its normal produoidal category of *monoidal lenses*, $\mathsf{mLens}(\mathbb{C})$, has objects formed by pairs, $\mathsf{mLens}(\mathbb{C})_{obj} = (\mathbb{C}^{op} \times \mathbb{C})_{obj}$, and is defined by the following profunctors.

(1) Morphisms are diagrams with a single typed hole.

$$\mathsf{mLens}\mathbb{C}\,(\tbinom{X}{Y}; \tbinom{A}{B}) = \int^{M_1,M_2} \mathbb{C}(A; M_1 \otimes X \otimes M_2) \times \mathbb{C}(M_1 \otimes Y \otimes M_2; B),$$

(2) Sequential joins are diagrams with a pair of sequential holes.

$$\mathsf{mLens}\mathbb{C}(\tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}; \tbinom{A}{B};) = \int^{M_1,M_2} \mathbb{C}(A; M_1 \otimes X \otimes M_2) \times \mathbb{C}(M_1 \otimes Y \otimes M_2; \\ M_3 \otimes X' \otimes M_4) \times \mathbb{C}(M_3 \otimes Y' \otimes M_4; B);$$

(3) Parallel joins are diagrams with a pair of parallel holes.

$$\mathsf{mLens}(\mathbb{C})(\tbinom{X}{Y} \otimes \tbinom{X'}{Y'}; \tbinom{A}{B}) = \int^{M_1,M_2,M_3} \begin{array}{l}\mathbb{C}(A; M_1 \otimes X \otimes M_2 \otimes X' \otimes M_3)\times \\ \mathbb{C}(M_1 \otimes Y \otimes M_2 \otimes Y' \otimes M_3; B)\end{array}$$

(4) Units are complete diagrams with no holes, $\mathsf{mLens}(\mathbb{C})(\mathsf{N}; \tbinom{X}{Y}) = \mathbb{C}(X;Y)$.

Reading the profunctorial notation can be unenlightening. We provide the incomplete string diagrams for these profunctors in Figure 12 [Rom20b].

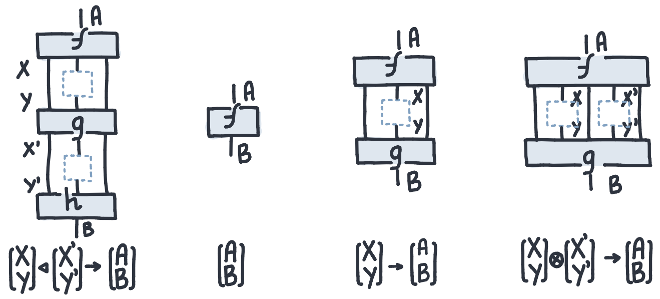

A
f
X
Y
g
X'
Y'
h
B
$[X/Y] \triangleleft [X'/Y'] \to [A/B]$
A
f
B
$[A/B]$
A
f
X
Y
g
B
$[X/Y] \to [A/B]$
A
f
X
Y
X'
Y'
g
B
$[X/Y] \otimes [X'/Y'] \to [A/B]$

FIGURE 12. Monoidal lenses.

THEOREM 5.3. *The category of monoidal lenses forms a normal produoidal category with its units, sequential and parallel joins. Monoidal lenses are the free normalization of the cofree produoidal category over a category. In other words, monoidal lenses are the normalization of spliced monoidal arrows,*

$$\mathsf{mLens}(\mathbb{C}) \cong \mathsf{Nor}(\mathsf{mSplice}(\mathbb{C})).$$

PROOF. The core of this result is in Theorem 4.6, which says that the normalization procedure yields the free normalization over a produoidal category. It is only left to check that this produoidal category of monoidal lenses that we have explicitly constructed in this section is precisely the normalization of the produoidal category of spliced arrows. We do so for morphisms, the rest of the proof is similar; the proof shows that $\mathsf{mSplice}(\mathbb{C})\ (\mathsf{N} \otimes \tbinom{X}{Y} \otimes \mathsf{N}; \tbinom{A}{B})$, the normalization of spliced monoidal arrows, is isomorphic to monoidal lenses, $\mathsf{mLens}(\mathbb{C})\ (\tbinom{A}{B}; \tbinom{X}{Y})$. We employ the Yoneda lemma on both V and V'.

$$\int^{U,V,U',V' \in \mathbb{C}} \mathbb{C}(A; U \otimes X \otimes U') \times \mathbb{C}(V \otimes Y \otimes V'; B) \times \mathbb{C}(U;V) \times \mathbb{C}(U';V') \quad \cong$$

$$\int^{U,U' \in \mathbb{C}} \mathbb{C}(A; U \otimes X \otimes U') \times \mathbb{C}(U \otimes Y \otimes U'; B)$$

The rest of the profunctors follow a similar reasoning. □

5.2. Symmetric Monoidal Lenses. A symmetric monoidal lens of type $\mathsf{smLens}(\mathbb{C})(\begin{smallmatrix}X\\Y\end{smallmatrix}; \begin{smallmatrix}A\\B\end{smallmatrix})$ represents a process in a symmetric monoidal category with a hole admitting a process from X to Y.

Symmetric monoidal lenses are monoidal lenses, but we stop caring where the hole is. Again, the category of symmetric monoidal lenses has a rich algebraic structure; and again, most of this structure exists only virtually in terms of profunctors. In this case, though, the monoidal tensor *does* indeed exist: contrary to monoidal lenses, symmetric monoidal lenses form also a monoidal category. This is perhaps why applications of monoidal lenses have grown popular in recent years [Ril18], with applications in decision theory [GHWZ18], supervised learning [CGG⁺22, FJ19] and most notably in functional data accessing [Kme12, PGW17, BG18, CEG⁺20]. The promonoidal structure of optics was ignored, even when, after now identifying for the first time its relation to the monoidal structure of optics, we argue that it could be potentially useful in these applications: e.g. in multi-stage decision problems, or in multi-stage data accessors.

This section explicitly constructs the normal symmetric produoidal category of *symmetric monoidal lenses.* We describe it for the first time by a universal property: it is the free symmetric normalization of the cofree produoidal category.

Definition 5.4. Let $(\mathbb{C}, \otimes, I)$ be a symmetric monoidal category. *Symmetric monoidal lenses* are the elements of the profunctor

$$\mathsf{smLens}\mathbb{C}\left(\begin{smallmatrix}X\\Y\end{smallmatrix}; \begin{smallmatrix}A\\B\end{smallmatrix}\right) = \int^{M} \mathbb{C}(A; M \otimes X) \times \mathbb{C}(M \otimes Y; B).$$

In other words, a *symmetric monoidal lens* from A to B, *with a hole* from X to Y, is an equivalence class consisting of a pair of objects $M \in \mathbb{C}_{obj}$ and a pair of morphisms $f \in \mathbb{C}(A; M \otimes X)$ and $g \in \mathbb{C}(M \otimes Y; B)$, quotiented by dinaturality of M.

Definition 5.5. Let $(\mathbb{C}, \otimes, I)$ be a symmetric monoidal category. Its normal symmetric produoidal category of *symmetric monoidal lenses*, $\mathsf{smLens}(\mathbb{C})$, has objects formed by pairs, $\mathsf{smLens}(\mathbb{C})_{obj} = (\mathbb{C}^{op} \times \mathbb{C})_{obj}$, and is defined by the following profunctors.

(1) Morphisms are diagrams with a single typed hole.

$$\mathsf{smLens}(\mathbb{C})\left(\begin{smallmatrix}X\\Y\end{smallmatrix}; \begin{smallmatrix}A\\B\end{smallmatrix}\right) = \int^{M} \mathbb{C}(A; M \otimes X) \times \mathbb{C}(M \otimes Y; B),$$

(2) Sequential joins are diagrams with a pair of sequential holes.

$$\mathsf{smLens}(\mathbb{C})(\begin{smallmatrix}X\\Y\end{smallmatrix} \triangleleft \begin{smallmatrix}X'\\Y'\end{smallmatrix}; \begin{smallmatrix}A\\B\end{smallmatrix}) = \int^{M_1, M_2} \mathbb{C}(A; M_1 \otimes X) \times \mathbb{C}(M_1 \otimes Y; M_2 \otimes X') \times \mathbb{C}(M_2 \otimes Y'; B);$$

(3) Parallel joins are diagrams with a hole encompassing parallel wires.

$$\mathsf{smLens}(\mathbb{C})(\tbinom{X}{Y} \otimes \tbinom{X'}{Y'}; \tbinom{A}{B}) = \int^{M} \mathbb{C}(A; M \otimes X \otimes X') \times \mathbb{C}(M \otimes Y \otimes Y'; B).$$

(4) Units are complete diagrams with no holes, $\mathsf{smLens}(\mathbb{C})(\mathsf{N}; \tbinom{A}{B}) = \mathbb{C}(A; B)$.

Reading the profunctorial notation can be unenlightening. We provide the incomplete string diagrams for these profunctors in Figure 12 [Rom20b]. The only substantial difference with monoidal lenses is that we do not need to keep track of where the hole is placed.

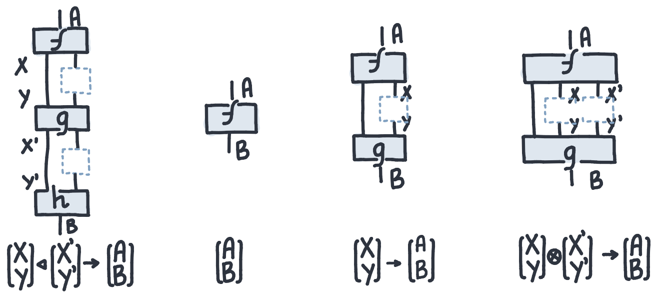

A
f
X
Y
g
X'
Y'
h
B
$\binom{X}{Y} \triangleleft \binom{X'}{Y'} \to \binom{A}{B}$
A
f
B
$\binom{A}{B}$
A
f
X
Y
g
B
$\binom{X}{Y} \to \binom{A}{B}$
A
f
X
X'
Y
Y'
g
B
$\binom{X}{Y} \otimes \binom{X'}{Y'} \to \binom{A}{B}$

FIGURE 13. Symmetric monoidal lenses.

The term "monoidal lenses" has usually been reserved for the morphisms of this category; in the literature, the sequential splits – the lenses with multiple holes – get a name from their distinctive shape: these are *combs* or *quantum combs* [CFS16].

5.3. Towards Message Theories. Lenses, or *combs*, can be interpreted as incomplete morphisms, but also as morphisms that send and receive resources. The next chapter will exploit this intuition.

Remark 5.6 (Session notation for combs)**.** We will write $A^{\circ} = \tbinom{A}{I}$ and $B^{\bullet} = \tbinom{I}{B}$ for the objects of the symmetric produoidal category of lenses that have a monoidal unit as one of its objects. Thanks to $A^{\circ} \otimes B^{\bullet} = \tbinom{A}{B}$, these are enough to express all objects.

Proposition 5.7. *Let* $(\mathbb{C}, \otimes, I)$ *be a symmetric monoidal category. There exist monoidal functors*

$$(^{\circ}) \colon \mathbb{C} \to \mathsf{smLens}(\mathbb{C}), \qquad \textit{and} \qquad (^{\bullet}) \colon \mathbb{C}^{op} \to \mathsf{smLens}(\mathbb{C}).$$

Moreover, they satisfy the following properties definitionally: $\mathbb{C}(A^{\bullet} \triangleleft B^{\bullet}; \bullet) \cong \mathbb{C}(A^{\bullet} \otimes B^{\bullet}; \bullet)$*;* $(A \otimes B)^{\circ} = A^{\circ} \otimes B^{\circ}$*;* $\mathbb{C}(A^{\circ} \triangleleft B^{\circ}; \bullet) \cong \mathbb{C}(A^{\circ} \otimes B^{\circ}; \bullet)$*;* $(A \otimes B)^{\bullet} = A^{\bullet} \otimes B^{\bullet}$*; and* $\mathbb{C}(A^{\circ} \triangleleft B^{\bullet}; \bullet) \cong \mathbb{C}(A^{\circ} \otimes B^{\bullet}; \bullet)$*.*

PROOF. We define $f^\circ = (f \fatsemi \square \fatsemi \mathrm{id}_I)$ and $g^\bullet = (\mathrm{id}_I \fatsemi \square \fatsemi g)$, and then check that compositions and tensoring of morphisms are compatible with composition and tensoring of monoidal lenses, this is straightforward. Moreover, we can see that, by definition,

$$(A \otimes B)^\circ = \binom{A\otimes B}{I} = \binom{A}{I} \otimes \binom{B}{I} = A^\circ \otimes B^\circ, \quad \text{and}$$
$$(A \otimes B)^\bullet = \binom{I}{A\otimes B} = \binom{I}{A} \otimes \binom{I}{B} = A^\bullet \otimes B^\bullet.$$

This proof appears with a different language in the work of Riley [Ril18, Proposition 2.0.14]. In fact, there, the combined identity-on-objects functor $(\circ \times \bullet) \colon \mathbb{C} \times \mathbb{C}^{op} \to \mathsf{smLens}(\mathbb{C})$ is shown to be monoidal. $\square$

Example 5.8. Let us give a first example of how to employ combs for the description of concurrent protocols. Broadbent and Karvonen [BK22] propose a formalization of the *one-time pad* encryption protocol in a symmetric monoidal category endowed with a Hopf algebra with an integral.

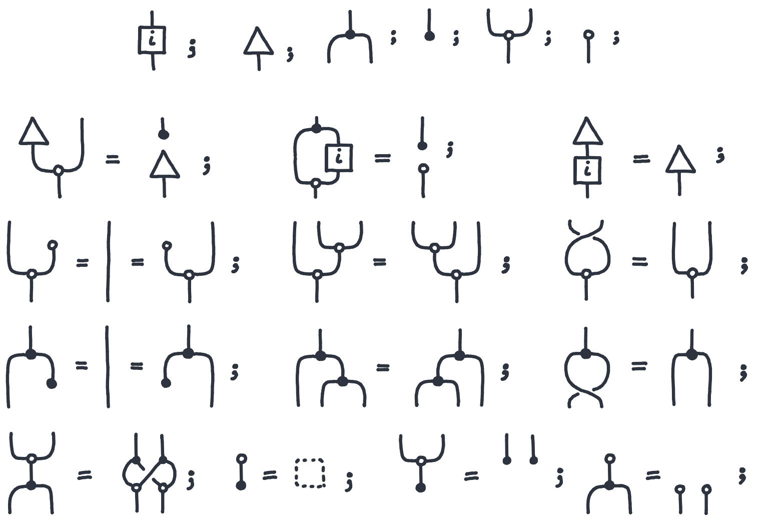


FIGURE 14. Theory of a Hopf algebra with an integral.

Definition 5.9. A Hopf algebra with an integral, $(X, \text{⅄}, \text{⫯}, \text{Y}, \text{⫯}, i, d)$, is a commutative bialgebra endowed with an *antipode* map $i \colon X \to X$, representing inversion; and endowed with an *integral map*, $d \colon I \to X$, representing a non-determined value, or pure noise. These must satisfy the equations in Figure 14.

The one-time pad is a mathematically secure encryption technique. It works as follows: *(i)* the two parties communicating – say, Alice and Bob – start by preparing some random bits and sharing them; *(ii)* when the message is ready, Alice applies bitwise XOR with the random bits to encrypt the message, and then broadcasts the encrypted message – a potential attacker, Eve, will receive this encrypted message; *(iii)* finally, Bob receives the encrypted message and applies again bitwise XOR with the random bits to decrypt the message (Figure 15).

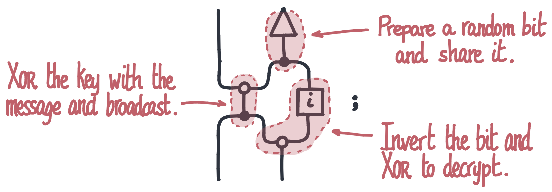

Prepare a random bit and share it.
Xor the key with the message and broadcast.
Invert the bit and Xor to decrypt.
i
;

FIGURE 15. Description of the one-time pad.

Proposition 5.10. *The one-time pad is* secure*, meaning that it is equal to the process that sends a message from* Alice *to* Bob *and outputs random noise through the attacker's channel.*

PROOF. We repeat the proof from Broadbent and Karvonen [BK22]. We employ string diagrams of symmetric monoidal categories, in Figure 16, to show that the morphism is equal to an identity tensored by the integral of the Hopf algebra. □

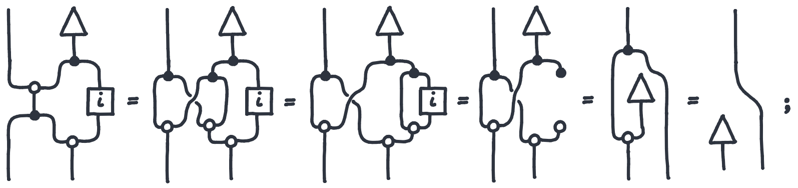


FIGURE 16. Correctness of the one-time pad.

The interesting part comes when we want to split the morphism into its different constituents: there should be a *stage* where the three actors play; Alice does not control the fact that the encrypted message will be broadcast; Eve, the attacker, can only attack at the end; Bob will need to keep a bit in memory. These considerations are part of the problem statement the one-time pad is solving. It is easy to come up with a morphism that connects an input to an output: the problem the one-time pad is solving is to do so on a stage that has been preset.

The components of the one time pad are not simply morphisms of a monoidal category (Figure 17). They must be understood as monoidal lenses. After this section, we can declare that a possible typing for these components is the following:

(1) Alice: $\mathsf{N} \to \binom{X \otimes X}{X}$;
(2) Bob: $X^{\bullet} \triangleleft X^{\circ} \to \binom{I}{X}$;
(3) Eve: $\mathsf{N} \to \binom{X}{X}$;
(4) Stage: $X^{\circ} \triangleleft X^{\bullet} \triangleleft X^{\bullet} \triangleleft X^{\circ} \triangleleft X^{\bullet} \triangleleft X^{\bullet} \triangleleft X^{\circ} \triangleleft X^{\circ} \to \binom{X}{X \otimes X}$;

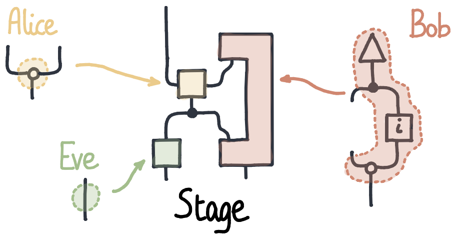


FIGURE 17. Components of the one-time pad.

Still, at this stage it is not easy to talk about message passing with this syntax. It is true that the new produoidal types can track all the exchanges that happen along a boundary; but these types are tedious – see, for instance, the long type of Stage – and it is not clear how to compose them. What we are missing is a combinatorial description of the different ways we can combine elements of this produoidal algebra.

This is what the next chapter will solve: we will propose a combinatorial algebra of message passing and show that lenses, the normalized cofree produoidal algebra over a symmetric monoidal category have a second universal property – they also constitute the free message theory.

5.4. Bibliography. Lenses [FGM+07] are a notion of bidirectional transformation that can be cast in arbitrary monoidal categories. The first mention of monoidal lenses separate from their classical database counterparts [JRW12] is due to Pastro and Street [PS07], who identify them as an example of a promonoidal category. However, it was with a different monoidal structure [Ril18] that they became popular in recent years, spawning applications not only in bidirectional transformations [FGM+07] but also in functional programming [PGW17, CEG^{+}20], open games [GHWZ18], polynomial functors [NS22] and quantum combs [HC22]. Relating this monoidal category of lenses with the previous promonoidal category of lenses was an open problem; and the promonoidal structure was mostly ignored in applications. We solve this problem, proving that lenses are a universal normal symmetric produoidal category (the symmetric monoidal lenses), which endows them with a novel algebra and a novel universal property. This also extends work on the relation between *incomplete diagrams*, *comb-shaped diagrams*, and *lenses* [Rom20a, Rom20b].

Lenses themselves have been applied to protocol specification [VC22]. Spivak [Spi13] also discusses the multicategory of *wiring diagrams*, later used for incomplete diagrams [PSV21] and related to lenses [SSV20]; we conjecture that this multicategory of wiring diagrams is precisely the produoidal category of lenses, once we stop tracking dependencies explicitly.

Conjecture 5.11. *Each physical produoidal category induces a multicategory given by its physical lax tensor (Section 2.3). The multicategory of wiring diagrams [Spi13] of symmetric monoidal categories is the multicategory induced by the produoidal category of monoidal lenses.*

The promonoidal categories we use can be seen as multicategories with an extra coherence property. In this sense, we contribute the missing algebraic structure of the universal multicategory of *wiring diagrams relative to a monoidal category.*

CHAPTER 4

Monoidal Message Passing

Monoidal Message Passing

This chapter develops message passing in monoidal categories following the theory of context we just constructed. We have already defined what incomplete morphisms in monoidal categories are: we will now study the structure of all their possible compositions. This includes not only the obvious operations of composition but any possible *wiring* of a diagram that could combine them while respecting the acyclicity of string diagrams.

Studying seriously the combinatorial structure of string diagram composition arrives at the same conclusion as axiomatizing a naive theory of message passing: message theories. Indeed, we prove that the polarized shufflings that describe string diagram composition have as algebras precisely the message theories.

Section 1 introduces our minimalistic theory of message passing, with axioms that should hopefully be acceptable to any reader. Section 2 starts developing the categorical semantics for message theories, based on physical monoidal multicategories (a variant of duoidal categories); it then shows that shufflings from the free physical monoidal multicategory. Section 3 provides the second ingredient for this categorical semantics: polarization. We then combine both ingredients in Section 4: polar shuffles form the combinatorial structure that combines incomplete string diagrams; message theories are precisely the algebras of physical monoidal multicategory of polar shuffles. This chapter ends with an adjunction between message theories and symmetric monoidal categories, which ensures that we can construct a free message theory on top of any symmetric monoidal category.

1. Message Theories

1.1. Message Theories. Message passing requires the interplay of at least two mathematical structures: the ability to *interleave* events in time and the ability to connect a *sender and a receiver*. Let us propose a minimally axiomatized algebra of interleaving and sending/receiving: interleaving will correspond to a normal duoidal algebra, and sending/receiving will correspond to polarization.

Definition 1.1. A *message theory* $\mathbb{M}$ consists of a set of types, $\mathbb{M}_{obj}$ with extra structure: a *send/receive session type* is a polarized list of types; for each session type, we have a collection of *sessions* with that type,

$\mathbb{M}(X_1^{\bullet_1}, \dots, X_n^{\bullet_n})$, for each $X_1, \dots, X_n \in \mathbb{M}_{obj}$, and each polarization $\bullet_i \in \{\circ, \bullet\}$.

A message theory must contain operations for *(i)* binary shuffling, *(ii)* and nullary shuffling, *(iii)* linking a sent message to immediately receive it, and *(iv)* spawning a channel that receives a message and sends it immediately.

(1) $\text{SHF}_\sigma \colon \mathbb{M}(\Gamma) \times \mathbb{M}(\Delta) \to \mathbb{M}(\sigma(\Gamma, \Delta))$, shuffling two processes;
(2) $\text{NOP} \colon \mathbb{M}()$, a no-operation, doing nothing;
(3) $\text{LNK}_x^{\Gamma;\Delta} \colon \mathbb{M}(\Gamma, X^\bullet, X^\circ, \Delta) \to \mathbb{M}(\Gamma, \Delta)$, linking send to receive;
(4) $\text{SPW}_x^{\Gamma;\Delta} \colon \mathbb{M}(\Gamma, \Delta) \to \mathbb{M}(\Gamma, X^\circ, X^\bullet, \Delta)$, a receive to send channel.

Message theories may be better understood in the notation of a logic, as in Figure 1. Types form a free polarized monoid; each term describes a possible communication protocol.

$$\frac{\Gamma \qquad \Delta}{[\Gamma, \Delta]_\sigma}\ (\text{SHF}_\sigma) \qquad \frac{\Gamma, X^\bullet, X^\circ, \Delta}{\Gamma, \Delta}\ (\text{LNK}) \qquad \frac{\Gamma, \Delta}{\Gamma, X^\circ, X^\bullet, \Delta}\ (\text{SPW}) \qquad \frac{-}{\varepsilon}\ (\text{NOP})$$

FIGURE 1. Type-theoretic presentation of a message theory.

A message theory must satisfy the following axioms: *(i)* shuffles compose as in their symmetric malleable multicategory, where we write $(\sigma \mathbin{⨾}_1 \tau)$ to be the same by associativity as $(\tau' \mathbin{⨾}_2 \sigma')$, we write $(*)$ for the trivial shuffle, and we write $\tilde{\sigma}$ for the symmetric counterpart of σ; *(ii)* linking is natural with respect to the shuffles; *(iii)* spawning is natural with respect to the shuffles; *(iv)* linking is dual to spawning; and *(v)* independent linkings and spawnings commute.

(1a) $\text{SHF}_\tau(\text{SHF}_\sigma(m_1 m_2), m_3) = \text{SHF}_{\sigma'}(m_1, \text{SHF}_{\tau'}(m_2, m_3))$;
(1b) $\text{SHF}_*(m, \text{NOP}) = m$;
(1c) $\text{SHF}_\sigma(m_1, m_2) = \text{SHF}_{\tilde{\sigma}}(m_2, m_1)$;
(2a) $\text{SHF}_{\sigma,\tau}(\text{LNK}_x^{\Gamma_1,\Gamma_2}(m_1), m_2) = \text{LNK}_x^{\Gamma_1,\Delta_1;\Gamma_2,\Delta_2}(\text{SHF}_{\sigma,x,\tau}(m_1, m_2))$;
(2b) $\text{SHF}_{\sigma,\tau}(m_1, \text{LNK}_x^{\Delta_1,\Delta_2}(m_2)) = \text{LNK}_x^{\Gamma_1,\Delta_1;\Gamma_2,\Delta_2}(\text{SHF}_{\sigma,x,\tau}(m_1, m_2))$;
(3a) $\text{SHF}_{\sigma,\tau}(\text{SPW}_x^{\Gamma_1,\Gamma_2}(m_1), m_2) = \text{SPW}_x^{\Gamma_1,\Delta_1;\Gamma_2,\Delta_2}(\text{SHF}_{\sigma,\tau}(m_1, m_2))$;

(3b) $\textsc{shf}_{\sigma,\tau}(m_1, \textsc{spw}_x^{\Delta_1,\Delta_2}(m_2)) = \textsc{spw}_x^{\Gamma_1,\Delta_1;\Gamma_2,\Delta_2}(\textsc{shf}_{\sigma,\tau}(m_1, m_2))$;
(4a) $\textsc{lnk}_x^{\Gamma,X^\circ;\Delta}(\textsc{spw}_x^{\Gamma;X^\circ,\Delta}(m)) = m$;
(4b) $\textsc{lnk}_x^{\Gamma;X^\bullet,\Delta}(\textsc{spw}_x^{\Gamma,X^\bullet;\Delta}(m)) = m$;
(5a) $\textsc{lnk}_x^{\Gamma_1;\Gamma_2Y\Gamma_3}(\textsc{spw}_y^{\Gamma_1X\Gamma_2;\Gamma_3}(m)) = \textsc{spw}_y^{\Gamma_1\Gamma_2;\Gamma_3}(\textsc{lnk}_x^{\Gamma_1;\Gamma_2\Gamma_3}(m))$;
(5b) $\textsc{lnk}_y^{\Gamma_1X\Gamma_2;\Gamma_3}(\textsc{spw}_x^{\Gamma_1;\Gamma_2Y\Gamma_3}(m)) = \textsc{spw}_x^{\Gamma_1;\Gamma_2\Gamma_3}(\textsc{lnk}_y^{\Gamma_1X\Gamma_2;\Gamma_3}(m))$;
(5c) $\textsc{spw}_x^{\Gamma_1;\Gamma_2Y\Gamma_3}(\textsc{spw}_y^{\Gamma_1\Gamma_2;\Gamma_3}(m)) = \textsc{spw}_y^{\Gamma_1X\Gamma_2;\Gamma_3}(\textsc{spw}_x^{\Gamma_1;\Gamma_2\Gamma_3}(m))$;
(5d) $\textsc{lnk}_x^{\Gamma_1;\Gamma_2\Gamma_3}(\textsc{lnk}_y^{\Gamma_1X\Gamma_2;\Gamma_3}(m)) = \textsc{lnk}_y^{\Gamma_1\Gamma_2;\Gamma_3}(\textsc{lnk}_x^{\Gamma_1;\Gamma_2Y\Gamma_3}(m))$.

These axioms are again better understood in logic notation, as equations between derivations, see Figure 2.

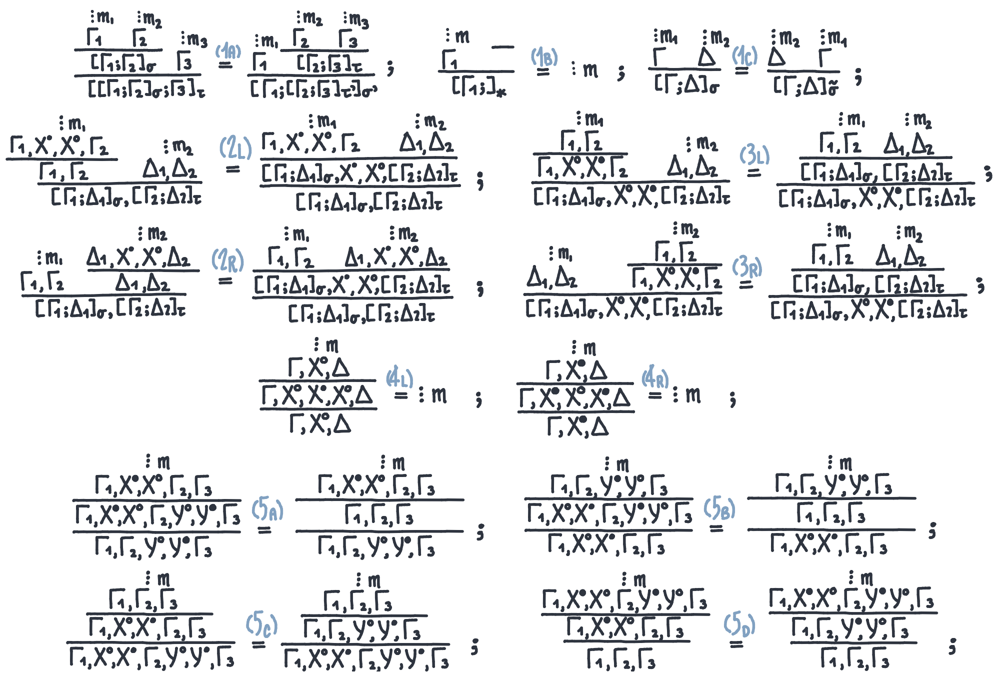


FIGURE 2. Axioms of a message theory.

Definition 1.2. A *message functor* $F\colon \mathbb{M} \to \mathbb{N}$ between two message theories, $\mathbb{M}$ and $\mathbb{N}$, is a function on objects $F_{obj}\colon \mathbb{M}_{obj} \to \mathbb{N}_{obj}$ that extends to a family of functions on session sets,

$$F\colon \mathbb{M}(X_1^\bullet, \dots, X_n^\bullet) \to \mathbb{N}(FX_1^\bullet, \dots, FX_n^\bullet).$$

This function must *(i)* preserve shuffling, $F(\textsc{shf}_\sigma(f,g)) = \textsc{shf}_\sigma(Ff, Fg)$; *(ii)* preserve spawning, $F(\textsc{spw}_x) = \textsc{spw}_{Fx}$; *(iii)* connecting, $F(\textsc{lnk}_x(f)) = \textsc{lnk}_x(Ff)$;

and *(iv)* the no-operation, $F(\textsc{nop}) = \textsc{nop}$. Message theories form a category **Msg** with message functors between them.

1.2. Properties of a Message Theory. Our goal is to prove a coherence theorem for message theories. Building up to this result, let us reason with message theories to understand the basic properties that can be derived from the axioms.

Proposition 1.3. *Message theories have a derived operation for each shuffling; moreover, these operations compose as in the multicategory of shufflings.*

$$\mathsf{shuf}_{\sigma}^{\Gamma_1,\dots,\Gamma_n} : \mathbb{M}(\Gamma_1) \times \ldots \times \mathbb{M}(\Gamma_n) \to \mathbb{M}([\Gamma_1, \dots, \Gamma_n]_\sigma)$$

Proof. We have defined an operation for binary and nullary shufflings, and we have defined them to compose exactly as shufflings do. Because shufflings form a malleable multicategory, each n-ary shuffling can be recovered uniquely from the binary and nullary shufflings. □

Proposition 1.4. *A session can always send later and receive sooner, but it cannot send sooner nor receive later. Formally, there exist derived operations*

$$\mathsf{wait}_X^{\Gamma,\Delta,\Psi} : \mathbb{M}(\Gamma, X^\bullet, \Delta, \Psi) \to \mathbb{M}(\Gamma, \Delta, X^\bullet, \Psi),$$
$$\mathsf{rush}_X^{\Gamma,\Delta,\Psi} : \mathbb{M}(\Gamma, \Delta, X^\circ, \Psi) \to \mathbb{M}(\Gamma, X^\circ, \Delta, \Psi).$$

Proof. We can construct the derivation trees of both operations. They both spawn a new channel, shuffle its ends to the origin and target position, and they connect the channel.

$$\dfrac{\dfrac{\Gamma, X^\bullet, \Delta, \Psi \qquad \dfrac{}{X^\circ, X^\bullet}\,(\textsc{spw})}{\Gamma, X^\bullet, X^\circ, \Delta, X^\bullet, \Psi}\,(\textsc{shf})}{\Gamma, \Delta, X^\bullet, \Psi}\,(\textsc{com}) \qquad\qquad \dfrac{\dfrac{\Gamma, \Delta, X^\circ, \Psi \qquad \dfrac{}{X^\circ, X^\bullet}\,(\textsc{spw})}{\Gamma, X^\circ, \Delta, X^\circ, X^\bullet, \Psi}\,(\textsc{shf})}{\Gamma, X^\circ, \Delta, \Psi}\,(\textsc{com})$$

Figure 3. Derivation of wait and rush.

The explicit construction is in Figure 3. Note how, thanks to polarization, it is not possible to use the same technique to *send sooner* nor *receive later*. In fact, we can reason by contradiction that sending sooner, $\mathbb{M}(\Gamma, \Delta, X^\bullet, \Psi) \to \mathbb{M}(\Gamma, X^\bullet, \Delta, \Psi)$, is impossible: the only possible operations we can apply in an arbitrary message theory to an arbitrary session are *shufflings with a spawned channel* and *connections*; we require at least a connection to eliminate the $X^\bullet$, but because the X° must come from shuffling a spawned channel, the corresponding $X^\bullet$ must be placed strictly after it. □

Proposition 1.5. *In particular, we can always swap the order of objects with the same polarity,*

$$\mathsf{swap}_\circ^{\Gamma;X;Y;\Delta} \colon \mathbb{M}(\Gamma, X^\circ, Y^\circ, \Delta) \to \mathbb{M}(\Gamma, Y^\circ, X^\circ, \Delta),$$
$$\mathsf{swap}_\bullet^{\Gamma;X;Y;\Delta} \colon \mathbb{M}(\Gamma, X^\bullet, Y^\bullet, \Delta) \to \mathbb{M}(\Gamma, Y^\bullet, X^\bullet, \Delta).$$

These are self-inverses, forming not only braidings but symmetries in the underlying monoidal category of positively or negatively polarized objects.

Proof. Swaps are constructed from rushing and waiting (Proposition 1.4); explicitly,

$$\mathsf{swap}_\circ^{\Gamma;X;Y;\Delta} = \mathsf{rush}_Y^{\Gamma,X^\circ,\Psi}, \quad \text{and} \quad \mathsf{swap}_\bullet^{\Gamma;X;Y;\Delta} = \mathsf{wait}_Y^{\Gamma,Y^\bullet,\Psi}.$$

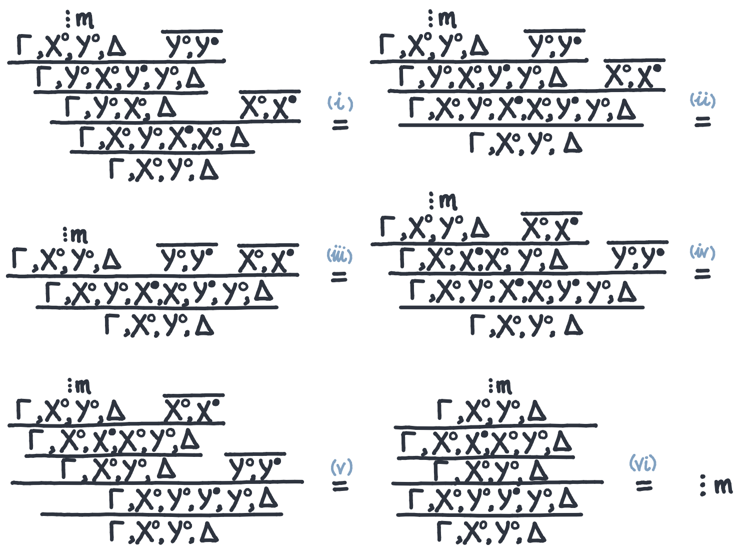


Figure 4. Swaps are self-inverses.

Let us prove that they are self-inverses; we do so with the negatively-polarized one (Figure 4), the other case is analogous. We reason using naturality of linking, *(i,iv)*, that shuffles compose as shuffles *(ii,iii)* (Proposition 1.3), the interaction between shuffling and spawning *(v)*, and the duality between spawning and linking *(vi)*. □

Proposition 1.6. *We can link sooner or later without changing the result. Formally, the equations in Figure 5 hold.*

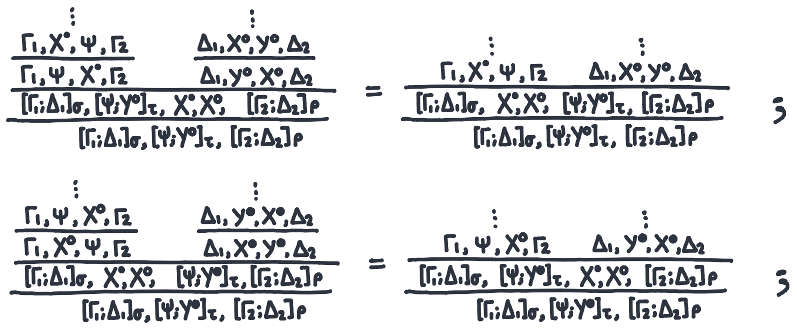


FIGURE 5. Linking sooner or later does not change the restult.

PROOF. We will prove the first one (Figure 6), the second one follows analogously. We reason using *(i)* the definition of wait; *(ii,iii)* that shuffles compose as shuffles (Proposition 1.3); *(iv)* naturality of linking; and *(v)* that swaps are self-inverses (Proposition 1.5). □

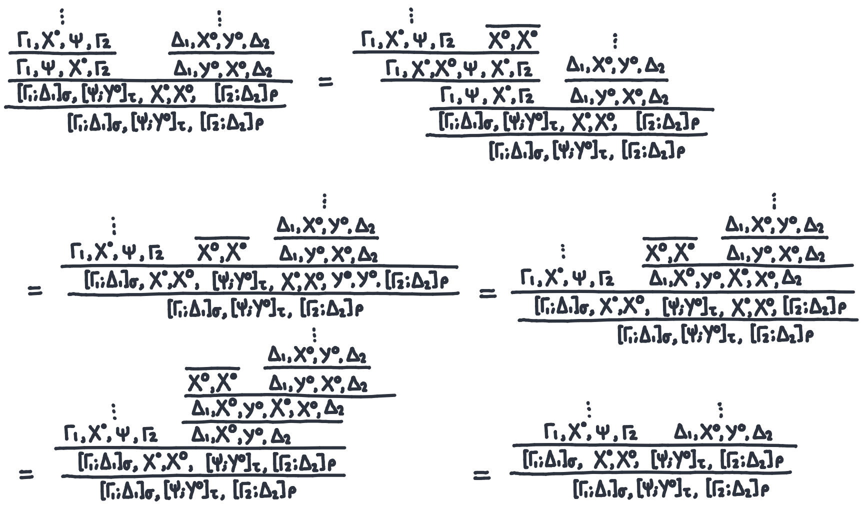


FIGURE 6. Proof of Proposition 1.6.

Proposition 1.7. *Any spawning factors as the spawning of a single channel followed by a shuffling. Formally,*

$$\textsc{spw}^{\Gamma,\Delta}_{X}(m) = \textsc{shf}_{\Gamma,X^{\circ},X^{\bullet},\Delta}(m, \textsc{spw}^{;}_{X}(\textsc{nop})).$$

Proof. This is a direct consequence of Axioms (3a,3b). □

Theoretically, it would be possible to reason with message theories at the level of derivations. However, as we have done through this text, we will try to find better categorical semantics and a better combinatorial expression of the free message theory. We will be most interested in the categorical semantics of message theories and how do they interplay with process theories, in the sense of monoidal categories. We introduce specialized semantics in terms of physical monoidal multicategories, which are another instance of the idea of partially representing a physical duoidal category. For all of this, we will need a coherence theorem.

1.3. Coherence for Message Theories. Symmetric monoidal categories are not *perfectly coherent*: easily, we can find that there are two formally well-typed structure maps $A \otimes A \to A \otimes A$, the identity and the swap. However, what is indeed true is that any two *distinctly typed* structure maps in the free symmetric monoidal category are equal. "Distinct typing" is a notion that only makes sense in the free symmetric monoidal category over some generators; it means that the generators comprising the lists that are our objects appear only once with each variance: arrows $A \otimes B \otimes C \to B \otimes C \otimes A$ are distinctly typed, but arrows $A \otimes A \to A \otimes A$ are not, because A appears twice with each variance.

Shufflings satisfy a similar form of coherence: there is a unique way of shuffling two words into a third one if these words are distinctly typed. This section proves that message theories satisfy the same form of coherence. It is not true that any two parallel formal arrows are equal in any message theory: for instance, there are two ways of deriving X°, X° from X° and X°. It is true, however, that there is a unique arrow for any distinctly typed domains and codomain.

Theorem 1.8. *Message theories are coherent. In the free message theory over a set of objects, there is at most a single derivation between any distinctly typed premises and conclusion.*

Proof. Consider distinctly typed premises and conclusion. There must be three different classes of types in this derivation: *(1)* those that appear twice on the conclusions with different polarity, *(2)* those that appear twice on the premises with different polarity, and *(3)* those that appear once in the premises and once in the conclusions with the same polarity.

We will construct a non-unique normal form for derivations in a message theory, taking into account each one of these cases.

(1) In the first case, the types must have been created by spawning a channel. Using naturality of spawning (Axioms 3a, 3b, 5c), we can move these spawning operations to be shuffled at the end of the derivation. Note that it is not true that we can move *all* spawning to the end of the derivation, but if the types appear only in the conclusions, then we can always do so.
(2) In the second case, these variables must get linked to each other. We can always move the linkings to the end of the derivation (before spawning new objects) using again the naturality of the linkings (Axioms 2a, 2b, 5a).
(3) For the third case, only a shuffling, rushing and waiting can be involved (any linking or spawning that does not involve rushing or waiting has already been moved to the end). Rushing and waiting can be moved to the beginning of the derivation because of naturality of spawning and linking (Axioms 2a, 2b, 3a, 3b).

All of this argues that we can always factor a derivation in a message theory as *(i)* rushing and waiting, *(ii)* a shuffle, *(iii)* linkings, *(iv)* spawnings and shufflings of new variables; but we have not yet shown that this derivation is unique.

Imagine that we know the premises $(\Gamma_1, \dots, \Gamma_n)$ and the conclusion (Δ) of a derivation in a message theory. Let us argue that there is at most a unique derivation between these two.

(1) Removing the objects that appear twice in the conclusion (Δ), we obtain a new conclusion (Δ_1); there is, at most, a unique way of getting from (Δ_1) to (Δ) using spawnings and shufflings: spawning all the possible variables in any order and employing the only possible shuffle (using Proposition 1.3, axioms 1a, 1b, 1c); the order of spawning does not matter because shuffles are symmetric (Axioms 1c, 5c).
(2) Adding all of the objects that get linked and appear twice with different polarities on the premises, in any order, we obtain a new conclusion (Δ_2). There is a unique way of getting from (Δ_2) to (Δ_1) because of the interchanging axioms of linking (Axioms 5b and 5d). The only obstruction to uniqueness here is that we could choose different conclusions (Δ_2) depending on where they place the variables that will be linked; however, we have already shown that all possible choices lead to the same result (Proposition 1.6).
(3) We are left with a shuffle and some rushings and waitings. From the conclusion (Δ_2) we obtain now a sequence of premises $(\Gamma'_1, \dots, \Gamma'_n)$ that are the same as the original premises $(\Gamma_1, \dots, \Gamma_n)$ with the only difference that the objects appear ordered as in the conclusion (Δ_2). There is a unique possible shuffling from $(\Gamma'_1, \dots, \Gamma'_n)$ to (Δ_2).

(4) Finally, for each premise, there will be rushings and waitings. Rushings and waitings interchange (Proposition 1.5), and so there is a unique way of going from the original premise Γ_i to the new premise Γ'_i.

A summary of the proof can be found in Figure 7: Δ_1 is determined from Δ; Δ_2 is not, but all the possible choices lead to the same result (Figure 5); $\Gamma'_1, \dots, \Gamma'_n$ are determined from Δ_2 and $\Gamma_1, \dots, \Gamma_n$; we have argued that in each one of the steps of the proof there is a single possible derivation. □

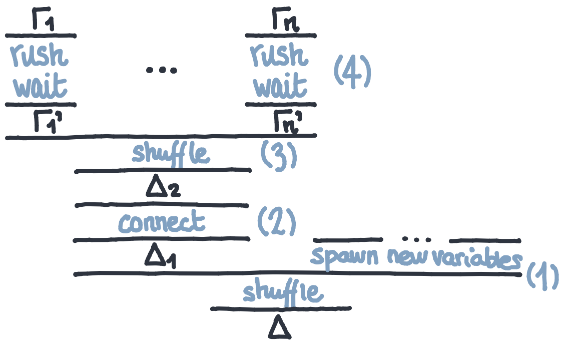


FIGURE 7. Schema for the proof of uniqueness.

Bibliography. Honda pioneered *binary session types* in the 90s [Hon93]; and further work with Yoshida and Carbone extended them to the multi-party case [HYC08]. Session types [Hon93, HYC08] are the mainstay type formalism for communication protocols, and they have been extensively applied to the π-calculus [SW01]. Our approach is not set up to capture all of the features of a fully-fledged session type theory [KPT96]. Explicitly, our framework of message theories can be compared to asynchronous session types without choice. Arguably, this makes it more general: it always provides a universal way of implementing send (A°) and receive ($A^{\bullet}$) operations in an arbitrary process theory represented by a monoidal category. For instance, recursion and the internal/external choice duality [GH99, PS93] are not discussed, although they could be considered as extensions in the same way they are to monoidal categories: via trace [Has97] and linear distributivity [CS97b].

2. Physical Monoidal Multicategories, and Shufflings

Physical monoidal multicategories are physical duoidal categories [Spi13] where the parallel tensor ($\otimes$) is not representable. They follow the same idea of produoidal categories, but they will be a better framework for message passing. The theory of physical monoidal multicategories will need of three ingredients: *(i)* symmetric multicategories, which correspond to the symmetry of the parallel tensor ($\otimes$); *(ii)* monoidal multicategories, which correspond to the half-representable sequential tensor ($\triangleleft$); and *(iii)* normality, which we will need to reinterpret in this setting.

2.1. Symmetric Multicategories.

Symmetric multicategories [BD98, CS09, Shu16] are to multicategories what symmetric monoidal categories are to monoidal categories. Let us write $\sigma \in S(n)$ for an element of the permutation group on n points. Let us write $\sigma_1 + \sigma_2 \in S(n+m)$ for the disjoint union of two permutations, $\sigma_1 \in S(n)$ and $\sigma_2 \in S(m)$. Let us write as $\sigma \wr (k_1, \dots, k_n) \colon k_{\sigma 1} + \dots + k_{\sigma n} \to k_1 + \dots + k_n$ the thickening of a permutation $\sigma \in S(n)$ to a permutation $S(k_1 + \dots + k_n)$ that applies it *considering each block* of k_n elements separately.

Definition 2.1. A *symmetric multicategory* is a multicategory $\mathbb{M}$ together with the following family of functions

$$\sigma^* \colon \mathbb{M}(X_{\sigma 1}, \dots, X_{\sigma n}; Y) \to \mathbb{M}(X_1, \dots, X_n; Y), \text{ for each } \sigma \in S(n),$$

that moreover satisfy the following axioms: *(i)* functoriality, $(\tau \fatsemi \sigma)^*(f) = \tau^*(\sigma^*(f))$ and $\mathrm{id}^*(f) = f$; *(ii)* preservation of disjoint unions, $\sigma_1^*(f_1) \fatsemi_1 \dots \fatsemi_{n-1} \sigma_n^*(f_n) \fatsemi_n g = (\sigma_1 + \dots + \sigma_n)(f_1 \fatsemi_1 \dots \fatsemi_{n-1} f_n \fatsemi_n g)$; and *(iii)* naturality, $f_1 \fatsemi_1 \dots \fatsemi_{n-1} f_n \fatsemi_n \sigma^*(g) = (\sigma \wr (k_1, \dots, k_n))^*(f_{\sigma 1} \fatsemi_1 \dots \fatsemi_{n-1} f_{\sigma n} \fatsemi_n g)$, for any f_n having arity k_n.

In physical monoidal multicategories, symmetry appears with the non representable parallel tensor ($\otimes$); while the sequential tensor will still form a representable monoidal category, this is the notion of monoidal multicategory.

2.2. Monoidal Multicategories.

Multicategories helped us describe typed algebras: the objects of the multicategory were the types, and the multimorphisms were the operations we were allowed to use in that algebra. As we saw with produoidal categories, we need a second dimension if we want to study parallelism: what happens when the types themselves form a monoid? Monoidal multicategories are the monoidal version of algebra.

A *monoidal category* was a 2-monoid on the 2-category of categories, functors and natural transformations. Analogously, a *monoidal multicategory*, sometimes called a *virtual duoidal category* [Shu17] or *monoidal operad*, is a 2-monoid of the 2-category **Mult** of multicategories, multifunctors, and multinatural transformations.

Definition 2.2. A *monoidal multicategory* $(\mathbb{M}, \triangleleft, N, m, i, ⨾)$ is a multicategory $(\mathbb{M}, ⨾)$ with a tensor and a unit both on objects $(\triangleleft) \colon \mathbb{M}_{obj} \times \mathbb{M}_{obj} \to \mathbb{M}_{obj}$ and $N \colon \mathbb{M}_{obj}$, and a tensor and unit on multimorphisms,

$$m_n \colon \mathbb{M}(X_1, ..., X_n; Y) \times \mathbb{M}(X'_1, ..., X'_n; Y) \to \mathbb{M}(X_1 \triangleleft X'_1, ..., X_n \triangleleft X'_n; Y \triangleleft Y'),$$
$$n_n \colon 1 \to \mathbb{M}(N, \overset{n}{\dots}, N; N).$$

These are associative and unital up to multinatural transformations $\alpha_{X,Y,Z} \in \mathbb{M}(X \triangleleft (Y \triangleleft Z); (X \triangleleft Y) \triangleleft Z)$, $\lambda_X \in \mathbb{M}(I \triangleleft X; X)$ and $\rho_X \in \mathbb{M}(X \triangleleft I; X)$, satisfying the pentagon and triangle equations.

Remark 2.3. Any monoidal multicategory has an underlying monoidal category. Multifunctors between representable multicategories are lax monoidal functors, and multinatural transformations are lax monoidal transformations. This implies that a representable monoidal multicategory is exactly a duoidal category [Shu17].

2.3. Physical Monoidal Multicategories. Bringing the notion of normality to the context of monoidal multicategories is not completely trivial: here, there is no map between the two units $I \to N$. Instead, what we will have is a map between the hom-sets, given by the precomposition of the multimap $n_0 \in \mathbb{M}(;N)$ that we have because of the monoidality of the unit.

Definition 2.4. A *physical monoidal multicategory* is a symmetric multicategory $(\mathbb{M}, ⨾, \mathrm{id})$ that is moreover monoidal forming a virtual duoidal category $(\mathbb{M}, \triangleleft, N)$ and that contains an invertible composition with the unit map. That is, the following composition is an isomorphism,

$$(n_0 ⨾ \bullet)^{-1} \colon \mathbb{M}(X_0, \overset{n}{\dots}, X_n; Y) \to \mathbb{M}(X_0, \dots, N, \dots, X_n; Y).$$

Remark 2.5. We know the structure we need, but we will need to be able to compute with it. A better combinatorial description of physical monoidal multicategories will be possible once we characterize the free ones. The next section shows that the physical monoidal multicategory of shufflings is the free normal symmetric monoidal multicategory over a set of objects.

2.4. Shuffling. Shuffling events is the principle on which we have based our definition of message theories. The first step towards endowing them with categorical semantics is to show that shuffling arises naturally in a categorical setting: if physical duoidal categories represented inclusions of posets, physical monoidal multicategories will represent inclusions of linear posets, which correspond to shuffling.

Definition 2.6. Let A be an alphabet. The monoidal multicategory of *shuffling words*, wShuf_A, has as objects the set A^* of words on the alphabet A. The multimorphisms $\mathsf{wShuf}_A(w_1, \dots, w_n; w)$ are precisely the shufflings of the letters of the words $w_1, \dots, w_n$ that result into the word w. The monoidal tensor is given by concatenation of words.

Remark 2.7. This multicategory is not posetal: for instance, there exist two multimaps $\mathsf{wShuf}_A(a, a; aa)$ for any letter $a \in A$. More generally, the number of shuffles of a repeated letter is given by the binomial coefficients,

$$\#\mathsf{wShuf}(a^{n_1}, \dots, a^{n_k}; a^{n_1+\dots+n_k}) = \frac{(n_1 + \dots + n_k)!}{n_1! \dots n_k!}.$$

However, it is true that, if each letter appears at most once on the component words, then the shuffle, if it exists, must be unique. For instance, there exists a unique shuffle $\mathsf{wShuf}(xy, wz; xwyz)$, and so we can refer to it without explicitly specifying it. In other words, there exists at most one morphism between formal distinctly-typed expressions; shuffles are coherent.

Remark 2.8. This monoidal multicategory is symmetric and normal. In other words, it is a physical monoidal multicategory.

Proposition 2.9. *Shuffling words are the free physical monoidal multicategory on a set of objects.*

Proof. We have already shown how they form a physical monoidal multicategory; we need to show that it is the free one. Let $\mathbb{V}$ be any physical duoidal category, for each map $A \to \mathbb{V}_{obj}$ there must exist a unique physical monoidal multifunctor $\mathsf{wShuf}_A \to \mathbb{V}$ that factors on objects through the former map.

First, we argue that the physical monoidal multifunctor is forced. Any shuffle can be reconstructed from the operations of a physical monoidal multicategory, and so any shuffle must be mapped accordingly. For instance, if we want to reconstruct the shuffle $\mathbb{V}(xy, z; xzy)$, we multiply together: first $\mathbb{V}(x, i; x)$, then $\mathbb{V}(i, z; z)$, and finally $\mathbb{V}(y, i; y)$.

Let us do this in general. Assume a shuffle $\mathsf{wShuf}(w_1, \dots, w_n; w)$, where the words are $w_j = a_{j,1} \dots a_{j,k_j}$. We position each one of the letters in their position: for instance, if $a_{j,i}$ appears in the jth-word, we can use the normalization isomorphism

$$\mathbb{V}(a_{j,i}; a_{j,i}) \cong \mathbb{V}(I, \dots, a_{j,i}^{(j)}, \dots, I; a_{j,i})$$

to position it in the jth input, via the identity multimorphism. Then, we can multiply all of the letters forming the shuffle, using monoidality, and obtain the desired shuffle in any physical monoidal multicategory,

$$\prod_{a_{j,i} \in w} \mathbb{V}(I, \dots, a_{j,i}^{(j)}, \dots, I; a_{j,i}) \to \mathbb{V}(w_0, \dots, w_n; w).$$

We have used exactly one factor for each letter of the resulting word.

Finally, we need to prove that this assignment is well-defined and that it defines a physical monoidal multifunctor. These will be both a consequence of the same fact: there exists at most a unique shuffle between words with different letters and there exists at most a unique formally distinctly typed map between any poset shapes. To prove this, we use the characterization of the free duoidal category as poset shapes: there is a single formally well-typed map between any two poset shapes, so there is a single formally well-typed map between any two duoidal expressions; in particular, there exists at most one multimorphism between any words with distinct letters in a physical monoidal multicategory. This renders the map well-defined: there is at most one way of constructing any shuffle, and we have shown that it is possible. On the other hand, this also makes the map a physical monoidal multifunctor: all the equations are formal and thus they hold automatically. $\square$

Shuffling takes care of the first and most important aspect of message theories: message theories are algebras for the shuffling words physical monoidal multicategory of their objects. However, message theories do have an extra structure: each object is a left dual, naturally with respect to these shufflings – the second ingredient for message theories is *polarization*.

2.5. Bibliography. This formulation of symmetric multicategories is a particular case of Shulman's definition of multicategory over a *faithful cartesian club* [Shu16, §2]. The theory of symmetric multicategories is less developed than the theory of *cartesian multicategories*, which are well-known to correspond to Lawvere theories. Monoidal multicategories are also not particularly explored, but Shulman has a note relating them directly to duoidal categories and proposing the name *virtual duoidal category* for them [Shu17].

3. Polarization

Polarization is not a property of our structures, but more of a particular way of constructing free structures. This is, we do not say that a monoid is "polarized"; but we have constructed free polarized monoids when talking about message theories. Saying that a monoid is polarized would seem to imply that all of its elements have a sign, which is not what happens in the free polarized monoid: generators do appear with a sign, but the words on the monoid are combinations without a particular sign.

Remark 3.1 (Polarization of a monoid)**.** Given any monoid M, we can consider its polarization, $\mathsf{Polar}(M)$, to be the monoid generated by two copies of each element of M, denoted $a^{\bullet} \in \mathsf{Polar}(M)$ and $a^{\circ} \in \mathsf{Polar}(M)$, and quotiented by the equations $a^{\bullet}b^{\bullet} = (ab)^{\bullet}$ and $a^{\circ}b^{\circ} = (ab)^{\circ}$.

Taking seriously this idea, we will not regard polarization as part of an algebraic structure (as it happens with monoidal categories). Instead, polarization will arise as a left adjoint. To quip, *polarization is left adjoint to taking left adjoints.* Let us see first how this works in the context of monoidal categories; we will later extend it to physical monoidal multicategories.

3.1. Monoidal Polarization.

Every monoidal category has a notion of *duality* inside it. It does not suffice to say that some objects *have duals*: a duality is not only a property. Instead, we need to specify the maps that constitute the duality. Dualities in a monoidal category form a monoidal category themselves. Polarization is the left adjoint to taking this category of dualities.

Definition 3.2 (Category of dualities)**.** Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. We define the category of left duals, $\mathsf{Duals}(\mathbb{C})$ to have objects the dualities $(L \dashv R, \varepsilon, \eta)$ of the monoidal category and morphisms

$$(f_L, f_R) \colon (L \dashv R, \varepsilon, \eta) \to (L' \dashv R', \varepsilon', \eta')$$

to be pairs of morphisms $f_L \colon L \to L'$ and $f_R \colon R' \to R$ such that the equations in Figure 8 hold.

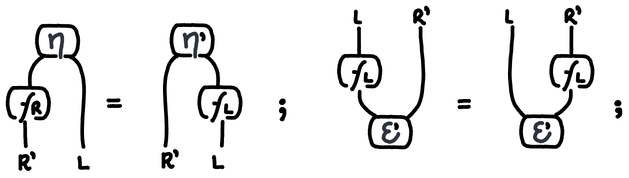


FIGURE 8. Morphism of adjunctions.

The category of left duals is a monoidal category where the tensor of two dualities, $(L \dashv R, \varepsilon, \eta)$ and $(L' \dashv R', \varepsilon', \eta')$, is defined to be

$$(L \otimes L' \dashv R' \otimes R, (\mathrm{id} \otimes \varepsilon \otimes \mathrm{id}) \mathbin{⨾} \varepsilon', \eta \mathbin{⨾} (\mathrm{id} \otimes \eta \otimes \mathrm{id})).$$

Taking the left duals extends to an endofunctor $\mathsf{Duals}\colon \mathbf{MonCat} \to \mathbf{MonCat}$. This endofunctor has a right adjoint $\mathsf{Polar}\colon \mathbf{MonCat} \to \mathbf{MonCat}$.

Definition 3.3 (Polarization)**.** Let $(\mathbb{C}, \otimes, I)$ be a strict monoidal category. Its polarization, $\mathsf{Polar}(\mathbb{C})$, is a monoidal category presented by the following data. It contains two objects, $A^\bullet$ and A°, for each object of the original category $A \in \mathbb{C}_{obj}$, and quotiented by equalities $(A \otimes B)^\bullet = A^\bullet \otimes B^\bullet$ and $(A \otimes B)^\circ = B^\circ \otimes A^\circ$.

It contains two morphisms, $f^\bullet\colon A^\bullet \to B^\bullet$ and $f^\circ\colon A^\circ \to B^\circ$, for each morphism $f\colon A \to B$; it also contains a pair of morphisms $\varepsilon_A\colon A^\bullet \otimes A^\circ \to I$ and $\eta_A\colon I \to A^\circ \otimes A^\bullet$. These are quotiented by equations explicitly asking for functoriality: $f^\bullet \mathbin{⨾} g^\bullet = (f \mathbin{⨾} g)^\bullet$ and $\mathrm{id}^\bullet = \mathrm{id}$, and also $f^\circ \mathbin{⨾} g^\circ = (g \mathbin{⨾} f)^\circ$ and $\mathrm{id}^\circ = \mathrm{id}$. Moreover, they are quotiented by equations asking for a duality $A^\bullet \dashv A^\circ$: that is, $\varepsilon_A \mathbin{⨾} \eta_A = \mathrm{id}$ and $\eta_A \mathbin{⨾} \varepsilon_A = \mathrm{id}$, with the duality being monoidal, and satisfying both $\varepsilon_{A \otimes B} = \varepsilon_A \mathbin{⨾} \varepsilon_B$ and $\eta_{A \otimes B} = \eta_B \mathbin{⨾} \eta_A$, and moreover $\varepsilon_I = \eta_I = \mathrm{id}_I$.

Proposition 3.4. *Polarization is left dual to taking left duals.*

Proof. We already know that $\mathsf{Duals}\colon \mathbf{MonCat} \to \mathbf{MonCat}$ is a functor; we only need to construct a universal arrow $\eta_{\mathbb{M}}\colon \mathbb{M} \to \mathsf{Duals}(\mathsf{Polar}(\mathbb{M}))$. The universal arrow will be given by $\eta(X) = (X^\circ \dashv X^\bullet, \varepsilon, \eta)$, which uses the fact that $X^\circ \dashv X^\bullet$ determines an adjunction. This assignment extends to a functor that sends a morphism $f\colon X \to Y$ to a morphism of dualities $(f, f^*)\colon (X^\circ \dashv X^\bullet, \varepsilon, \eta) \to (Y^\circ \dashv Y^\bullet, \varepsilon, \eta)$, where $f^*\colon Y^\bullet \to X^\bullet$ is the dual of $f\colon X^\circ \to Y^\circ$. This constructs the candidate universal arrow.

Let us show that it is indeed universal. Consider a functor $H\colon \mathbb{M} \to \mathsf{Duals}(\mathbb{N})$ and pick an object $X \in \mathbb{M}_{obj}$ that is sent to $H(X) = (A \dashv A', \varepsilon_A, \eta_A)$. We will show that there is a unique $H^*\colon \mathsf{Polar}(\mathbb{M}) \to \mathbb{N}$ such that $H = \eta_{\mathbb{M}} \mathbin{⨾} \mathsf{Duals}(H^*)$. To make this equation hold, it is necessary that $H^*(X^\circ) = A$ and $H^*(X^\bullet) = B$. Moreover, this forces $H^*(f^\circ) = H(f)$, but also $H^*(\varepsilon_X) = \varepsilon_A$ and $H^*(\eta_X) = \eta_A$. Because $(A \dashv A', \varepsilon_A, \eta_A)$ forms a duality, this assignment satisfies all necessary equations. This determines the value of H^* on all of the generating maps while satisfying all of the equations of $\mathsf{Polar}(\mathbb{M})$. □

3.2. Monoidal Polarization is Not Enough. An important insight of some works into the categorical semantics of message passing is the importance of polarization. Given this, one could expect that the polarization of a process theory is its corresponding message theory. Indeed, this chapter will claim that the polarization of a symmetric monoidal category exhibits the necessary structure to discuss message passing; however, at the same time, it will claim that the algebraic structure that allows it to do so is not that of a polarized monoidal category: it is that of a message theory.

Remark 3.5. Some simpler interleavings of events can be expressed using dualities in a monoidal category. For instance, let us consider the polarization of a symmetric monoidal category: wires with the same sign can be swapped, but wires with different sign cannot. We can interleave two maps $f\colon I \to X^\circ \triangleleft Y^\circ \triangleleft Z^\bullet$ and $g\colon I \to U^\circ \triangleleft V^\bullet$ to produce a map $h\colon I \to X^\circ \triangleleft U^\circ \triangleleft V^\bullet \triangleleft Y^\circ \triangleleft Z^\bullet$ as in Figure 9 – note how we use the adjunction to contort the wires and allow wires to pass over wires with different polarization.

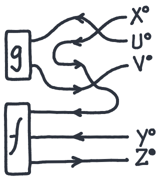


FIGURE 9. A shuffling, using polarization.

In general, thanks to the dualities, a *receiving* resource can be used later, $X^\circ \triangleleft A \to A \triangleleft X^\circ$, and a *sending* resource can be sent sooner, $A \triangleleft X^\bullet \to X^\bullet \triangleleft A$, but not the other way around. These two laws govern which wires can pass over other wires. A natural question, then, is to ask if all possible interleavings can be expressed in the same way: polarizing objects, and using dualities.

Proposition 3.6. *It is not the case that every shuffling of events can be expressed in a category with polarized objects and dualities.*

PROOF. Consider two cells as in Figure 10, with types

$$f\colon I \to X^\bullet \triangleleft Y^\circ \triangleleft Z^\bullet \triangleleft W^\circ, \text{ and } g\colon I \to A^\bullet \triangleleft B^\circ \triangleleft C^\circ \triangleleft D^\bullet \triangleleft E^\bullet \triangleleft F^\circ$$

and imagine we want to interleave them into $A^\bullet \triangleleft B^\circ \triangleleft X^\bullet \triangleleft Y \circ \triangleleft C^\circ \triangleleft D^\bullet \triangleleft Z^\bullet \triangleleft W^\circ \triangleleft E^\bullet \triangleleft F^\circ$ using only the dualities. There are two possible cases: *(i)* we place f before g and we try to position the wires of f correctly; or *(ii)* we place g before f and we try to do the same.

In the first case, we will find that we can pass the wire X° over $A^\bullet \triangleleft B^\circ$ – but the same cannot be done with the wire $Y^\bullet$. Therefore, we are forced to pass the wire $Y^\bullet$ over $C^\circ \triangleleft D^\bullet \triangleleft E^\bullet \triangleleft F^\circ$, using the dualities; but then there will be no way of moving the wire $Z^\bullet$ to its place.

In the second case, the first two wires, $A^\bullet \triangleleft B^\circ$, are automatically correctly placed. The third, C°, must pass over $Z^\bullet \triangleleft W^\circ$ because it could not do the same with $X^\bullet \triangleleft Y^\circ$; so $D^\bullet$ would be forced to do the same, but it cannot. □

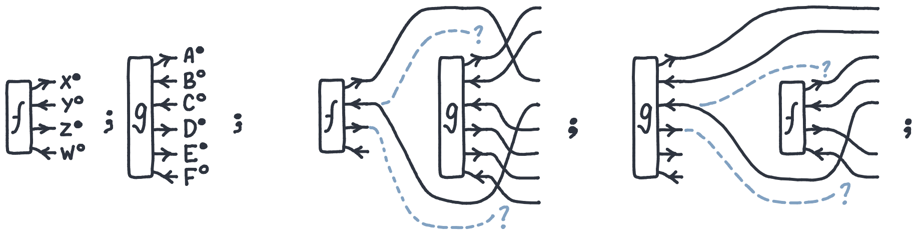


FIGURE 10. A shuffling that cannot be expressed.

Then, if this algebra is not what we are using about the polarization of a monoidal category, what are we using exactly? It happens that the polarization of a monoidal category posesses extra structure: its states (the morphisms without input) can always be split into morphisms that take negative objects on the left and produce positive objects on the right. This property is what makes it possible to express any interleaving of events.

With this in mind, the polarization of a monoidal category seems interesting not only because it produces a category with duals (or, in the work of Nester [Nes21], a proarrow equipment, or a *cornering*), but because it constructs a full message theory. We do not only care about the free polarization, we care about its message theory.

3.3. Polarization of a Physical Monoidal Multicategory. Every physical monoidal multicategory has a notion of dual, inherited from that of its underlying monoidal category. The next section will construct the free polarized physical monoidal multicategory over a set of objects. Let us define here the right adjoint: the functor that picks the set of left adjoints of a physical monoidal multicategory.

Definition 3.7. Let $(\mathbb{M}, \triangleleft, I)$ be a physical monoidal multicategory. We define the set of left duals, $\mathsf{Duals}(\mathbb{M})$, to have objects the dualities $(L \dashv R, \varepsilon, \eta)$ where $\varepsilon \in \mathbb{M}(L \triangleleft R; I)$ and $\eta \in \mathbb{M}(I; R \triangleleft L)$ are 1-to-1 multimorphisms satisfying the adjoint equations, $(\varepsilon \triangleleft \mathrm{id}_L) \fatsemi (\mathrm{id}_R \triangleleft \eta) = \mathrm{id}_L$ and $(\mathrm{id}_R \triangleleft \varepsilon) \fatsemi (\eta \triangleleft \mathrm{id}_R) = \mathrm{id}_R$.

3.4. Bibliography. A categorical treatment of polarization appears in the work of Cockett and Seely [CS07], which points to the connection with Abramsky-Jagadesaan games [AJ94]. "Polarized category" takes a different meaning there: it is a pair of categories endowed with a profunctor between them. However, we do follow the same core idea: using walking adjunctions for sending and receiving.

Nester notices the importance of polarization for message passing [Nes21] via a single-object proarrow equipment; and all the credit for this idea should go there. This was later extended by Nester and Voorneveld [NV23] to include iteration and choice. The reader will find a discussion of its relationship with lenses in a joint paper of this author with Nester and Boisseau [BNR22]. The graphical calculus of proarrow equipments was described by Myers [Mye16]; we can reuse this calculus to explain polarization in monoidal categories. Here, we prefer to avoid discussing polarization through proarrow equipments, noticing that this adjunction already works at the level of monoidal categories.

4. Polar Shuffles

4.1. Polar Shuffles. Polarization in a physical monoidal multicategory leads us to consider polar shuffles: shufflings endowed with a polarization for each one of its elements. This section will show that polar shuffles are the free polarized monoidal multicategory.

The objects of the category of shuffles could be seen as the finite ordinals: finite linear posets with inclusions on each other, preserving the ordering. Similarly, the objects of the category of polar shuffles are polarized finite ordinals.

Definition 4.1. A *polar list* over a set of types T is a list of types $X \in T^*$ endowed with a function $p\colon X \to \{\circ, \bullet\}$. In any polar list, we consider the *negative elements*, $X^\circ = p^{-1}(\circ)$, and the *positive elements*, $X^\bullet = p^{-1}(\bullet)$.

Definition 4.2. A *polar shuffle* over a set of types T, from a multiset of polar lists $X_0, \dots, X_n$ to a single polar list Y, is a bijection

$$f\colon X_1^\bullet + \ldots + X_n^\bullet + Y^\circ \to Y^\bullet + X_1^\circ + \ldots + X_n^\circ$$

preserving the types and such that the directed graph containing all polar lists (as linear posets) and an edge $x \to f(x)$ for each element in $X_1^\bullet + \ldots + X_n^\bullet + Y^\circ$, is acyclic.

For instance, an untyped polar shuffle (or typed over the singleton set) of shape $\mathsf{pShuf}(\circ \bullet \circ\bullet, \circ\circ \bullet \bullet, \circ; \circ\circ \bullet \bullet\circ)$, is given by the following acyclic graph in Figure 11. In black, we depict the edges that come from the graph of a function. In blue, the edges that come from the linear finite posets.

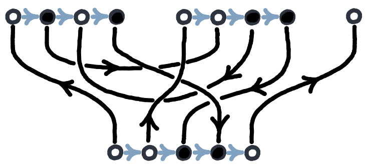
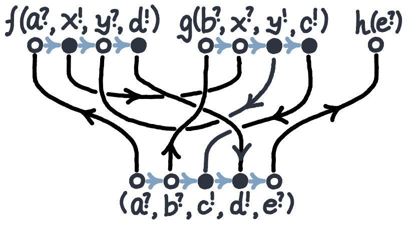


FIGURE 11. A polar shuffle.

4.2. Encoding of polar shuffles. Polar shuffles are ultimately graphs, and they can be encoded as such. We propose a notation suggestive of multiparty session calculi.

The *session encoding* of polar shuffles assigns a variable name to each polar list (say, $f, g, h, \dots$) and to each edge of the graph outside a polar list (say, $a, b, c, x, y, z, \dots$). The encoding of a polar shuffle starts by declaring the list of edges incident to the output polar list, together with their polarization. Then, enclosed in braces, we write the polar lists and the edges that incide on them.

For instance, the encoding of the polar shuffle in Figure 11 is in Figure 12.



```
(a?,b?,c!,d!,e?) {
  f(a?,x!,y?,d!),
  g(b,x?,y!,c!),
  h(e?) }
```

FIGURE 12. Encoding of a polar shuffle.

Remark 4.3. Parsing this notation requires checking whether the graph is acyclic. Checking if a graph is acyclic can be done in linear time on the sum of vertices and edges, $\mathcal{O}(v + e)$. The number of vertices in a polar shuffle is the sum of the lengths of all of the polar lists involved, $e = \#X_1 + \ldots + \#X_n + \#Y$, and each vertex receives at most three edges, giving a bound $e \leq 2v$. This means that checking if a polar shuffle is valid is linear in the length of its polar lists, $\mathcal{O}(\#X_1 + \ldots + \#X_n + \#Y)$.

The second implication of this encoding is that, if we label the vertices of a polar shuffle with types, there exists at most one polar shuffle with any distinctly typed sources and targets.

Proposition 4.4. *Polar shuffles are coherent. There exists at most a single polar shuffle between some distinctly typed polar lists: we say that a polar list is distinctly typed if each variable (each type) appears in the premises and the conclusion exactly twice, each time with a different variance.*

PROOF. In this combinatorial structure, coherence works almost by definition. Note that a polar shuffle is ultimately a bijection satisfying certain extra properties; but the distinct typing already forces where each element must be sent: to the only element with different variance but same type. Whether the polar shuffle exists at all depends on whether it satisfies the acyclicity property. □

4.3. The Multicategory of Polar Shuffles. Polar shuffles form a multicategory, as their shape already suggests. The composition and the rest of the structure follow that of the category of shufflings. For instance, the composition of polar shuffles is defined to be the substitution of the resulting polar list of a shuffle into the input polar list of another shuffle. See Figure 13 for an example.

Proposition 4.5. *The composition of two polar shuffles, $s \in \mathsf{pShuf}(X_1, \ldots, X_n; Y)$ and $t \in \mathsf{pShuf}(Y_1, \ldots, Y, \ldots, Y_m; Z)$, along a polar list Y, is a polar shuffle obtained by substituting the entire graph of the former into the polar list of the latter,*

$$s \mathbin{⨟}_Y t \in \mathsf{pShuf}(Y_1, \ldots, X_1, \ldots, X_n, \ldots, Y_m; Z).$$

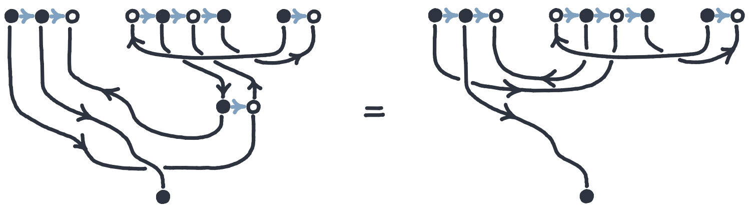


FIGURE 13. Composition of two polar shuffles.

Substituting a polar shuffle into the inputs of a polar shuffle forms again a polar shuffle. That is, composition is well-defined and preserves acyclicity.

PROOF. Composition happens across the resulting polar list of a polar shuffle, which coincides with one of the input polar lists of another polar shuffle; we say that this polar list is the *border*. The proof argues that two acyclic graphs glued along a linear graph are again acyclic.

Let us prove this by contradiction. Imagine there was a cycle in the multicategorical composition of polar shuffles. It must contain edges on both components of the composition, simply because each component is itself acyclic. This means that the cycle should cross the border between both components of the polar shuffle at least twice and always an even number of times – it must take two different directions.

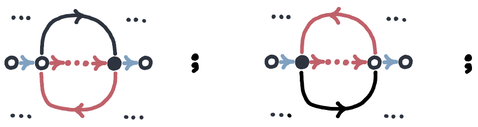


FIGURE 14. Composition along the borders of two polar shuffles.

Because the border is a linear poset, it must split the cycle (at least) in two parts, creating two undirected cycles to the two sides of the border. At least one of these two is forced to be directed, thus contradicting acyclicity on that side of the composition. □

Proposition 4.6. *The tensoring of two polar shuffles of the same arity,*

$$s \in \mathsf{pShuf}(X_0, \dots, X_n; Y) \text{ and } t \in \mathsf{pShuf}(X'_0, \dots, X'_n; Y'),$$

is a polar shuffle on the pointwise concatenation of the polar lists

$$(s \triangleleft t) \colon \mathsf{pShuf}(X_0 X'_0, \dots, X_n X'_n; YY'),$$

defined by the disjoint union of the two acyclic graphs determining the polar shuffles. The tensoring of two polar shuffles is well-defined and it is again a polar shuffle. See Figure 15 for an example.

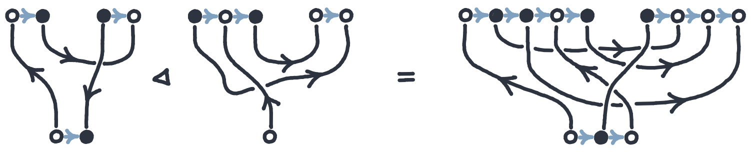


FIGURE 15. Parallel composition of polar shuffles.

PROOF. The graph of the tensoring is the disjoint union of two acyclic graphs, together with the edges determined by the polar lists. The directed edges coming from the polar lists always go from the first graph to the second; thus, they will not create cycles and the resulting graph with be acyclic. □

THEOREM 4.7. *Polar shuffles over a set of types are the morphisms of a physical monoidal multicategory,* pShuf*, that has the polar lists as objects.*

PROOF. We define the identity polar shuffle on a polar list to be the identity bijection linking each sign to itself. The identity polar shuffle is acyclic because the identity bijection preserves the linear ordering. We have already defined the composition and shown that it is acyclic in Proposition 4.5. Associativity of composition follows from associativity of glueing graphs; unitality follows by construction. We have already defined the tensoring in Proposition 4.6, and the tensor on objects is the concatenation of polar lists. The unit for the tensoring is the empty polar list, and because it does not appear in a polar shuffle, it makes the monoidal multicategory normal. □

4.4. Message Theories are Algebras of Polar Shuffles. What makes polar shuffles relevant to the discussion of message-passing is that they promise us a better syntax for message theories. Instead of thinking of the operations of a message theory as generated by a few primitives satisfying equations, we can give them a combinatorial characterization in terms of polar shuffles. This section shows that message theories are precisely the algebras for the operations given by polar shuffles.

Definition 4.8. An algebra over a multicategory, $(\mathbb{M}, ⨾, \mathrm{id})$, is the assignment of a set $S(X)$ to each object $X \in \mathbb{M}_{ob}$, and the assignment of a function

$$S(f) \colon S(X_1) \times \ldots \times S(X_n) \to S(Y),$$

for each multimorphism $f \in S(X_1, \dots, X_n; Y)$. The assignment must preserve compositions, $S(f \mathbin{⨟}_X g) = S(f) \mathbin{⨟}_X S(g)$ and identities, $S(\mathrm{id}) = \mathrm{id}$. Alternatively, it is a multifunctor to the cartesian monoidal category of sets and functions.

Proposition 4.9. *Message theories are precisely the algebras of the free polarized physical monoidal multicategory over their respective sets of types. In other words, the derivations of a message theory form the free polarized physical monoidal multicategory over its set of types.*

Proof. This will follow almost by definition. The definition of message theories includes an operation for each shuffling, and these operations compose as shufflings (Proposition 1.3 and Axioms 1a, 1b, and 1c); equivalently, this is saying that message theories are in particular algebras of the multicategory of shufflings, the free physical monoidal multicategory over their set of types.

The rest of the axioms of message theories are exactly asking that each object is a left adjoint: the spawning and linking operations are describing the unit and the counit of the adjunction; the rest of the axioms are saying that: the unit of the adjunction is natural with respect to shufflings (Axioms 2a and 2b); the counit of the is natural with respect to shufflings (Axioms 3a and 3b); unit and counit satisfy the snake equations (4a and 4b); and they are natural with respect to each other (Axioms 5a, 5b, 5c and 5d). □

Corollary 4.10. *Polarized physical monoidal multicategories are coherent; there exists at most one multimorphism between any distinctly typed objects of the free physical monoidal multicategory over some objects.*

Proof. This is now a consequence of Proposition 4.9 and Theorem 1.8. The derivations of a message theory form the free polarized physical monoidal multicategory, but we have already shown that they are coherent. □

Theorem 4.11. *Polar shuffles form the free polarized physical monoidal multicategory over a set of types.*

Proof. Polar shuffles are coherent (Proposition 4.4), and polarized physical monoidal multicategories are coherent as well (Corollary 4.10); this simplifies the proof because, to show that they coincide, we only need to show that a polar shuffle between some types exists if an only if a multimorphism in the free polarized physical monoidal multicategory exists.

If a multimorphism of a certain type exists in the free polarized physical monoidal multicategory, then it exists in all polarized physical monoidal multicategories and, in particular, in polar shuffles (Theorem 4.7).

Let us prove the converse implication: if a polar shuffle between some types does exist, then there is a multimorphism in the free polarized physical monoidal multicategory between these types. For this, we will use that a polar shufflecan be always factored (not necessarily uniquely!) in the following way: *(i)* we exchange positions inside each polar list to get them to their final relative position; *(ii)* we use a series of spawnings, or polar shuffles $I \to X^\circ \lhd X^\bullet$; *(iii)* we use a pure, non-polarized shuffle; and *(iv)* some final linkings of type $X^\bullet \lhd X^\circ \to I$. This is easy to verify topologically: we can always 'pull down the linkings' and 'pull up' the spawnings, and we can always factor any shuffle in two parts – a pure shuffle and a shuffle internal to each one of the components. For instance, consider the following example in Figure 16, adapted from Figure 15 with an extra spawning.

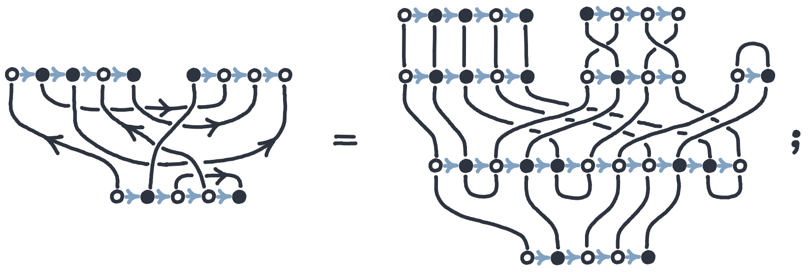


Figure 16. Factored polar shuffle.

Spawning, linking, shuffling, waiting and rushing are all operations on a physical monoidal multicategory (see Section 1); so this proves that there is at least one multimorphism in the free physical monoidal multicategory with these types. We do not care about the specific choice of multimorphism because polar shuffles are coherent (Proposition 4.4), and polarized physical monoidal multicategories are coherent as well (Corollary 4.10) □

Corollary 4.12. *Message theories are the algebras of the physical monoidal multicategory of polar shuffles.*

Proof. We will use that polar shuffles are the free polarized physical monoidal multicategory (Theorem 4.11) and that message theories are precisely the algebras of the free polarized physical monoidal multicategory over their objects (Proposition 4.9). □

4.5. Bibliography. The definition – and the notation – of polar shuffles takes inspiration from a different concept: Hughes' *partial leaf functions* [Hug12]. Partial leaf functions are the **Int**-construction – the free compact closed category over a traced monoidal category – applied to the category of finite sets and partial functions [AM99]; here we follow a similar idea, but we work over finite sets and bijections, which are not traced. Not only the **Int**-construction, but also the idea of shuffling two traces point back to game semantics [MS18].

5. Processes versus Sessions

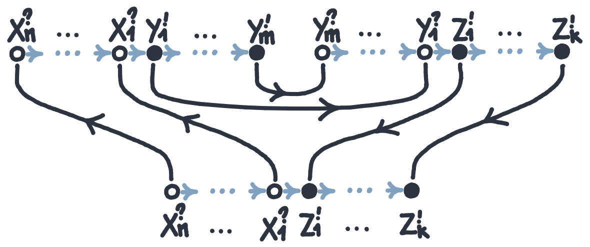
5.1. Processes of a message theory. Inside of a message theory, we call *processes* the sessions that happen in two parts: *(i)* first they ask for some indeterminate number of resources (possibly zero), $X_1^\circ, \dots X_n^\circ$ and *(ii)* then, they give some indeterminate number of resources (possibly zero), $Y_1^\bullet, \dots, Y_m^\bullet$. This simple definition builds a symmetric monoidal category inside any message theory, and we will find a left adjoint to this construction.

Proposition 5.1. *Let* $\mathbb{M}$ *be a message theory. There exists a symmetric monoidal category,* $\mathsf{Proc}(\mathbb{M})$*, whose objects are the lists of objects of the message theory,* $\mathsf{Proc}(\mathbb{M})_{obj} = \mathbb{M}^*_{obj}$*, and whose morphisms are the sessions that first ask some resources and then provide some resources,*

$$\mathsf{Proc}(\mathbb{M})(X_1 \otimes \dots \otimes X_n; Y_1 \otimes \dots \otimes Y_m) = \mathbb{M}(X_n^?, \dots X_1^?; Y_1^!, \dots, Y_m^!).$$

Note how we reverse the order of inputs; this will make reasoning easier even if it is unnecessary: the monoidal category will be symmetric in any case.

Proof. Let us define the composition and identities. Composition is given by the polar shuffle that connects the middle outputs to inputs, see Figure 17; because message theories are algebras of polar shuffles, such a polar shuffle defines an operation, composition, that takes two sessions to a third one.



$X_n^?$ ... $X_1^?$ $Y_1^!$ ... $Y_m^!$ $Y_m^?$ ... $Y_1^?$ $Z_1^!$ ... $Z_k^!$
$X_n^?$ ... $X_1^?$ $Z_1^!$ ... $Z_k^!$

Figure 17. Composition of processes of a message theory.

Identities are created by spawning channels, see Figure 18. Again, because message theories are algebras of polar shuffles, each message theory must contain a constant given by the polar shuffle that spawns a list of channels. Composition and identities are associative and unital: it can be checked from the definition that we are using the duality from spawning and connecting channels.

Symmetries are given by the polar graph that spawns a channel for each one of the objects and then shuffles them so that the inputs and the outputs are divided in two blocks and positioned in reverse order, see Figure 23.

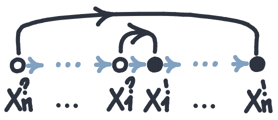


FIGURE 18. Identity process of a message theory.

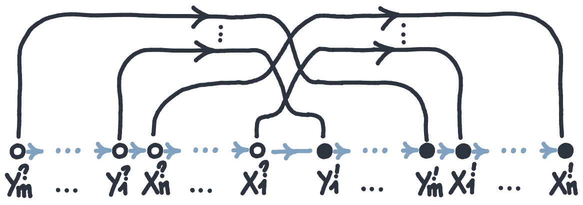


FIGURE 19. Symmetries of processes in a message theory.

Tensoring is given by the polar shuffle that preserves all outputs and inputs but passes the outputs of a process pass the inputs of the other, see Figure 20. This operation is again associative and unital with the empty polar shuffle that represents the monoidal unit.

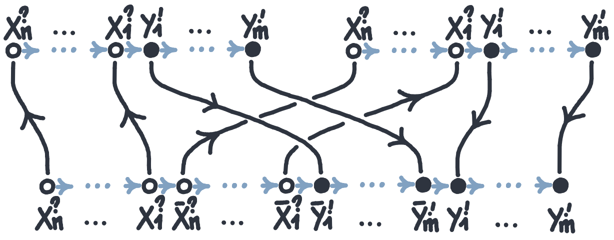


FIGURE 20. Tensor of processes in a message theory.

It concludes the proof to check, by following the connections of the polar shuffles, that tensoring behaves functorially with composition so that the interchange law of monoidal categories holds. □

Proposition 5.2. *The construction of the symmetric monoidal category of processes extends to a functor*

$$\mathsf{Proc}\colon \mathbf{Msg} \to \mathbf{SymMonCat}_{\mathsf{Str}}.$$

Proof. We will show that any message functor $F\colon \mathbb{M} \to \mathbb{N}$ induces a strict symmetric monoidal functor $\mathsf{Proc}(F)\colon \mathsf{Proc}(\mathbb{M}) \to \mathsf{Proc}(\mathbb{N})$. Because the category of processes is freely generated on objects, it suffices to explain that the object $X \in \mathbb{M}_{ob}$ is sent to the object $FX \in N_{ob}$. On morphisms, we already have a map

$$F\colon \mathbb{M}(X^?_n, \dots, X^?_1, Y^!_1, \dots, Y^!_m) \to \mathbb{N}(FX^?_n, \dots, FX^?_1, FY^!_1, \dots, FY^!_m)$$

that gets reinterpreted as a map

$$\mathsf{Proc}(\mathbb{M})(X_1 \otimes \dots \otimes X_n; Y_1 \otimes \dots \otimes Y_m) \to \mathsf{Proc}(\mathbb{N})(F(X_1 \otimes \dots \otimes X_n); F(Y_1 \otimes \dots \otimes Y_m)).$$

Composition, identities and tensoring are operations constructed as polar shuffles, and so they must be preserved by a message functor; this means that $\mathsf{Proc}(F)$ becomes a strict monoidal functor. □

5.2. Sessions of a process theory. We will now construct message sessions over an arbitrary process theory, and we will do so in a minimalistic theory. Message passing consists of two effects: *sending* and *receiving*. Premonoidal categories already are a framework for effectful computation in process theories, so we employ them here.

Definition 5.3. The effectful category of *sessions* over a strict symmetric monoidal category $\mathbb{C}$ is the effectful category $\mathbb{C} \to \mathsf{Session}(\mathbb{C})$ generated by

(1) all of the morphisms of the original monoidal category, $\mathbb{C}$, quotiented by the equations of the original monoidal category, as pure morphisms;
(2) and a pair of *send* and *receive* generators for each object $X \in \mathbb{C}_{obj}$ imposing no further equations. We write these generators as $(\circ)_X : X \to I$ and $(\bullet)_X : I \to X$.

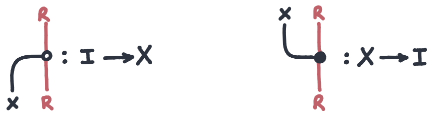


Figure 21. Session runtime generators.

This naive theory of message passing on top of a monoidal category may remind us of the combs that we were studying before: instead of using incomplete diagrams, we are marking the interchanges explicitly now. This intuition can be made formal: we will now prove that combs of type $X^\bullet_1 \triangleleft \dots \triangleleft X^\bullet_n \to \binom{A}{B}$ correspond to sessions $A \to B$ where the events are exactly $X^\bullet_1, \dots, X^\bullet_n$, and happen in that specific order. First, note that we can define a sequence of events associated with a particular session.

Definition 5.4. The *sequence of events* of a session is the list of effectful generators obtained by following only the effectful wire on the diagram. Formally, it is defined by structural induction over the premonoidal category of sessions as follows.

(1) It is the empty list for a pure morphism.
(2) It is invariant to whiskering.
(3) It contains a single element $[X^{\circ}]$ for each generator $(\circ)_X : I \to X$.
(4) It contains a single element $[X^{\bullet}]$ for each generator $(\bullet)_X : X \to I$.
(5) It concatenates the lists for a composition of morphisms.

It becomes straightforward to check that the sequence of events is well-defined. We write $\mathsf{Session}(A;B)[X_1^{\circ\!\!\!\bullet}, \dots, X_n^{\circ\!\!\!\bullet}]$ for the set of sessions $A \to B$ with a sequence of events $X_1^{\circ\!\!\!\bullet}, \dots, X_n^{\circ\!\!\!\bullet}$.

Proposition 5.5 (Combs are sessions)**.** *Sessions from A to B with a sequence of events $X_1^{\circ\!\!\!\bullet}, \dots, X_n^{\circ\!\!\!\bullet}$ are in bijective correspondence with combs with the same events happening sequentially,*

$$\mathsf{Session}(A;B)[X_1^{\circ\!\!\!\bullet}, \dots, X_n^{\circ\!\!\!\bullet}] \cong \mathsf{mLens}\left(X_1^{\circ\!\!\!\bullet} \triangleleft \dots \triangleleft X_n^{\circ\!\!\!\bullet}; \tfrac{A}{B}\right).$$

Proof. We proceed by structural induction over the presentation of the cteogry of sessions. The base case consists of a morphism that is pure: by definition, those are the combs $\mathsf{mLens}(\mathsf{N}; \frac{A}{B})$.

The inductive case considers a morphism $A \to B$ with at least one first occurrence of the effectful generators, $(\circ)_X$ or $(\bullet)_X$. We assume without loss of generality that this is $(\bullet)_X : I \to X$, so its sequence of events is $X^{\bullet}, \Gamma$ – the other case is analogous. We consider it as a string diagram for a monoidal category, quotiented by the equations of the base monoidal category $\mathbb{C}$, the symmetries and only up to interchange by isotopy. We may split the diagram into two parts (as in Figure 22); we leave everything before the generator $(\bullet)_X$ to one side, $f \in \mathbb{C}(A; X \otimes M)$, and everything after the generator to the other side $g \in \mathsf{Session}(M,B)[\Gamma]$. This procedure constructs the following comb,

$$\int^{M \in \mathbb{C}} \mathbb{C}(A; M \otimes X)_f \times \mathsf{Session}(A;B)[\Gamma]_g.$$

Of course, the usual problem with this kind of definitions that split a string diagram is that we need to prove that they are well-defined. We need to show that this definition is invariant to isotopy; we do so by cases: *(i)* if the isotopy interchanges two boxes before the cutting line, then there were two pure morphisms and it is captured by an equation of $\mathbb{C}$; *(ii)* if the isotopy interchanges two boxes after the cutting line, then it is defining an equation of sessions; *(iii)* if the

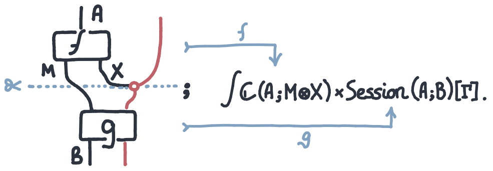


FIGURE 22. Splitting the diagram of a session.

isotopy interchanges two morphisms across the cutting line, then it is captured by dinaturality.

At the same time, note that these are exactly all of the equations imposed to combs: those of the original symmetric monoidal category and dinaturality; so the correspondence is bijective. Finally, by the induction hypothesis,

$$\begin{aligned}\mathsf{Session}(A;B)[X^\bullet,\Gamma] &\cong \int^{M\in\mathbb{C}} \mathbb{C}(A;X\otimes M)\times \mathsf{Session}(A;B)[\Gamma]\\ &\cong \int^{M\in\mathbb{C}} \mathbb{C}(A;X\otimes M)\times \mathsf{mLens}\left(\Gamma;\tfrac{A}{B}\right)\\ &\cong \mathsf{mLens}\left(X^\bullet \triangleleft \Gamma;\tfrac{A}{B}\right),\end{aligned}$$

which concludes the proof. □

The next step is to show that sessions over a process theory actually define a message theory. For this, we will only need sessions with no inputs or outputs: we write $\mathsf{Session}[\Gamma]$ for $\mathsf{Session}(I;I)[\Gamma]$.

Proposition 5.6. *Sessions over a strict symmetric monoidal category,* $\mathsf{Session}(\mathbb{C})$, *form a message theory. This construction extends to a functorial assignemnt*

$$\mathsf{Session}\colon \mathbf{SymMonCat}_{\mathsf{Str}} \to \mathsf{Msg}.$$

PROOF. We will show that sessions over a strict symmetric monoidal category, $\mathsf{Session}(\mathbb{C})$, form an algebra for the multicategory of polar shuffles. Consider a family of sessions, $s_i \in \mathsf{Session}(\Gamma_i)$; and consider a polar shuffle

$$p \in \mathsf{pShuf}(\Gamma_1,\dots,\Gamma_n;\Delta),$$

we need to construct a new session of type Δ.

The construction follows a topological intuition: consider the hypergraph defining the string diagrams of the sessions; and consider at the same time the acyclic graph defined by the polar shuffle. We glue the string diagram of each s_i, along its runtime wire, to the inputs and outputs, Γ_i, of the polar shuffle, removing these nodes on the process. The crucial step happens now: we have a graph containing nodes of the premonoidal category of sessions, and it has been constructed by gluing acyclic graphs along linear boundaries – it must be acyclic, and it must be a string diagram.

We define the composition of the sessions $s_i \in \mathsf{Session}(\Gamma_i)$ along the polar shuffle $p \in \mathsf{pShuf}(\Gamma_1, \dots, \Gamma_n; \Delta)$, to be the session represented by the string diagram here obtained. This assignment preserves the composition of polar shuffles – which is also defined topologically by gluing – and thus it determines an algebra.

Finally, the assignment is functorial with respect to the base monoidal category because *(i)* the construction of the sessions is functorial and *(ii)* we only apply operations of a symmetric premonoidal category when we build a string diagram over this premonoidal category of sessions. □

Example 5.7. Consider two morphisms, $f \colon A \to M \otimes X$ and $g \colon M \otimes Y \to B$, determining a session of type $[A^\circ, X^\bullet, Y^\circ, B^\bullet]$; and consider two other morphisms, $h \colon X \otimes U \to N$ and $k \colon N \to Y \otimes V$, determining a session of type $[X^\circ, U^\circ, V^\bullet, Y^\bullet]$. They can compose along the polar shuffle we defined in Figure 12; the result is in Figure 23.

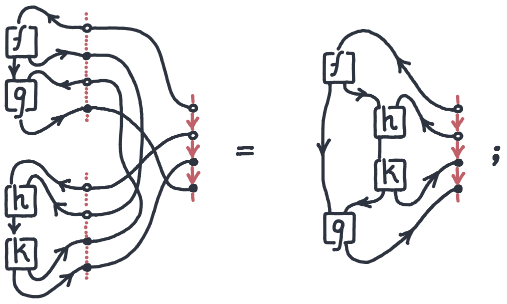


Figure 23. Two sessions compose along a polar shuffle.

5.3. Sessions versus Processes.

The final result of this section is to prove that *sessions* over a process theory are the free message theory over a symmetric monoidal category.

Lemma 5.8. *There exists a strict symmetric monoidal functor*

$$\mathsf{inProc} \colon \mathbb{C} \to \mathsf{Proc}(\mathsf{Session}(\mathbb{C}))$$

that includes a monoidal category in the processes of its message theory.

Proof. The functor will act as the identity on objects. We already know that combs are sessions (Proposition 5.5), and we can use this fact to construct the assignment on morphisms.

$$\mathsf{Proc}(\mathsf{Session}(\mathbb{C})(A; B)) = \mathsf{Session}(\mathbb{C})[A^\circ, B^\bullet] \cong \mathsf{mLens}(A^\circ, B^\bullet; {}^{I}_{I}) \cong \mathbb{C}(A; B).$$

It only remains to show that this assignment defines a strict symmetric monoidal functor: we need to show that it preserves composition, tensoring, identities and symmetries. This is straightforward, as we only need to check that the operations that we defined for the process theory of a message theory, $\mathsf{Proc}(\mathbb{M})$, correspond to the operations of a symmetric monoidal category. Let us explicitly check composition, $\mathsf{inProc}(f) ⨾ \mathsf{inProc}(g) = \mathsf{inProc}(f ⨾ g)$, in Figure 24, the rest follow a similar pattern.

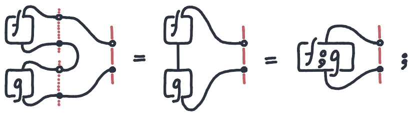


FIGURE 24. The inclusion of processes preserves composition.

Checking the rest of the cases concludes the construction. □

THEOREM 5.9. *Sessions and processes form an adjunction,* $\mathsf{Session} \dashv \mathsf{Proc}$*; where sessions,* $\mathsf{Session} \colon \mathbf{SymMonCat}_{\mathsf{Str}} \to \mathsf{Msg}$*, construct the free message theory over a symmetric monoidal category, and where processes,* $\mathsf{Proc} \colon \mathsf{Msg} \to \mathbf{SymMonCat}_{\mathsf{Str}}$*, construct the cofree symmetric monoidal category over a message theory.*

PROOF. Consider a strict symmetric monoidal category, $\mathbb{C}$, and a message theory, $\mathbb{M}$, endowed with a strict symmetric monoidal functor $F \colon \mathbb{C} \to \mathsf{Proc}(\mathbb{M})$. We will construct a message functor $F^\sharp \colon \mathsf{mLens}(\mathbb{C}) \to \mathbb{M}$ and prove that it is the unique one satisfying $\mathsf{inProc} ⨾ \mathsf{Proc}(F^\sharp) = F$.

Let us show that such a message functor, if it were to exist, would be unique. Firstly, the image on message types is already determined to be $F^\sharp(A) = F(A)$. Secondly, the image on sessions consisting on a single morphism, $\mathsf{inProc}(f) \colon [A^\circ, B^\bullet]$, is determined, $F^\sharp(\mathsf{inProc}(f)) = F(f) \colon [FA^\circ, FB^\bullet]$. We will show now that this reasoning can be extended to all sessions: we know that sessions of type $[X_1^{\bullet}, \dots, X_n^{\bullet}]$ are combs (Proposition 5.5) of type

$$(f_0 | \dots | f_n) \colon X_1^{\bullet} \triangleleft \dots \triangleleft X_n^{\bullet} \to \binom{I}{I}.$$

This determines their image: combs can be factored as the composition, in the message theory of multiple sessions consisting of a single morphism (Figure 25). Accordingly, their image, $F^\sharp(f_0 | \dots | f_n)$, should be the composition, on the message theory, of these pieces (Figure 25).

Let us now show that we have constructed a well-defined assignment. Our construction should preserve the dinaturality equivalence relation imposed to combs. This happens, indeed, and the proof simply checks that the images of two combs,

$$(f_0 ⨾ (\mathrm{id} \otimes h_0) | f_1 ⨾ (\mathrm{id} \otimes h_1) | \dots | f_n) = (f_0 | (\mathrm{id} \otimes h_0) ⨾ f_1 | \dots | (\mathrm{id} \otimes h_{n-1}) ⨾ f_n),$$

are equal (Figure 26). We have constructed a well-defined assignment on sessions.

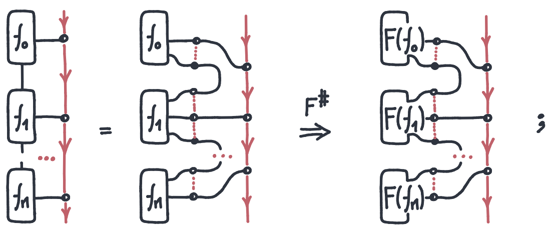


Figure 25. Image of a comb under the message functor.

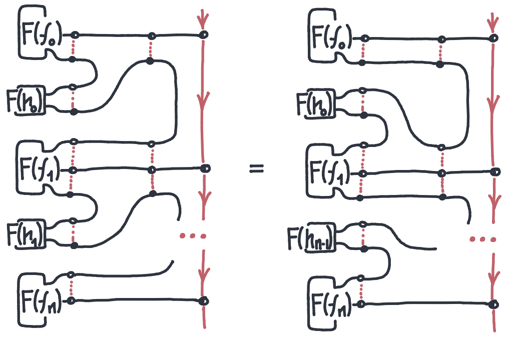


Figure 26. The assignment of combs to sessions preserves dinaturality.

Finally, we need to check that $F^\sharp$ is a message functor preserving all of the operations determined by polar shuffles. In the theory of combs, applying a polar shuffle corresponds to a rewiring into another comb; applying the polar shuffle in $\mathbb{M}$ must result in the same rewiring of the pieces forming the comb – which, as we have already shown, is precisely the image of that comb. This forces $F^\sharp$ to preserve the application of a polar shuffle.

We have shown that $F^\sharp$ is indeed a message functor and that it is the only possible one satisfying $\mathsf{inProc} \mathbin{;} \mathsf{Proc}(F^\sharp) = F$. □

5.4. Example: One-Time Pad, as a Message Session. Let us come back to Example 5.8, where we discussed a decomposition of the one-time pad. We now know that there is an adjunction between symmetric monoidal categories and message theories, let us use it to provide semantics to the decomposition of the one-time pad example.

The theory for the one-time pad problem can be expressed in a message theory $\mathbb{O}$ where we have a single object generator for the type of a message, X, and a single session generator for each one of the actors.

(1) $\mathsf{Stage}\colon X^\circ \triangleleft X^\bullet \triangleleft X^\bullet$,
(2) $\mathsf{Bob}\colon X^\bullet \triangleleft X^\circ \triangleleft X^\bullet$,
(3) $\mathsf{Alice}\colon X^\circ \triangleleft X^\circ \triangleleft X^\bullet$,
(4) $\mathsf{Eve}\colon X^\circ \triangleleft X^\bullet$.

We will interpret these generators in the free message theory over the category of finite sets and stochastic maps: thanks to the adjunction, we know that, if we could interpret each one of the components presenting the category of finite sets in any message theory, then we can interpret this whole example inside it.

We already gave an interpretation to each one of the components in terms of combs. We rewrite now the example explicitly separating each one of the parts that form it (Figure 27).

Proposition 5.10. *The session describing the one-time pad protocol is equal to a session where* Alice *and* Bob *communicate the message directly and* Eve *attacks a signal representing pure noise.*

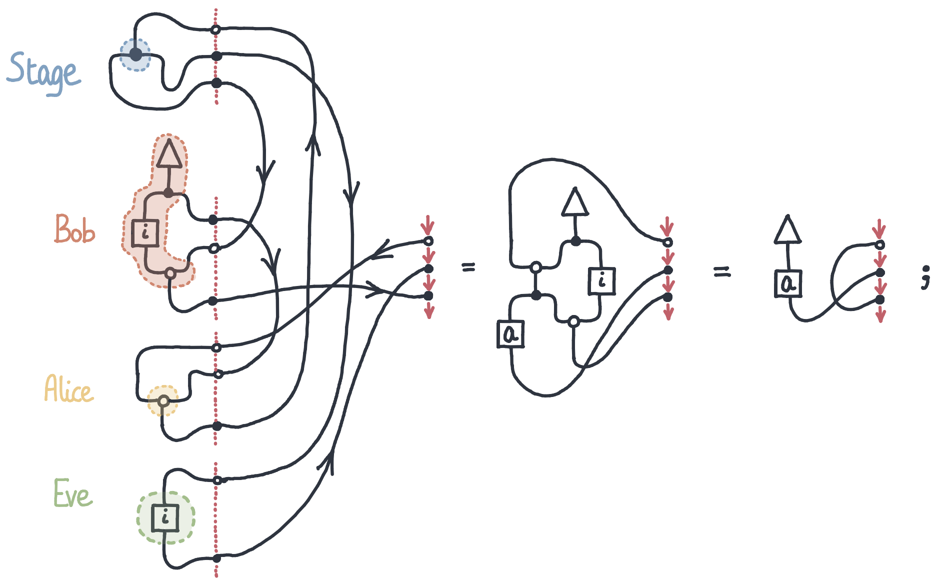


FIGURE 27. One-time pad, complete session.

Proof. Evaluating the session that describes the one-time pad example using the components described before, in Example 5.8, obtains the following polar shuffle applied to multiple combs. Evaluating the polar shuffle, as in Figure 27, produces the desired result. □

Remark 5.11. This discussion is not restricted to the modularity of the string diagrams: it affects the modularity of the code itself. Recall that we have a notation for sessions and polar shuffles; we use it in Figure 28 to write the one-time pad.

```
oneTimePad(?msg, !crypt, !decrypt) = {
  bob(!key, ?cryptBob, !decrypt),
  alice(!msg, ?key),
  eve(!cryptEve),
  stage(?crypt, !cryptBob, !cryptEve),
}
```

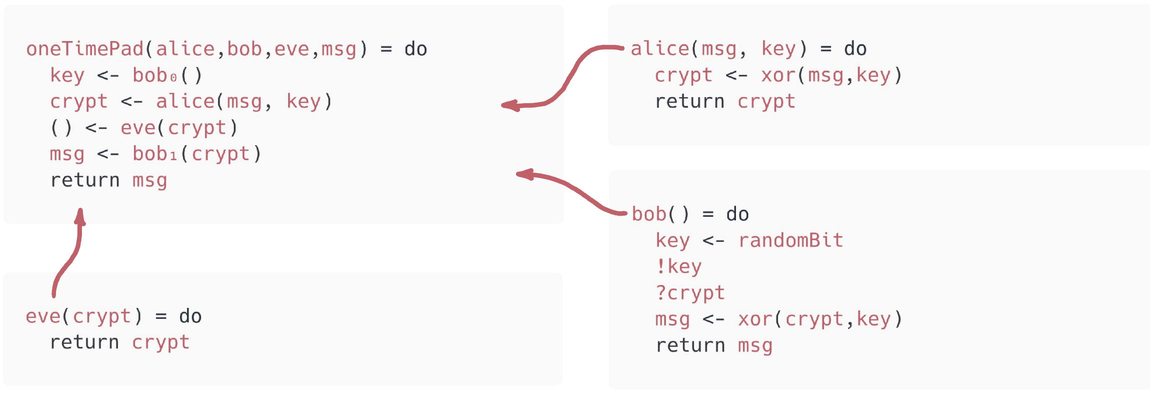
Figure 28. Notation for the one-time pad session.

At the same time, we have second a notation for sessions: sessions are ultimately morphisms of an effectful category, so we can use *do-notation* without the interchange axiom to represent them. The sending and receiving effects can be written as (!/?) respectively. Lenses are tuples of morphisms, and they can be represented in do-notation using that exact characterization. The following Figure 29 shows a modular implementation of the one-time pad that separates each one of the actors into a different module.

Which notation should we settle for? It seems that both interplay nicely together: the best way of writing a message session seems to be to write its underlying polar shuffle, as in Figure 28, while the best way of writing processes may be the usage of do-notation as in Figure 29, which is well-known and imposes a human-readable order on the operations.

5.5. Case Study: Causal versus Evidential Decision Theories. Leibniz's dream was to see philosophical debates reduced to mathematical calculation, to have a formal language for decision theory and an algorithm to solve any dispute.

> "[...] if controversies were to arise, there would be no more need of disputation between two philosophers than between two calculators. For it would suffice for them to take their pencils in their hands and to sit down at the abacus, and say



```
oneTimePad(alice,bob,eve,msg) = do
  key <- bob₀()
  crypt <- alice(msg, key)
  () <- eve(crypt)
  msg <- bob₁(crypt)
  return msg
```

```
alice(msg, key) = do
  crypt <- xor(msg,key)
  return crypt
```

```
bob() = do
  key <- randomBit
  !key
  ?crypt
  msg <- xor(crypt,key)
  return msg
```

```
eve(crypt) = do
  return crypt
```

FIGURE 29. Do-notation for the one-time pad.

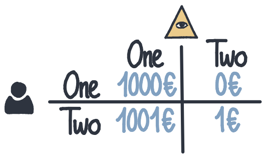
> to each other (and if they so wish also to a friend called to help): *calculemus* (let us calculate)."

However, our modern decision theory seems far from this dream. For instance, Monty Hall's problem caused famous controversies and confident blunders of some experts [vS] while being relatively simple to describe. It could seem that the passage from the statement to its formal encoding is more of an art than a science.

Let us try to understand one of these debates: *causal* versus *evidential* decision theory on Newcomb's problem [Noz69, Ahm14, YS17]. We will use message theories to set up the scene and partial Markov categories to compute the solution.

Definition 5.12. Newcomb's problem [Noz69] is a famous decision problem that sets apart Evidential and Causal Decision Theory. An agent (👤) is in front of two boxes: a transparent box filled with 1€ and an opaque box (▢▢). The agent is given the choice between taking both boxes (two-boxing, T) and taking just the opaque box (one-boxing, O). However, the opaque box is controlled by a "perfectly accurate" predictor (△). The predictor placed 1000€ in the opaque box if it predicted that the agent would one-box and left it empty otherwise. The agent knows this. Which action should the agent choose?

At the risk of oversimplifying, most philosophers are divided in two schools [Ahm14, YS17]. Those that follow *causal decision theory* would claim that no matter what the predictor does, the lower row of the table in Figure 30 contains strictly more utility; they prescribe *two-boxing*. Those that follow *evidential*



👤 \ ◬	One	Two
One	1000€	0€
Two	1001€	1€

FIGURE 30. Newcomb's problem: table of utilities.

decision theory claim that, because the predictor is omniscient, *one-boxing* is the only way of ensuring the biggest prize is on the box.

The analysis of the problem starts by dividing it into different parties: *(i)* the agent (👤) must only make a choice on whether to one-box or two-box; *(ii)* the stage (▢▢) takes the choice of the agent, the prediction of the predictor, broadcasts the choice of the agent and computes the final utility of the agent, and *(iii)* the predictor (◬) sends a prediction and, only afterwards, can see the choice of the agent. Let us call $X = \{\mathsf{O}, \mathsf{T}\}$ to the set containing *one-boxing* or *two-boxing*; we are claiming to have three elements of a message theory: the agent, (👤): $X^{\bullet}$; the predictor, (◬): $X^{\bullet} \triangleleft X^{\circ}$; and the stage, (▢▢): $X^{\circ} \triangleleft X^{\circ} \triangleleft X^{\bullet} \triangleleft X^{\bullet}$.

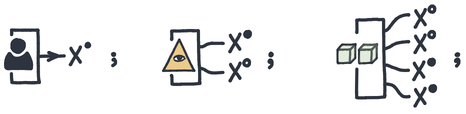


FIGURE 31. Newcomb's problem: components of a message theory.

The Evidential Decision Theory Solution: Let us take as an axiom that these components are constructed out of total stochastic channels; in other words, the message theory we use is the free message theory over the Kleisli category of the subdistribution monad, $\mathsf{Session}(\mathsf{Kleisli}(\mathbf{D}_{\leq 1}))$.

We cannot assume anything about the agent, but because of the construction of out free message theory (Definition 5.3), it must be given by a single stochastic channel (👤): $I \to X$. Even without assuming anything about the predictor, because of the construction of the free message theory (Definition 5.3), we know that must be constructed of two parts: the one that sends the prediction, $(◬)_1 \colon I \to M \otimes X$, and the one that receives the choice of the agent, $(◬)_2 \colon M \otimes X \to I$; of the first part we know nothing, but we have postulated that we will observe it to be perfectly accurate with the prediction, meaning that the second part will fail if it is not. Thus, we deduce it must factor as in Figure 32.

The wiring of the components is given by the statement: agent and predictor send choice and prediction to the stage, which answers giving back the choice to

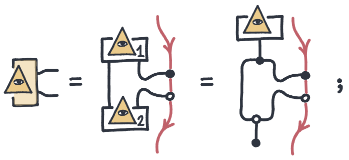


FIGURE 32. Evidential reading of the predictor.

the predictor and computing the output (Figure 35). We then reason *(i)* computing the polar shuffle; *(ii)* we analyze the agent by cases, the agent two-boxes (T) with probability a or one-boxes (O) with probability $(1 - a)$; *(iii)* because both cases are deterministic, they can be copied; *(iv)* we analyze then the predictor, it two-boxes (T) with probability p or one-boxes (O) with probability $(1-p)$; *(v)* we compute according to Figure 30, canceling the incompatible equality checks; and *vi* assuming that the first term is just an order of magnitude larger than the second, we can bound it by $p \cdot 1000$€, where we take $a = 1$: the agent should one-box.

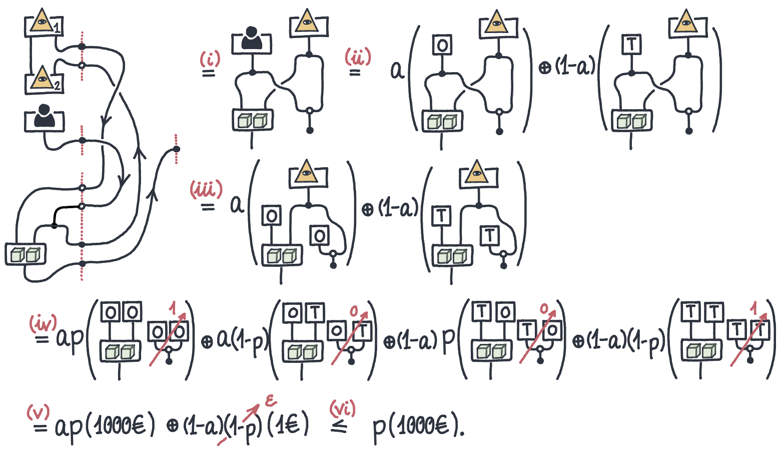


FIGURE 33. Newcomb's problem: the solution from Evidential Decision Theory.

The Causal Decision Theory Solution: Let us assume the same components (Figure 30). The only hypothesis over which we will place suspicion is that the predictor can be "perfectly accurate" without violating causality in some way. Causal decision theory assumes that all processes are causal, or total. That means that, after receiving the news of what the agent has chosen, the predictor can do nothing: there must exist a unique total morphism $X \otimes X \to I$.

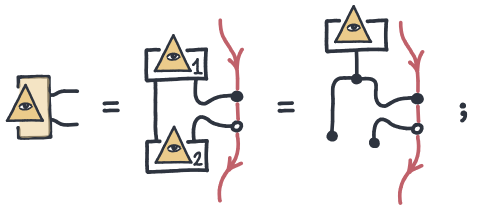


Figure 34. Causal reading of the predictor.

The wiring of the components is again the same, but the computation is now different: *(i)* we compute the polar shuffle; *(ii)* we analyze the agent by cases, the agent two-boxes (T) with probability a or one-boxes (O) with probability $(1-a)$; *(iii)* we analyze in the same way the predictor; *(iv)* we compute according to Figure 30, canceling the incompatible equality checks; and *(v)* we bound everything by the case where $a = 0$: this time, to maximize utility, the agent must two-box.

Was this formal analysis better than a pure discussion? We can now claim that the advantage is that the computations go from a starting diagram that represents our reading of the problem to a utility that we can maximize. We have turned most of the problem into a problem of computation: not only looking at the table of utilities (Figure 30) but at the whole statement of the problem. Of course, our formalization does not solve the debate on Newcomb's paradox, but at least it moves the controversy to a more fundamental point: are we fine with using the action of the agent to reason acausally about the predictor? The algebra of partial Markov categories provides a mathematical framework where it makes sense to assume so; the algebra of message theories takes care of the rest of the discussion.

Bibliography. The idea of using send/receive effects for encoding sessions is not new. Message passing can be also seen as a core component of game semantics, which has a vast literature [AJ94, AM99, Hyl97]. Game semantics has the ambition to provide the mathematical structures that describe coordination between distributed agents, starting from a duality between the Player's moves and

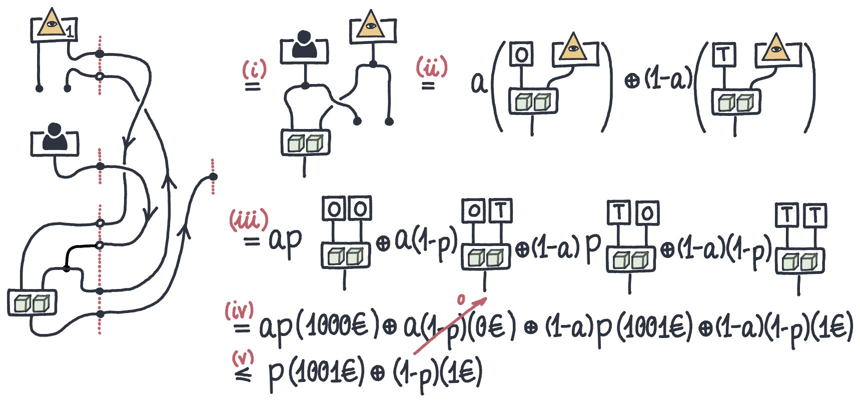

1
(i) =
(ii) =
a(O) ⊕ (1-a)(T)
(iii) = ap O O ⊕ a(1-p) O T ⊕ (1-a) p T O ⊕ (1-a)(1-p) T T
0
(iv) = ap(1000€) ⊕ a(1-p)(0€) ⊕ (1-a)p(1001€) ⊕ (1-a)(1-p)(1€)
(v) ≤ p(1001€) ⊕ (1-p)(1€)

FIGURE 35. Newcomb's problem: the solution from Causal Decision Theory.

the Opponent's moves; one of its achievements is to provide syntax-independent semantics for different extensions of PCF [McC00], including one with global state [AHM98].

Game semantics and session types [HYC08] have been called "two sides of the same coin" [CY19]. Orchard and Yoshida [OY16] discuss two mutual embeddings between an effectful λ-calculus (PCF) and a session π-calculus; further work also implemented the corresponding do-notation [OY17]. In our framework, this correspondence occurs between premonoidal categories and message theories; and thus between do-notation and polar shuffles.

Particularly relevant is Melliès' categorical approach to game semantics in the form of *template games* [Mel19]. The crucial difference between the present proposal and Melliès' line of work is that it starts from labelled transition systems as the basic notion. Melliès introduces *asynchronous graphs* – graphs with a set of commuting squares – and many of the same ingredients that we use here. Asynchronous graphs explain shuffles, polarization, and a failure of interchange in the form of a Grey tensor product [Mel21]. We would be interested to compare our approach to message passing with the framework of template games, and especially the points where they diverge: we land on normal duoidal categories while template games are based on Girard's linear logic [Gir89].

Finally, Newcomb's problem (or *paradox*) was first stated by Nozick [Noz69]. Evidential decision theory is defended in the work of Ahmad [Ahm14]; both Everitt, Leike and Hutter [ELH15], and Yudkowsky and Soares [YS17] have formalized comparisons of evidential decision theory, causal decision theory, and further variants.

CHAPTER 5

Conclusions and Further Work

Conclusions

Monoidal Context Theory. We have universally characterized the normal produoidal category of monoidal lenses (Theorem 5.3) as a free normalization of the cofree produoidal category on top of a monoidal category. The interpretation of this result is relevant: the splice-contour adjunction ([MZ22], Theorem 4.4) relates each category to its cofree promonoidal category of incomplete terms; in the same way, we have constructed a monoidal splice-contour adjunction relating each monoidal category to its cofree produoidal category of incomplete processes. The category underlying this universal produoidal category is familiar: it is the category of monoidal lenses.

Monoidal lenses have gained recent popularity in applications of category theory, apart from their classical counterparts in database theory [JRW12]: they have spawned applications in bidirectional transformations [FGM+07] but also in functional programming [PGW17, CEG+20], open games [GHWZ18], polynomial functors [NS22] and quantum combs [HC22]. Moreover, a different promonoidal structure for lenses had been already studied in the past by Pastro and Street [PS07]. Apart from lenses, incomplete processes have appeared implicitly multiple times in recent literature. Kissinger and Uijlen [KU17] describe higher-order quantum processes using contexts with holes in compact closed monoidal categories. Ghani, Hedges, Winschel and Zahn [GHWZ18] describe economic game theory in terms of *lenses* and incomplete processes in cartesian monoidal categories. Bonchi, Piedeleu, Sobociński and Zanasi [BPSZ19] study contextual equivalence in their monoidal category of affine signal flow graphs. Di Lavore, de Felice and Román [DLdFR22] define *monoidal streams* by iterating monoidal context coalgebraically. This situation prompted a question: why are lenses so prevalent? why do they appear in seemingly unrelated applications? We can now claim a conceptual answer with a mathematical justification. Lenses are the universal algebra for decomposing morphisms in process theories. The recent applications of lenses all describe incomplete processes.

Incomplete processes have two uses: on the one hand, they track the dependencies between monoidal processes; on the other hand, they allow us to split the multiple agents of a multi-party process. The former is quite useful in itself: duoidal categories provide two different tensors – a sequential, ($\triangleleft$), and a parallel one ($\otimes$) – that can track dependencies between different processes (as we saw in Section 1.2). Signalling and non-signalling conditions are important to the study of quantum theories, and recent work by Wilson and Chiribella [WC22] and Hefford and Kissinger [HK22] has studied signalling using structures close to monoidal lenses; we hope that the universal characterization of lenses as cofree produoidal categories may help extracting the exact structure needed for these physical frameworks. Even forgetting about dependencies explicitly, the theory of monoidal contexts allows us to pursue branches of computer science that were classically restricted to 1-dimensional syntaxes: the recent work of Eanrshaw and Sobocinski [ES22] studies monoidal regular languages as the natural monoidal analogue of the classical notion of regular language.

However, it is the second application of incomplete processes the one that this author found more surprising: we can now separate the different agents of a multi-party process in an arbitrary monoidal category. Message passing was not the main goal of this thesis, but developing it has brought interesting connections to *game semantics.*

Monoidal Message Passing. Symmetric monoidal categories have two operations: sequential composition ($⨟$) and parallel composition ($\otimes$). Naively, we would think then that there are two ways of decomposing monoidal processes: sequentially and in parallel. This is not false, and for many applications this may be the simpler way of dealing with this; after all, glueing sequentially and in parallel is the only thing we need to separate a string diagram in its atomic parts. However, this misses the rich algebra of incomplete morphisms and their compositions: monoidal lenses can be composed according to any polar shuffle, and that is the algebra that we are implicitly using when we cut a string diagram into pieces that do not necessarily follow the sequential and parallel divisions. We can now argue that the algebra of message passing for process theories is that of message theories. Message theories try to be a minimalistic axiomatization of what it means to communicate different processes that send and receive messages: we have argued for these axioms in Section 1, in a way that should appeal any reader not familiar with the categorical framework behind them. Their combinatorial characterization in terms of polar shuffles only makes them more concrete.

We have developed a theory of context on top of monoidal categories and we have used it to develop a canonical theory of message passing in monoidal categories. We have argued that this a fundamental structure for concurrency, and we have characterized it universally; however, we could still argue that it does not address the problem posed by Abramsky of finding the fundamental structures of concurrency [Abr05]. The main limitation of this framework is that it does not provide most of the features that we expect from fully-fledged session types: how can we model choice, or synchrony [Hon93, HYC08]? how can we model iteration, feedback, or other common programming constructs [McC00]? Minimalism, however, may be a good thing to separate these concerns from the fundamental structure: if we want to model choice, we can do so using *distributive categories* [CLW93], or *linear actegories* [CP09]; if we want feedback and iteration, we can recall traced categories [JSV96], categories with feedback [LGR+23], and notions of monoidal automata [DLdFR22]. Precisely because our framework is minimal, it seems that it is robust enough to support these additions; we will be interested in constructing models of game semantics in further work.

Future Work. Monoidal game semantics was not the original goal of this project but it became its most promising avenue; last section gives us a recipe to construct a *session do-notation* calculus on top of any monoidal category, which would include stochastic, non-deterministic and partial variants of a multi-party do-notation calculus. Levy, Power and Thielecke [LPT03] discuss the correspondence between premonoidal categories and call-by-value languages; we would be interested in extending this correspondence into message passing [OY17]. Once there, it seems plausible that we can connect this idea to the literature on game semantics for fully-fledged programming semantics [McC00]. Specifically, we would like to construct a model of probabilistic programming allowing for message passing, choice and iteration; multiple developments in categorical probability (such as quasiborel spaces [HKS+18] or Markov categories [Fri20]) make this possible.

The existence of a vast literature on message passing in terms of linear actions [CP09] makes it particularly important to understand exactly how duoidal categories and linear logic relate. We conjecture that physical duoidal categories also form *isomix* linearly distributive categories [CS97a], and this may be a categorical justification behind this connection.

For conciseness, we have not discussed approaches to iteration in monoidal categories. Joint work of this author with Di Lavore and de Felice [DLdFR22] has shown that it is possible to reason coinductively with automata in monoidal categories, using precisely monoidal lenses to describe incomplete processes. It seems plausible then, that mixing coinduction with message passing allows us to talk about networks of stochastic iterative processes: these have remarkable applications in scientific modelling, where we can write causal networks of stochastic processes in which unknown components are approximated stochastically.

Bibliography

[Abr05] Samson Abramsky. What are the fundamental structures of concurrency?: We still don't know! In Luca Aceto and Andrew D. Gordon, editors, *Proceedings of the Workshop "Essays on Algebraic Process Calculi", APC 25, Bertinoro, Italy, August 1-5, 2005*, volume 162 of *Electronic Notes in Theoretical Computer Science*, pages 37–41. Elsevier, 2005. 19, 174

[AC09] Samson Abramsky and Bob Coecke. Categorical quantum mechanics. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of Quantum Logic and Quantum Structures*, pages 261–323. Elsevier, Amsterdam, 2009. 28, 61

[AHLF18] Marcelo Aguiar, Mariana Haim, and Ignacio López Franco. Monads on higher monoidal categories. *Applied Categorical Structures*, 26(3):413–458, Jun 2018. 115

[AHM98] Samson Abramsky, Kohei Honda, and Guy McCusker. A fully abstract game semantics for general references. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No. 98CB36226)*, pages 334–344. IEEE, 1998. 171

[Ahm14] Arif Ahmed. *Evidence, Decision and Causality.* Cambridge University Press, 2014. 167, 171

[AHS02] Samson Abramsky, Esfandiar Haghverdi, and Philip J. Scott. Geometry of interaction and linear combinatory algebras. *Math. Struct. Comput. Sci.*, 12(5):625–665, 2002. 61

[AJ94] Samson Abramsky and Radha Jagadeesan. Games and full completeness for multiplicative linear logic. *The Journal of Symbolic Logic*, 59(2):543–574, 1994. 149, 170

[Alu21] Paolo Aluffi. *Algebra: Chapter 0*, volume 104. American Mathematical Society, 2021. 28

[AM] Marcelo Aguiar and Swapneel A. Mahajan. personal communication. 103

[AM99] Samson Abramsky and Guy McCusker. Game Semantics. In Ulrich Berger and Helmut Schwichtenberg, editors, *Computational Logic*, NATO ASI Series, pages 1–55, Berlin, Heidelberg, 1999. Springer.

156, 170

[AM10] Marcelo Aguiar and Swapneel A. Mahajan. *Monoidal functors, species and Hopf algebras*, volume 29. American Mathematical Society Providence, RI, 2010. 19, 98, 101, 103, 108

[BCST96] Robert F. Blute, Robin B. Cockett, Robert A. G. Seely, and Todd H. Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996. 61

[BD98] John C. Baez and James Dolan. Higher-dimensional algebra III. n-categories and the algebra of opetopes. *Advances in Mathematics*, 135(2):145–206, 1998. 142

[BE14] John C. Baez and Jason Erbele. Categories in control. *arXiv preprint arXiv:1405.6881*, 2014. 53

[Bec23] Jorge Becerra. Strictification and non-strictification of monoidal categories. *arXiv preprint arXiv:2303.16740*, 2023. 37

[Bén67] Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77, Berlin, Heidelberg, 1967. Springer Berlin Heidelberg. 41

[Bén68] Jean Bénabou. Structures algébriques dans les catégories. *Cahiers de topologie et géometrie différentielle*, 10(1):1–126, 1968. 53

[Bén00] Jean Bénabou. Distributors at work. *Lecture notes written by Thomas Streicher*, 11, 2000. 78, 79

[BG18] Guillaume Boisseau and Jeremy Gibbons. What you needa know about Yoneda: Profunctor optics and the Yoneda lemma (functional pearl). *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–27, 2018. 126

[BGK$^+$16] Filippo Bonchi, Fabio Gadducci, Aleks Kissinger, Paweł Sobociński, and Fabio Zanasi. Rewriting modulo symmetric monoidal structure. In *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 710–719, 2016. 42, 43, 45, 53

[BGMS21] John C. Baez, Fabrizio Genovese, Jade Master, and Michael Shulman. Categories of nets. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021. 42

[BK22] Anne Broadbent and Martti Karvonen. Categorical composable cryptography. In Patricia Bouyer and Lutz Schröder, editors, *Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings*, volume 13242 of *Lecture Notes in Computer Science*, pages 161–183. Springer, 2022.

17, 128, 129

[BNR22] Guillaume Boisseau, Chad Nester, and Mario Román. Cornering optics. In *ACT 2022*, volume abs/2205.00842, 2022. 25, 150

[BPS17] Filippo Bonchi, Dusko Pavlovic, and Pawel Sobocinski. Functorial semantics for relational theories. *CoRR*, abs/1711.08699, 2017. 18

[BPSZ19] Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Graphical affine algebra. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019. 173

[BR23] Dylan Braithwaite and Mario Román. Collages of string diagrams. *arXiv preprint arXiv:2305.02675*, 2023. 26, 83

[BS10] John C. Baez and Mike Stay. Physics, topology, logic and computation: A Rosetta stone. In *New Structures for Physics*, pages 95–172. Springer Berlin Heidelberg, 2010. 53

[BS13] Thomas Booker and Ross Street. Tannaka duality and convolution for duoidal categories. *Theory and Applications of Categories*, 28(6):166–205, 2013. 109

[BSS18] Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical conjunctive queries. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. 28, 60, 61

[BSZ14] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*, pages 435–450. Springer, 2014. 43

[BZ20] Nicolas Blanco and Noam Zeilberger. Bifibrations of polycategories and classical linear logic. In Patricia Johann, editor, *Proceedings of the 36th Conference on the Mathematical Foundations of Programming Semantics, MFPS 2020, Online, October 1, 2020*, volume 352 of *Electronic Notes in Theoretical Computer Science*, pages 29–52. Elsevier, 2020. 203

[Cam19] Alexander Campbell. How strict is strictification? *Journal of Pure and Applied Algebra*, 223(7):2948–2976, 2019. 36, 37, 39

[CEG$^+$20] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor optics, a categorical update. *CoRR*, abs/2001.07488, 2020. 24, 81, 124, 126, 131, 173

[CFS16] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Inf. Comput.*, 250:59–86, 2016. 28, 34, 127

[CGG⁺22] Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *European Symposium on Programming*, pages 1–28. Springer, Cham, 2022. 126

[CJ19] Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019. 28, 59, 60

[CLW93] Aurelio Carboni, Stephen Lack, and Robert F.C. Walters. Introduction to extensive and distributive categories. *Journal of Pure and Applied Algebra*, 84(2):145–158, 1993. 175

[CP09] Robin B. Cockett and Craig A. Pastro. The logic of message-passing. *Sci. Comput. Program.*, 74(8):498–533, 2009. 175

[CS97a] Robin B. Cockett and Robert A. G. Seely. Proof theory for full intuitionistic linear logic, bilinear logic, and mix categories. *Theory and Applications of categories*, 3(5):85–131, 1997. 175, 197

[CS97b] Robin B. Cockett and Robert A. G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997. 141, 203

[CS07] Robin B. Cockett and Robert A. G. Seely. Polarized category theory, modules, and game semantics. *Theory and Applications of Categories*, 18(2):4–101, 2007. 19, 149

[CS09] Geoffrey S. H. Cruttwell and Michael Shulman. A unified framework for generalized multicategories. *arXiv preprint arXiv:0907.2460*, 2009. 142

[CW87] Aurelio Carboni and Robert F. C. Walters. Cartesian bicategories I. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987. 34

[CY19] Simon Castellan and Nobuko Yoshida. Two sides of the same coin: session types and game semantics: a synchronous side and an asynchronous side. *Proceedings of the ACM on Programming Languages*, 3(POPL):1–29, 2019. 171

[Day70] Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, volume 137, pages 1–38, Berlin, Heidelberg, 1970. Springer Berlin Heidelberg. 113, 116, 122, 200, 202

[Dd09] Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009. 19

[DDR11] Jean-Guillaume Dumas, Dominique Duval, and Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. *Journal of*

Symbolic Computation, 46(3):272–293, 2011. 65

[DGNO10] Vladimir Drinfeld, Shlomo Gelaki, Dmitri Nikshych, and Victor Ostrik. On braided fusion categories I. *Selecta Mathematica*, 16(1):1–119, 2010. 67

[DLdFR22] 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*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. 173, 175

[DLLNS21] Ivan Di Liberti, Fosco Loregian, Chad Nester, and Paweł Sobociński. Functorial semantics for partial theories. *Proceedings of the ACM on Programming Languages*, 5(POPL):1–28, 2021. 18, 56

[DPS05] Brian Day, Elango Panchadcharam, and Ross Street. On centres and lax centres for promonoidal categories. In *Colloque International Charles Ehresmann*, volume 100, 2005. 110

[DS03] Brian Day and Ross Street. Quantum categories, star autonomy, and quantum groupoids, 2003. 200, 202, 203

[EH61] Beno Eckman and Peter Hilton. Structure maps in group theory. *Fundamenta Mathematicae*, 50(2):207–221, 1961. 108

[EHR23] Matt Earnshaw, James Hefford, and Mario Román. The produoidal algebra of process decomposition. *arXiv preprint arXiv:2301.11867*, 2023. 26, 93, 110, 113, 116, 202, 203

[ELH15] Tom Everitt, Jan Leike, and Marcus Hutter. Sequential extensions of causal and evidential decision theory. In *International Conference on Algorithmic Decision Theory*, pages 205–221. Springer, 2015. 171

[ES22] Matthew Earnshaw and Pawel Sobociński. Regular Monoidal Languages. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. 18, 116, 174

[FB94] David J Foulis and Mary K Bennett. Effect algebras and unsharp quantum logics. *Foundations of physics*, 24:1331–1352, 1994. 58, 60

[FGM$^+$07] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17–es, 2007. 130, 173

[FGP21] Tobias Fritz, Tomáš Gonda, and Paolo Perrone. De Finetti's theorem in categorical probability. *Journal of Stochastic Analysis*, 2(4), 2021.

60

[FJ19] Brendan Fong and Michael Johnson. Lenses and learners. *arXiv preprint arXiv:1903.03671*, 2019. 126

[Fon13] Brendan Fong. Causal Theories: A Categorical Perspective on Bayesian Networks. *Master's Thesis, University of Oxford. ArXiv preprint arXiv:1301.6201*, 2013. 60

[Fox76] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976. 54, 55, 60

[FP19] Tobias Fritz and Paolo Perrone. A probability monad as the colimit of spaces of finite samples. *Theory and Applications of Categories*, 34(7):170–220, 2019. 60

[FPR21] Tobias Fritz, Paolo Perrone, and Sharwin Rezagholi. Probability, valuations, hyperspace: Three monads on top and the support as a morphism. *Mathematical Structures in Computer Science*, 31(8):850–897, 2021. 60

[FR20] Tobias Fritz and Eigil Fjeldgren Rischel. Infinite products and zero-one laws in categorical probability. *Compositionality*, 2:3, 2020. 60

[Fri20] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. 28, 60, 175

[FS19] Brendan Fong and David I Spivak. Supplying bells and whistles in symmetric monoidal categories. *arXiv preprint arXiv:1908.02633*, 2019. 55

[FV20] Ignacio López Franco and Christina Vasilakopoulou. Duoidal categories, measuring comonoids and enrichment. *arXiv preprint arXiv:2005.01340*, 2020. 103

[GF16] Richard Garner and Ignacio López Franco. Commutativity. *Journal of Pure and Applied Algebra*, 220(5):1707–1751, 2016. 19, 97, 100, 101, 102, 103, 104, 105, 108, 115, 116, 121, 122

[GG09] Richard Garner and Nick Gurski. The low-dimensional structures formed by tricategories. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 146, pages 551–589. Cambridge University Press, 2009. 39

[GH99] Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In S. Doaitse Swierstra, editor, *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, 22-28 March, 1999, Proceedings*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999. 141

[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. 126, 131, 173

[Gir89] Jean-Yves Girard. Geometry of Interaction 1: Interpretation of System F. In R. Ferro, C. Bonotto, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, volume 127 of *Studies in Logic and the Foundations of Mathematics*, pages 221–260. Elsevier, 1989. 61, 171

[Gis88] Jay L. Gischer. The equational theory of pomsets. *Theoretical Computer Science*, 61(2-3):199–224, 1988. 108

[Gra81] Jan Grabowski. On partial languages. *Fundamenta Informaticae*, 4(2):427–498, 1981. 23, 106, 108

[Gro85] Alexandre Grothendieck. Récoltes et semailles: réflexions et témoignage sur un passé de mathématicien. *Grothendieck Circle Page*, 1985. 16

[Gui80] René Guitart. Tenseurs et machines. *Cahiers de topologie et géométrie différentielle catégoriques*, 21(1):5–62, 1980. 62

[Had18] Amar Hadzihasanovic. Weak units, universal cells, and coherence via universality for bicategories. *arXiv preprint arXiv:1803.06086*, 2018. 37

[Has97] Masahito Hasegawa. *Models of sharing graphs: a categorical semantics of let and letrec.* PhD thesis, University of Edinburgh, UK, 1997. 141

[HC22] James Hefford and Cole Comfort. Coend optics for quantum combs. *arXiv preprint arXiv:2205.09027*, 2022. 131, 173

[Hed19] Jules Hedges. Folklore: Monoidal kleisli categories, Apr 2019. 62

[Her00] Claudio Hermida. Representable multicategories. *Advances in Mathematics*, 151(2):164–225, 2000. 90

[Her01] Claudio Hermida. From coherent structures to universal properties. *Journal of Pure and Applied Algebra*, 165(1):7–61, 2001. 37

[HJ06] Chris Heunen and Bart Jacobs. Arrows, like monads, are monoids. In Stephen D. Brookes and Michael W. Mislove, editors, *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, volume 158 of *Electronic Notes in Theoretical Computer Science*, pages 219–236. Elsevier, 2006. 21, 53, 82

[HJW$^+$92] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S.

Nikhil, Will Partain, and John Peterson. Report on the Programming Language Haskell, A Non-strict, Purely Functional Language. *ACM SIGPLAN Notices*, 27(5):1, 1992. 21, 53

[HK22] James Hefford and Aleks Kissinger. On the pre- and promonoidal structure of spacetime. *arXiv preprint arXiv.2206.09678*, 2022. 105, 174

[HKS$^+$18] Chris Heunen, Ohad Kammar, Sam Staton, Sean Moss, Matthijs Vákár, Adam Ścibior, and Hongseok Yang. The semantic structure of quasi-borel spaces. In *PPS Workshop on Probabilistic Programming Semantics*, 2018. 175

[HMH14] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014. 61

[Hon93] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993. 19, 141, 175

[HR23] James Hefford and Mario Román. Optics for premonoidal categories. *CoRR*, abs/2305.02906, 2023. 25

[HS23] Chris Heunen and Jesse Sigal. Duoidally enriched Freyd categories. In *International Conference on Relational and Algebraic Methods in Computer Science*, pages 241–257. Springer, 2023. 105

[Huf54] David A. Huffman. The synthesis of sequential switching circuits. *Journal of the Franklin Institute*, 257(3):161–190, 1954. 21

[Hug00] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. 21, 27, 42, 45, 53, 61, 82

[Hug12] Dominic Hughes. Simple free star-autonomous categories and full coherence. *Journal of Pure and Applied Algebra*, 216(11):2386–2410, 2012. 156

[HV19] Chris Heunen and Jamie Vicary. *Categories for Quantum Theory: An Introduction.* Oxford University Press, 2019. 28

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284.

ACM, 2008. 141, 171, 175

[Hyl97] Martin Hyland. Game semantics. *Semantics and logics of computation*, 14:131, 1997. 170

[Jac15] Bart Jacobs. New directions in categorical logic, for classical, probabilistic and quantum logic. *Logical Methods in Computer Science*, 11, 2015. 58, 60

[Jef97a] Alan Jeffrey. Premonoidal categories and a graphical view of programs. *Preprint at ResearchGate*, 1997. 21, 74

[Jef97b] Alan Jeffrey. Premonoidal categories and flow graphs. *Electronical Notes in Theoretical Computer Science*, 10:51, 1997. 64

[JHH09] Bart Jacobs, Chris Heunen, and Ichiro Hasuo. Categorical semantics for arrows. *J. Funct. Program.*, 19(3-4):403–438, 2009. 61, 82

[JKZ21] Bart Jacobs, Aleks Kissinger, and Fabio Zanasi. Causal inference via string diagram surgery: A diagrammatic approach to interventions and counterfactuals. *Mathematical Structures in Computer Science*, 31(5):553–574, 2021. 60

[JRW12] Michael Johnson, Robert Rosebrugh, and Richard J. Wood. Lenses, fibrations and universal translations. *Mathematical structures in computer science*, 22(1):25–42, 2012. 130, 173

[JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. 30, 31, 33, 34, 36, 37, 53, 61

[JS93] André Joyal and Ross Street. Braided tensor categories. *Advances in Mathematics*, 102(1):20–78, 1993. 33

[JSV96] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447 – 468, 04 1996. 175

[JZ20] Bart Jacobs and Fabio Zanasi. The logical essentials of bayesian reasoning. *Foundations of Probabilistic Programming*, pages 295–331, 2020. 60

[KL80] Gregory Kelly and Miguel Laplaza. Coherence for compact closed categories. *Journal of pure and applied algebra*, 19:193–213, 1980. 103

[Kme12] Edward Kmett. lens library, version 4.16. *Hackage https://hackage. haskell. org/package/lens-4.16*, 2018, 2012. 126

[KPT96] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, pages 358–371. ACM Press,

1996. 141

[KSW97] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997. 34

[KU17] Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017. 173

[Lam69] Joachim Lambek. Deductive systems and categories II: standard constructions and closed categories. *Category Theory, Homology Theory and their Applications I*, 1969. 18, 90

[Lam86] Joachim Lambek. Cartesian closed categories and typed λ-calculi. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, Lecture Notes in Computer Science, pages 136–175, Berlin, Heidelberg, 1986. Springer. 18

[Lau05] Aaron D. Lauda. Frobenius algebras and ambidextrous adjunctions. *arXiv preprint math/0502550*, 2005. 200

[Law63] F. William Lawvere. Functorial semantics of algebraic theories. *Proceedings of the National Academy of Sciences*, 50(5):869–872, 1963. 18

[Lei04] Tom Leinster. *Higher Operads, Higher Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2004. 107, 108

[Lev22] Paul Blain Levy. Call-by-push-value. *ACM SIGLOG News*, 9(2):7–29, may 2022. 63

[Lew06] Geoffrey Lewis. Coherence for a closed functor. In *Coherence in categories*, pages 148–195. Springer, 2006. 103

[LGR$^+$21] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Pawel Sobocinski. A canonical algebra of open transition systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2021. 25

[LGR$^+$23] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Pawel Sobocinski. Span(Graph): a canonical feedback algebra of open transition systems. *Softw. Syst. Model.*, 22(2):495–520, 2023. 25, 175

[Lor21] Fosco Loregiàn. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. 79, 80, 82, 83

[LPT03] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and computation*, 185(2):182–210, 2003. 175

[LR23] Elena Di Lavore and Mario Román. Evidential decision theory via partial markov categories. In *LICS*, pages 1–14, 2023. 26, 56, 57, 59, 60

[LS09] F. William Lawvere and Stephen H. Schanuel. *Conceptual mathematics: a first introduction to categories.* Cambridge University Press, 2009. 16

[Mac63] Saunders MacLane. Natural associativity and commutativity. *Rice Institute Pamphlet-Rice University Studies*, 49(4), 1963. 33, 37, 52

[Mac78] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1978. 33

[Mar14] Daniel Marsden. Category theory using string diagrams. *arXiv preprint arXiv:1401.7220*, 2014. 40, 41

[McC00] Guy McCusker. Games and full abstraction for FPC. *Information and Computation*, 160(1-2):1–61, 2000. 171, 175

[Mel19] Paul-André Melliès. Template games and differential linear logic. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–13. IEEE, 2019. 171

[Mel21] Paul-André Melliès. Asynchronous template games and the gray tensor product of 2-categories. In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2021. 19, 171

[ML71] Saunders Mac Lane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics.* Springer Verlag, 1971. 28, 35, 36, 52, 80, 83

[MM90] José Meseguer and Ugo Montanari. Petri nets are monoids. *Inf. Comput.*, 88(2):105–155, 1990. 42

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. 21, 61, 82

[MP21] Cary Malkiewich and Kate Ponto. Coherence for bicategories, lax functors, and shadows, 2021. 103

[MS14] Rasmus Ejlers Møgelberg and Sam Staton. Linear usage of state. *Log. Methods Comput. Sci.*, 10(1), 2014. 74

[MS18] Paul-André Melliès and Léo Stefanesco. A game semantics of concurrent separation logic. *Electronic Notes in Theoretical Computer Science*, 336:241–256, 2018. 156

[Mye16] David Jaz Myers. String diagrams for double categories and equipments, 2016. 150

[MZ22] Paul-André Melliès and Noam Zeilberger. Parsing as a Lifting Problem and the Chomsky-Schützenberger Representation Theorem. In *MFPS 2022-38th conference on Mathematical Foundations for Programming Semantics*, 2022. 18, 77, 91, 92, 93, 116, 173

[Nes21] Chad Nester. The structure of concurrent process histories. In Ferruccio Damiani and Ornela Dardha, editors, *Coordination Models and Languages - 23rd IFIP WG 6.1 International Conference, COORDINATION 2021, Held as Part of the 16th International Federated Conference on Distributed Computing Techniques, DisCoTec 2021, Valletta, Malta, June 14-18, 2021, Proceedings*, volume 12717 of *Lecture Notes in Computer Science*, pages 209–224. Springer, 2021. 19, 149

[Noz69] Robert Nozick. Newcomb's Problem and Two Principles of Choice. In *Essays in honor of Carl G. Hempel*, pages 114–146. Springer, 1969. 167, 171

[NS22] Nelson Niu and David I. Spivak. Polynomial functors: A general theory of interaction. *In preparation*, 2022. 131, 173

[NV23] Chad Nester and Niels F. W. Voorneveld. Protocol choice and iteration for the free cornering. *CoRR*, abs/2305.16899, 2023. 149

[Ord20] Toby Ord. *The precipice: Existential risk and the future of humanity.* Hachette Books, 2020. 16

[OY16] Dominic Orchard and Nobuko Yoshida. Effects as sessions, sessions as effects. *ACM SIGPLAN Notices*, 51(1):568–581, 2016. 20, 171

[OY17] Dominic Orchard and Nobuko Yoshida. Session types with linearity in Haskell. In Simon Gay and António Ravara, editors, *Behavioural Types: from Theory to Tools*, River Publishers Series in Automation, Control and Robotics. River Publishers, 2017. 171, 175

[Pat01] Ross Paterson. A new notation for arrows. In Benjamin C. Pierce, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, pages 229–240. ACM, 2001. 45, 82

[Pat03] Ross Paterson. Arrows and computation. *The Fun of Programming*, pages 201–222, 2003. 53

[Pav13] Dusko Pavlovic. Monoidal computer I: basic computability by string diagrams. *Inf. Comput.*, 226:94–116, 2013. 61

[PC07] Jorge Picado and Maria Manuel Clementino. *An Interview with William F. Lawvere*. Online, `https://www.mat.uc.pt/~picado/lawvere/interview.pdf`., 2007. 16

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *Art Sci. Eng. Program.*, 1(2):7, 2017. 124, 126, 130, 173

[Pow02] John Power. Premonoidal categories as categories with algebraic structure. *Theor. Comput. Sci.*, 278(1-2):303–321, 2002. 21, 61

[PR84] Roger Penrose and Wolfgang Rindler. *Spinors and Spacetime.* Cited by Aleks Kissinger at the Categories mailing list. Cambridge University Press, 1984. 17

[PR97] John Power and Edmund Robinson. Premonoidal categories and notions of computation. *Math. Struct. Comput. Sci.*, 7(5):453–468, 1997. 62

[PS93] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 376–385. IEEE Computer Society, 1993. 141

[PS07] Craig Pastro and Ross Street. Doubles for Monoidal Categories. *arXiv preprint arXiv:0711.1859*, 2007. 116, 130, 173

[PSV21] Evan Patterson, David I. Spivak, and Dmitry Vagner. Wiring diagrams as normal forms for computing in symmetric monoidal categories. *Electronic Proceedings in Theoretical Computer Science*, page 49–64, Feb 2021. 131

[PT99] John Power and Hayo Thielecke. Closed Freyd- and kappa-categories. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 625–634. Springer, 1999. 61, 62

[Ril18] Mitchell Riley. Categories of Optics. *arXiv preprint arXiv:1809.00738*, 2018. 124, 126, 128, 130

[Rom20a] Mario Román. Comb Diagrams for Discrete-Time Feedback. *CoRR*, abs/2003.06214, 2020. 131

[Rom20b] Mario Román. Open diagrams via coend calculus. In David I. Spivak and Jamie Vicary, editors, *Proceedings of the 3rd Annual International Applied Category Theory Conference 2020, ACT 2020, Cambridge, USA, 6-10th July 2020*, volume 333 of *EPTCS*, pages 65–78, 2020. 24, 83, 125, 127, 131

[Rom22] Mario Román. Promonads and string diagrams for effectful categories. In Jade Master and Martha Lewis, editors, *Proceedings Fifth International Conference on Applied Category Theory, ACT 2022, Glasgow, United Kingdom, 18-22 July 2022*, volume 380 of *EPTCS*, pages 344–361, 2022. 24, 25, 65

[Sel10] Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. 53

[Shu16] Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. 29, 31, 34, 47, 53, 86, 142, 145

[Shu17] Michael Shulman. Duoidal category (nlab entry), section 2., 2017. https://ncatlab.org/nlab/show/duoidal+category, Last accessed on 2022-12-14. 115, 142, 143, 145

[Shu18] Michael Shulman. The 2-Chu-Dialectica construction and the polycategory of multivariable adjunctions. *arXiv preprint arXiv:1806.06082*, 2018. 67, 197

[SL13] Sam Staton and Paul Blain Levy. Universal properties of impure programming languages. In Roberto Giacobazzi and Radhia Cousot, editors, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, pages 179–192. ACM, 2013. 61, 63

[Sob10] Paweł Sobociński. Representations of Petri net interactions. In *International Conference on Concurrency Theory*, pages 554–568. Springer, 2010. 34

[Sob13] Pawel Sobociński. Graphical linear algebra. Online, personal blog, https://graphicallinearalgebra.net, 2013. 30, 34

[Spi13] David I. Spivak. The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits. *CoRR*, abs/1305.0297, 2013. 107, 131, 142

[SS22] Brandon T. Shapiro and David I. Spivak. Duoidal structures for compositional dependence. *arXiv preprint arXiv:2210.01962*, 2022. 19, 97, 98, 101, 104, 105, 107, 108

[SSV20] Patrick Schultz, David I. Spivak, and Christina Vasilakopoulou. Dynamical systems and sheaves. *Applied Categorical Structures*, 28(1):1–57, 2020. 131

[Str04] Ross Street. Frobenius monads and pseudomonoids. *Journal of mathematical physics*, 45(10):3930–3948, 2004. 203

[Str12] Ross Street. Monoidal categories in, and linking, geometry and algebra. *Bulletin of the Belgian Mathematical Society-Simon Stevin*, 19(5):769–820, 2012. 18, 19, 102, 108, 113, 116

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes.* Cambridge University Press, 2001. 141

[Sza75] Manfred E. Szabo. Polycategories. *Communications in Algebra*, 3(8):663–689, 1975. 196, 203

[Tod10] Todd Trimble. Coherence theorem for monoidal categories (nlab entry), section 3. discussion, 2010. https://ncatlab.org/nlab/show/coherence+theorem+for+monoidal+categories, Last accessed on 2022-05-10. 67

[UV08] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. In Jiří Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 263–284. Elsevier, 2008. 61

[UVZ18] Tarmo Uustalu, Niccolò Veltri, and Noam Zeilberger. The sequent calculus of skew monoidal categories. *Electronic Notes in Theoretical Computer Science*, 341:345–370, 2018. 18

[VC22] André Videla and Matteo Capucci. Lenses for composable servers. *CoRR*, abs/2203.15633, 2022. 131

[vdW21] John van de Wetering. A categorical construction of the real unit interval. *arXiv preprint arXiv:2106.10094*, 2021. 58, 60

[vN58] John von Neumann. *The Computer and the Brain.* Yale University Press, quoted by David Darlymple in "A New Type of Mathematics" (2018), a Transcript from a talk at Montreal, 1958. 16

[vS] Marilyn vos Savant. Parade 16: Ask Marilyn (Archived). `https://web.archive.org/web/20130121183432/http://marilynvossavant.com/game-show-problem/`. Accessed: 2013-01-21. 167

[WC22] Matt Wilson and Giulio Chiribella. A mathematical framework for transformations of physical processes. *arXiv preprint arXiv:2204.04319*, 2022. 174

[YS17] Eliezer Yudkowsky and Nate Soares. Functional Decision Theory: a New Theory of Instrumental Rationality. *ArXiv preprint arXiv:1710.05060*, 2017. 167, 171

APPENDIX A

Supplementary material

1. Coherence diagrams for a duoidal category

$$\begin{tikzcd}
((A \lhd B) \otimes (C \lhd D)) \otimes (E \lhd F) \arrow[r, "\alpha"] \arrow[d, "\psi_2 \otimes id"'] & (A \lhd B) \otimes ((C \lhd D) \otimes (E \lhd F)) \arrow[d, "id \otimes \psi_2"] \\
((A \otimes C) \lhd (B \otimes D)) \otimes (E \lhd F) \arrow[d, "\psi_2"'] & (A \lhd B) \otimes ((C \otimes E) \lhd (D \otimes F)) \arrow[d, "\psi_2"] \\
((A \otimes C) \otimes E) \lhd ((B \otimes D) \otimes F) \arrow[r, "\alpha \lhd \alpha"] & (A \otimes (C \otimes E)) \lhd (B \otimes (D \otimes F))
\end{tikzcd}$$

$$\begin{tikzcd}
((A \lhd B) \lhd C) \otimes ((D \lhd E) \lhd F) \arrow[r, "\beta \otimes \beta"] \arrow[d, "\psi_2"'] & (A \lhd (B \lhd C)) \otimes (D \lhd (E \lhd F)) \arrow[d, "\psi_2"] \\
((A \lhd B) \otimes (D \lhd E)) \lhd (C \otimes F) \arrow[d, "\psi_2 \otimes id"'] & (A \otimes D) \lhd ((B \lhd C) \otimes (E \lhd F)) \arrow[d, "id \otimes \psi_2"] \\
((A \otimes D) \lhd (B \otimes E)) \lhd (C \otimes F) \arrow[r, "\beta"] & (A \otimes D) \lhd ((B \otimes E) \lhd (C \otimes F))
\end{tikzcd}$$

FIGURE 1. Coherence diagrams for associativity of a duoidal category.

$$\begin{tikzcd}
I \otimes (A \lhd B) \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\lambda"'] & (I \lhd I) \otimes (A \lhd B) \arrow[d, "\psi_2"] & (A \lhd B) \otimes I \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\rho"'] & (A \lhd B) \otimes (I \lhd I) \arrow[d, "\psi_2"] \\
A \lhd B & (I \otimes A) \lhd (I \otimes B) \arrow[l, "\lambda \lhd \lambda"] & A \lhd B & (A \otimes I) \lhd (B \otimes I) \arrow[l, "\rho \lhd \rho"]
\end{tikzcd}$$

FIGURE 2. Coherence diagrams for $\otimes$-unitality of a duoidal category.

$$\begin{tikzcd}
N \lhd (A \otimes B) \arrow[d, "\kappa"'] & (N \otimes N) \lhd (A \otimes B) \arrow[l, "\varphi_2 \lhd id"'] \arrow[d, "\psi_2"] \\
A \otimes B & (N \lhd A) \otimes (N \lhd B) \arrow[l, "\kappa \otimes \kappa"] \\
(A \otimes B) \lhd N \arrow[d, "\nu"'] & (A \otimes B) \lhd (N \otimes N) \arrow[l, "id \lhd \varphi_2"'] \arrow[d, "\psi_2"] \\
A \otimes B & (A \lhd N) \otimes (B \lhd N) \arrow[l, "\nu \otimes \nu"]
\end{tikzcd}$$

FIGURE 3. Coherence diagrams for $\lhd$-unitality of a duoidal category.

$$\begin{tikzcd}
(N \otimes N) \otimes N \arrow[rr, "\alpha"] \arrow[d, "\varphi_2 \otimes id"'] & & N \otimes (N \otimes N) \arrow[d, "id \otimes \varphi_2"] \\
N \otimes N \arrow[r, "\varphi_2"'] & N & N \otimes N \arrow[l, "\varphi_2"]
\end{tikzcd}$$

$$\begin{tikzcd}
I \lhd I \arrow[d, "\psi_0 \otimes id"'] & I \arrow[l, "\psi_0"'] \arrow[r, "\psi_0"] & I \lhd I \arrow[d, "id \otimes \psi_0"] \\
(I \lhd I) \lhd I \arrow[rr, "\beta"'] & & I \lhd (I \lhd I)
\end{tikzcd}$$

FIGURE 4. Associativity and coassociativity for N and I in a duoidal category.

$$\begin{tikzcd}
N \otimes I \arrow[r, "\rho"] \arrow[d, "id \otimes \varphi_0"'] & N \\
N \otimes N \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
I \otimes N \arrow[r, "\lambda"] \arrow[d, "\varphi_0 \otimes id"'] & N \\
N \otimes N \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
I \lhd N \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\nu"'] & I \lhd I \\
I \arrow[ur, "\psi_0"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
N \lhd I \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\kappa"'] & I \lhd I \\
I \arrow[ur, "\psi_0"'] &
\end{tikzcd}$$

FIGURE 5. Unitality and counitality for N and I in a duoidal category.

2. Polycategories

This extra section repeats the splice-contour adjunction for polycategories. It is a detour from the main text; and it is not necessary for its development: this does not seem to be the direction we want to follow to study context in categories or monoidal categories. It is, however, another proof of the resilience of the splice-contour adjunction: the duality between a category and its opposite induces a pseudofrobenius algebra on the monoidal bicategory of profunctors.

2.1. Polycategories. A polycategory is like a category where every morphism has both a list of inputs and a list of outputs [Sza75]. This does not mean that its inputs and outpuss start forming a monoid, as in strict monoidal categories; morphisms really have different multiple inputs and outputs, and we need to choose a single one to compose along it. A polycategory, $\mathbb{P}$, contains a set of objects, $\mathbb{P}_{obj}$, as categories and multicategories; but instead of a set of morphisms, it will have a set of *polymorphisms*,

$$\mathbb{P}(X_1, \dots, X_n; Y_1, \dots, Y_m),$$

for each two lists of objects $X_1, \dots, X_n, Y_0, \dots, Y_m \in \mathbb{P}_{obj}$. As in linear logic, we denote both sides of a derivation by two metavariables, $\Gamma = X_1, \dots, X_n$ and $\Delta = Y_0, \dots, Y_m$, and write $\mathbb{P}(\Gamma; \Delta)$ for the set of polymorphisms.

Definition 2.1. A *polycategory* $\mathbb{P}$ is a collection of objects, $\mathbb{P}_{obj}$, together with a collection of *polymorphisms*, $\mathbb{P}(X_0, \dots, X_n; Y_0, \dots, Y_m)$, for each two lists of objects $X_0, \dots, X_n \in \mathbb{P}$ and $Y_0, \dots, Y_m \in \mathbb{P}$. For each object $X \in \mathbb{P}_{obj}$, there must be an identity, $\mathrm{id}_X \in \mathbb{P}_{obj}(X; X)$; and for each pair of composable maps,

$$f \in \mathbb{P}(\Gamma; \Delta_1, X, \Delta_2) \text{ and } g \in \mathbb{P}(\Gamma_1, X, \Gamma_2; \Delta),$$
$$\text{where either } \Delta_1 \text{ or } \Gamma_1, \text{ and either } \Delta_2 \text{ or } \Gamma_2, \text{ are empty,}$$

there must be a composite polymorphism $f \mathbin{⨟_X} g \in \mathbb{P}(\Gamma_1, \Gamma, \Gamma_2; \Delta_1, \Delta, \Delta_2)$. This means that there are four possible types of composition (Figure 6), and they yield the same polymorphism whenever they overlap.

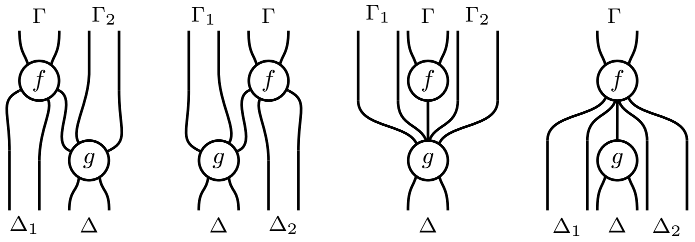


FIGURE 6. Four planar polycategorical compositions.

Moreover, polycategories must satisfy the following two unitality axioms, $f \fatsemi_X \mathrm{id}_X = f$ and $\mathrm{id}_X \fatsemi_X f = f$; two associativity axioms, $f \fatsemi_X (g \fatsemi_Y h) = (f \fatsemi_X g) \fatsemi_Y h$ and $f \fatsemi_X (g \fatsemi_Y h) = g \fatsemi_Y (f \fatsemi_X h)$; and an interchange axiom, $(f \fatsemi_A g) \fatsemi_B h = (f \fatsemi_B h) \fatsemi_A g$; whenever any of these is formally well-typed.

Remark 2.2. Asking for an identity on each object, $\mathrm{id}_X \in \mathbb{P}(X; X)$, is different from asking for an identity on each list of objects,

$$\mathrm{id}_{X_0,\dots,X_n} \in \mathbb{P}(X_0, \dots, X_n; X_0, \dots, X_n).$$

The latter gives rise to *isomix* categories [CS97a] and we will not discuss it here.

Example 2.3. A *polyfunctional relation*, $R \in \mathbf{MultiFun}(A_1, \dots, A_n; B_1, \dots, B_m)$, is a relation $R \colon A_1 \times \dots \times A_n \to B_1 \times \dots \times B_m$ together with representing functions that, given an element of the relation missing exactly one element, return the element missing. Explicitly, there exist two families of functions,

$$f_j \colon A_1 \times \dots \times A_n \times B_0 \times \overset{\cancel{B_j}}{\dots} \times B_m \to B_j \quad \text{and}$$
$$g_i \colon A_1 \times \overset{\cancel{A_i}}{\dots} \times A_n \times B_0 \times \dots \times B_m \to A_i,$$

such that $R(a_1, \dots, b_m)$ if and only if $f_j(a_1, \dots, b_m) = b_j$ and if and only if $g_i(a_1, \dots, b_m) = a_i$ for each two indices i and j. Polyfunctional relations form a polycategory with relational composition.

A $(1,1)$-polyfunctional relation is a pair of inverse functions. A (2,1) or (1,2)-polyfunctional relation is a triple functions $f_0 \colon A_1 \times A_2 \to A_0$, $f_1 \colon A_2 \times A_0 \to A_1$ and $f_2 \colon A_0 \times A_1 \to A_2$ such that $f_1(a_1, a_2) = a_0$ if and only if $f_1(a_2, a_0) = a_1$ and if and only if $f_2(a_0, a_1) = a_2$. Polyfunctional relations are a decategorification of the *multivariable adjunctions* in the work of Shulman [Shu18].

2.2. The Category of Polycategories. Analogously to the categorical and multicategorical case, the theory of polyfunctors and polynatural transformations is synthetised in the 2-category **PolyCat** of polycategories, polyfunctors and polynatural transformations.

Definition 2.4. A *polyfunctor*, $F \colon \mathbb{P} \to \mathbb{Q}$, between two polycategories $\mathbb{P}$ and $\mathbb{Q}$, is an assignment on objects, $F_{obj} \colon \mathbb{P}_{obj} \to \mathbb{Q}_{obj}$ together with an assignment on polymorphisms

$$F_{n,m} \colon \mathbb{P}(X_0, \dots, X_n; Y_0, \dots, Y_m) \to \mathbb{Q}(F_{obj}X_0, \dots, F_{obj}X_n; F_{obj}Y_0, \dots, F_{obj}Y_m).$$

This assignment must be functorial, in that $F(f \fatsemi_{X_i} g) = F(f) \fatsemi_{X_i} F(g)$ and that $F(\mathrm{id}_X) = \mathrm{id}_{FX}$, whenever these are formally well-typed.

Definition 2.5. A polynatural transformation $\theta \colon F \to G$ between two polyfunctors $F, G \colon \mathbb{P} \to \mathbb{Q}$ is given by a family of polymorphisms $\theta_X \in \mathbb{Q}(FX; GX)$

such that, for each polymorphism $f \in \mathbb{P}(X_1, \dots, X_n; Y_1, \dots, Y_m)$, the following naturality condition holds

$$\theta_{X_1} \fatsemi \dots \fatsemi \theta_{X_n} \fatsemi G(f) = F(f) \fatsemi \theta_{Y_1} \fatsemi \dots \fatsemi \theta_{Y_m}.$$

Definition 2.6. Polyfunctors between polycategories form a category, **PolyCat**. This is moreover a 2-category with polynatural transformations.

Remark 2.7. In the same sense that a multifunctor from the terminal multicategory picks a monoid, a polyfunctor from the terminal polycategory should pick a *polyoid* – instead, we call these *Frobenius monoids*.

2.3. Polycategorical Contour.

Definition 2.8. Let $\mathbb{P}$ be a polycategory. Its contour, $\mathsf{Contour}\mathbb{P}$, is the category presented by the following generators and equations:

- two polarized objects, X^ℓ and X^r, for each object $X \in \mathbb{P}_{obj}$;
- for each polymorphism, $f \in \mathbb{P}(X_1, \dots, X_n; Y_1, \dots, Y_m)$, the following generators,

$$\begin{aligned} &f_1^r \colon X_1^r \to X_2^\ell, \quad \dots, \quad f_{n-1}^r \colon X_{n-1}^r \to X_n^\ell, \quad f^d \colon X_n^r \to Y_m^r, \\ &f_1^\ell \colon Y_2^\ell \to Y_1^r, \quad \dots, \quad f_{m-1}^\ell \colon Y_m^\ell \to Y_{m-1}^r, \quad f^u \colon Y_1^\ell \to X_1^\ell, \end{aligned}$$

  having instead $f^u \colon Y_1^\ell \to Y_m^r$ when $n = 0$, having instead $f^d \colon X_n^r \to X_1^\ell$ when $m = 0$, and using no generators for $(0,0)$-polymorphisms;

to which we impose equations requiring contour to preserve identities, $(\mathrm{id}_X)^u = \mathrm{id}_{X^\ell}$ and $(\mathrm{id}_X)^d = \mathrm{id}_{X^r}$; and requiring contour to preserve compositions, meaning that for each $f \in \mathbb{P}(X_1, \dots, X_n; Y_1, \dots, Y_m)$ and each $g \in \mathbb{P}(Z_1, \dots, Z_p; Q_1, \dots, Q_q)$ such that $Y_m = U_1$, the contour of the composition along $Y_m = U_1$ is defined by the following eight cases

$$\begin{aligned} &(f \fatsemi_{X_i} g)^u = f^u; && (f \fatsemi_{X_i} g)_i^r = f_i^r, \text{ for } i = 1, \dots, n-1; \\ &(f \fatsemi_{Y_m} g)_n^r = f^d \fatsemi g_1^r; && (f \fatsemi_{Y_m} g)_j^r = g_{j-n+1}^r, \text{ for } j = n+1, \dots, n+p-2; \\ &(f \fatsemi_{Y_m} g)^d = g^d; && (f \fatsemi_{Y_m} g)_i^\ell = f_i^\ell, \text{ for } i = 1, \dots, m-2; \\ &(f \fatsemi_{Y_m} g)_{m-1}^d = g^u \fatsemi f_{m-1}^\ell; && (f \fatsemi_{Y_m} g)_j^d = g_{j-m+1}^\ell, \text{ for } j = m+1, \dots, m+q-2; \end{aligned}$$

and similar conditions for the rest of the compositions. These equations are depicted in Figure 7.

Proposition 2.9. *Contouring extends to a functor from the category of polycategories to the category of categories,* $\mathcal{C} \colon \mathbf{PolyCat} \to \mathbf{Cat}$.

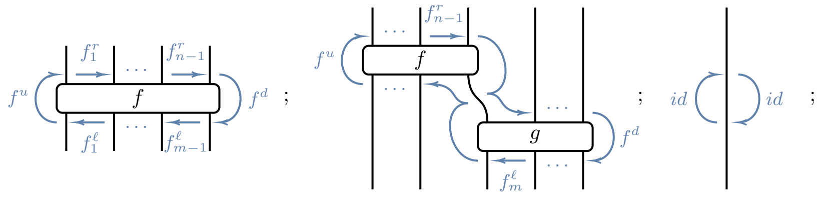


FIGURE 7. Contour of a morphism, composition of contours, and identity contours.

2.4. Malleable Polycategories. A malleable polycategory is a polycategory where each morphism can be morphed uniquely into any possible shape. This means that there exist unique factorizations of each morphism into each one of the possible shapes for composition.

Definition 2.10. The (1,1)-polymorphisms of a polycategory $\mathbb{P}$ form an underlying category $\mathbb{P}^u$. The polymorphisms form profunctors over the polycategory and their composition, in its four possible forms, is dinatural with respect to the underlying category. This means that the following four operations are well-defined:

$$(\fatsemi)_1\colon \left(\int^{X\in\mathbb{P}^u} \mathbb{P}(\Gamma;\Delta,X)\times\mathbb{P}(X,\Gamma';\Delta')\right) \to \mathbb{P}(\Gamma,\Gamma';\Delta,\Delta'),$$
$$(\fatsemi)_2\colon \left(\int^{X\in\mathbb{P}^u} \mathbb{P}(\Gamma;X,\Delta)\times\mathbb{P}(\Gamma',X;\Delta')\right) \to \mathbb{P}(\Gamma,\Gamma';\Delta,\Delta'),$$
$$(\fatsemi)_3\colon \left(\int^{X\in\mathbb{P}^u} \mathbb{P}(\Gamma;\Delta_1,X,\Delta_2)\times\mathbb{P}(X;\Delta)\right) \to \mathbb{P}(\Gamma;\Delta_1,\Delta,\Delta_2),$$
$$(\fatsemi)_4\colon \left(\int^{X\in\mathbb{P}^u} \mathbb{P}(\Gamma;X)\times\mathbb{P}(\Gamma_1,X,\Gamma_2;\Delta)\right) \to \mathbb{P}(\Gamma_1,\Gamma,\Gamma_2;\Delta).$$

Definition 2.11. A *malleable polycategory* is a polycategory where dinatural composition, in all its four forms, is invertible.

Remark 2.12. If a polycategory is malleable, we can reconstruct it up to isomorphism from its binary, cobinary, nullary and conullary maps. When defining a malleable polycategory, it is usually easier to provide these binary, cobinary, unary, nullary and conullary maps, and deduce from those the rest of the structure. The situation is now similar to that of linearly distributive categories: we do not need to provide all n-ary tensors in order to define a linearly distributive category, we only provide the binary $(\otimes,\parr)$ and unary (I,Z) tensors.

This suggests that we will really work with a *biased* version of malleable polycategories, one that privileges the binary and nullary tensors over the others. Biased malleable polycategories are what we will call *prostar autonomous categories.*

2.5. Prostar-Autonomous Categories. Prostar-autonomous categories provide an algebra for both coherent composition and decomposition. Apart from the usual *morphisms*, $\mathbb{V}(X;Y)$; and the *joints*, $\mathbb{V}(X_0 \otimes X_1; Y)$, and *units*, $\mathbb{V}(\top; Y)$, of a promonoidal category; a prostar-autonomous category has *splits*, $\mathbb{V}(X; Y_0 \parr Y_1)$, and *atoms*, $\mathbb{V}(X; \bot)$. As in the case of multicategories, these compositions and decompositions must be coherent, which translates into the existence of natural isomorphisms witnessing a Frobenius rule, the *Frobenius distributors*

$$\varphi_l \colon \int^W \mathbb{C}(A; C \parr W) \times \mathbb{C}(W \otimes B; D) \xrightarrow{\cong} \mathbb{C}(A \otimes B; C \parr D), \text{ and}$$
$$\varphi_r \colon \int^W \mathbb{C}(A \otimes W; C) \times \mathbb{C}(B; W \parr D) \xrightarrow{\cong} \mathbb{C}(A \otimes B; C \parr D).$$

In summary, after this section, we will have developed the relation between malleability and profunctorial structures in an analogous way for both multicategories and polycategories.

Multicategory	Malleable Multicategory	Promonoidal category
Polycategory	Malleable Polycategory	Prostar autonomous category

Definition 2.13. Prostar-autonomous categories are the 2-Frobenius monoids of the monoidal bicategory of profunctors, which is equivalent to the following definition. A *prostar autonomous category* is a category $\mathbb{C}$ endowed with a promonoidal structure $(\mathbb{C}, \otimes, \top)$, and procomonoidal structure $(\mathbb{C}, \parr, \bot)$, that interact as a Frobenius pseudomonoid [DS03, Lau05]. That is, it is a category endowed with four profunctors, suggestively written $\mathbb{C}(\bullet \otimes \bullet; \bullet)$, $\mathbb{C}(\top; \bullet)$, $\mathbb{C}(\bullet; \bot)$ and $\mathbb{C}(\bullet; \bullet \parr \bullet)$, as if they were representable. These profunctors form two promonoidal categories [Day70] with coherent associators and unitors. Further, they are endowed with invertible Frobenius distributors,

$$\varphi_l \colon \int^W \mathbb{C}(A; C \parr W) \times \mathbb{C}(W \otimes B; D) \xrightarrow{\cong} \mathbb{C}(A \otimes B; C \parr D),$$
$$\varphi_r \colon \int^W \mathbb{C}(A \otimes W; C) \times \mathbb{C}(B; W \parr D) \xrightarrow{\cong} \mathbb{C}(A \otimes B; C \parr D),$$

such that every formal diagram formed of these distributors and promonoidal coherences commutes.

Prostar autonomous categories have a canonical *prostar* given by profunctors $\mathbb{C}(\bullet \otimes \bullet; \bot)$ and $\mathbb{C}(\top; \bullet \parr \bullet)$. We may think of a prostar autonomous category as a category $\mathbb{C}$ equipped with sets of polymorphisms $\mathbb{C}(\bullet \otimes ... \otimes \bullet; \bullet \parr ... \parr \bullet)$. The Frobenius isomorphisms let us decompose polymorphisms into combinations of the pro(co)monoidal structures: this decomposition is unique up to dinaturality. Informally, prostar autonomous categories are to polycategories what promonoidal categories are to (co)multicategories.

Definition 2.14. A *prostar functor* $F\colon \mathbb{V} \to \mathbb{W}$ is a quintuple $(F_{obj}, F_{\otimes}, F_{\parr}, F_{\top}, F_{\bot})$ where $(F_{obj}, F_{\otimes}, F_{\top})$ and $(F_{obj}, F_{\parr}, F_{\bot})$ are promonoidal functors that together strictly preserve the Frobenius distributors, in that $\varphi_l ⨟ (F_{\otimes} \times F_{\parr}) = (F_{\parr} \times F_{\otimes}) ⨟ \varphi'_l$ and $\varphi_r ⨟ (F_{\otimes} \times F_{\parr}) = (F_{\otimes} \times F_{\parr}) ⨟ \varphi'_r$. Prostar functors between prostar autonomous categories form a category, **ProStar**.

2.6. Prostar Autonomous are Malleable Polycategories. In this section, we show that the category of prostar autonomous categories is equivalent to that of malleable polycategories. In this sense, the study of malleable polycategories is the study of prostar autonomous categories.

Definition 2.15 (Polycategorical analogue of Definition 3.9)**.** Let $\mathbb{W}$ be a prostar autonomous category. There is a malleable polycategory, $\mathbb{W}^m$, that has the same objects but polymorphisms defined by the elements of the prostar autonomous category. By induction, we define

$$\begin{aligned}
\mathbb{W}^m(X_0, X_1, \Gamma; \Delta) &= \textstyle\int^V \mathbb{W}(X_0 \otimes X_1; V) \times \mathbb{W}^m(V, \Gamma; \Delta),\\
\mathbb{W}^m(; \Delta) &= \textstyle\int^V \mathbb{W}(\top; V) \times \mathbb{W}(V; \Delta),\\
\mathbb{W}^m(X; Y_0, Y_1, \Delta) &= \textstyle\int^V \mathbb{W}^m(X; V, \Delta) \times \mathbb{W}(V; Y_0 \parr Y_1),\\
\mathbb{W}^m(X;) &= \mathbb{W}(X; \bot).
\end{aligned}$$

In other words, the polymorphisms are elements of the left-biased tree reductions of the promonoidal category, seen as a 2-monoid. The four forms of dinatural composition are then defined to be the unique map relating two tree expressions in a 2-Frobenius monoid, which exist uniquely by coherence,

$$\begin{aligned}
&(\mathrm{coh})_1\colon \left(\textstyle\int^{X \in \mathbb{W}^m} \mathbb{W}^m(\Gamma; \Delta, X) \times \mathbb{W}^m(X, \Gamma'; \Delta')\right) \to \mathbb{W}^m(\Gamma, \Gamma'; \Delta, \Delta'),\\
&(\mathrm{coh})_2\colon \left(\textstyle\int^{X \in \mathbb{W}^m} \mathbb{W}^m(\Gamma; X, \Delta) \times \mathbb{W}^m(\Gamma', X; \Delta')\right) \to \mathbb{W}^m(\Gamma, \Gamma'; \Delta, \Delta'),\\
&(\mathrm{coh})_3\colon \left(\textstyle\int^{X \in \mathbb{W}^m} \mathbb{W}^m(\Gamma; \Delta_1, X, \Delta_2) \times \mathbb{W}^m(X; \Delta)\right) \to \mathbb{W}^m(\Gamma; \Delta_1, \Delta, \Delta_2),\\
&(\mathrm{coh})_4\colon \left(\textstyle\int^{X \in \mathbb{W}^m} \mathbb{W}^m(\Gamma; X) \times \mathbb{W}^m(\Gamma_1, X, \Gamma_2; \Delta)\right) \to \mathbb{W}^m(\Gamma_1, \Gamma, \Gamma_2; \Delta).
\end{aligned}$$

Coherence maps are isomorphisms, and so dinatural composition is invertible, making the polycategory malleable. By coherence for pseudomonoids, composition must satisfy associativity and unitality.

Proposition 2.16. *The category of prostar autonomous categories and the category of malleable polycategories are equivalent with the functor* $(\bullet)^m\colon$ **ProStar** $\to$ **mPoly** *induced by the construction of the underlying malleable polycategory of a prostar autonomous category. This is the polycategorical analogue of Proposition 3.10.*

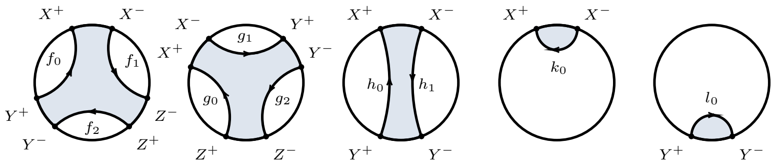
Proof. First, let us show that a prostar functor, $F\colon \mathbb{V} \to \mathbb{W}$, induces a polyfunctor, $F^m\colon \mathbb{V}^m \to \mathbb{W}^m$, between the Underlying polycategories. On objects, we define it to be the same, $F^m_{obj} = F_{obj}$. On polymorphisms, we can define the binary, nullary, cobinary, conullary and unary using the prostar functor structure,

$$F^m_{2,1} = F_\otimes;\ F^m_{0,1} = F_\top;\ F^m_{1,2} = F_\parr;\ F^m_{1,0} = F_\perp;\ \text{and } F^m_{1,1} = F.$$

□

2.7. Splice of a Polycategory.

Definition 2.17. Let $\mathbb{C}$ be a category. Its prostar autonomous category of *spliced arrows*, $S\mathbb{C}$, has underlying category $\mathbb{C}^{\mathrm{op}} \times \mathbb{C}$. Intuitively, its profunctors are defined by spliced circles of morphisms.



Explicitly, it is defined by the following profunctors (below, left). The coherence isomorphisms are defined by glueing circles along the desired boundary and composing the relevant arrows; two compositions are isomorphic if and only if they determine the same arrows (below, right).

Remark 2.18. This structure appeared in Day & Street [DS03, Ex. 7.3], where it was noticed that the canonical promonoidal category induced by a small category [Day70] has an involution. As a multicategory, it was rediscovered by Melliès & Zeilberger [?]. Monoidal spliced arrows were explicitly introduced and characterized as an adjunction in a joint work [EHR23].

$$\mathsf{Splice}(\mathbb{C})\left({}^{X^+}_{X^-}; {}^{Y^+}_{Y^-} \parr {}^{Z^+}_{Z^-}\right) = \mathbb{C}(Y^+; X^+) \times \mathbb{C}(X^-; Z^-) \times \mathbb{C}(Z^+; Y^-);$$

$$\mathsf{Splice}(\mathbb{C})\left({}^{X^+}_{X^-} \otimes {}^{Y^+}_{Y^-}; {}^{Z^+}_{Z^-}\right) = \mathbb{C}(Z^+; X^+) \times \mathbb{C}(X^-; Y^+) \times \mathbb{C}(Y^-; Z^-);$$

$$\mathsf{Splice}(\mathbb{C})\left({}^{X^+}_{X^-}; {}^{Y^+}_{Y^-}\right) = \mathbb{C}(Y^+; X^+) \times \mathbb{C}(X^-; Y^-);$$

$$\mathsf{Splice}(\mathbb{C})\left({}^{X^+}_{X^-}; \perp\right) = \mathbb{C}(X^-; X^+);$$

$$\mathsf{Splice}(\mathbb{C})\left(\top; {}^{Y^+}_{Y^-}\right) = \mathbb{C}(Y^+; Y^-).$$

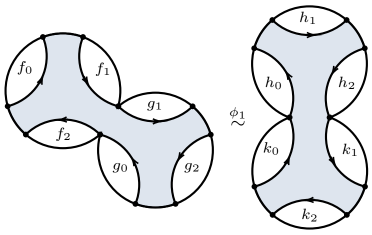


FIGURE 8. Holds if $f_0 = k_0 ⨾ h_0$, $f_1 ⨾ g_1 = h_1$, $g_2 = h_2 ⨾ k_1$, $g_0 ⨾ f_2 = k_2$.

Remark 2.19. $\mathsf{Splice}(\mathbb{C})$ has a representable prostar, given on objects by

$$\binom{X^+}{X^-}^* = \binom{X^-}{X^+}.$$

Proposition 2.20. *Spliced arrows extend to a functor,* $\mathsf{Splice} : \mathbf{Cat} \to \mathbf{ProStar}$.

THEOREM 2.21. *Contour extends to a functor* $\mathsf{Contour}\colon \mathbf{PolyCat} \to \mathbf{Cat}$, *splice extends to a functor* $\mathsf{Splice}\colon \mathbf{Cat} \to \mathbf{ProStar}$. $\mathsf{Contour}$ *is left adjoint to* Splice *composed with the forgetful functor,* $\mathsf{Contour} \dashv \mathsf{Splice} ⨾ \mathsf{Forget}$*; and* $\mathsf{Contour}$ *composed with the forgetful functor is left adjoint to* Splice*, meaning* $\mathsf{Forget} ⨾ \mathsf{Contour} \dashv \mathsf{Splice}$.

PROOF. The proof extends our previous one [EHR23, Theorem 3.7]. □

2.8. Bibliography. Polycategories were defined by Szabo [Sza75] in the symmetric case; Cockett and Seely contributed the planar version we study here [CS97b, BZ20].

Street [Str04] prove that Frobenius pseudomonoids in **Prof** are equivalent to what Day & Street [DS03] call "$*$-autonomous promonoidal categories". The minor twist we take, "prostar autonomous", emphasizes that the canonical prostar may not be representable. When all of the structure including the prostar is representable, we obtain $*$-autonomous categories.

APPENDIX B

Publications

1. Span(Graph): a Canonical Feedback Algebra of Open Transition Systems

Elena Di Lavore, Alessandro Gianola, Mario Román, Pawel Sobocinski, Nicoletta Sabadini
Software and Systems Modeling (SOSYM)

Abstract: We show that Span(Graph)*, an algebra for open transition systems introduced by Katis, Sabadini and Walters, satisfies a universal property. By itself, this is a justification of the canonicity of this model of concurrency. However, the universal property is itself of interest, being a formal demonstration of the relationship between feedback and state. Indeed, feedback categories, also originally proposed by Katis, Sabadini and Walters, are a weakening of traced monoidal categories, with various applications in computer science. A state bootstrapping technique, which has appeared in several different contexts, yields free such categories. We show that Span(Graph)* arises in this way, being the free feedback category over Span(Set). Given that the latter can be seen as an algebra of predicates, the algebra of open transition systems thus arises - roughly speaking - as the result of bootstrapping state to that algebra. Finally, we generalize feedback categories endowing state spaces with extra structure: this extends the framework from mere transition systems to automata with initial and final states.

Declaration: *Hereby I declare that my contribution to this manuscript was to: provide the main theorem and its proof, provide the main idea, write most of the paper with help from my supervisor Pawel Sobocinski and Elena Di Lavore, some examples were provided by Nicoletta Sabadini and Alessandro Gianola.*

Span(Graph): a Canonical Feedback Algebra of Open Transition Systems [⋆]

Elena Di Lavore[1], Alessandro Gianola[2], Mario Román[1], Nicoletta Sabadini[3], and Paweł Sobociński[1]

[1] Tallinn University of Technology, Ehitajate tee 5, 12616 Tallinn, Estonia
[2] Free University of Bozen-Bolzano, Piazza Domenicani, 3, 39100 Bolzano BZ, Italy
[3] Università degli Studi dell'Insubria, Via Ravasi, 2, 21100 Varese VA, Italy

Abstract. We show that **Span**(**Graph**)$_*$, an algebra for *open transition systems* introduced by Katis, Sabadini and Walters, satisfies a universal property. By itself, this is a justification of the canonicity of this model of concurrency. However, the universal property is itself of interest, being a formal demonstration of the relationship between feedback and state. Indeed, *feedback* categories, also originally proposed by Katis, Sabadini and Walters, are a weakening of traced monoidal categories, with various applications in computer science. A *state bootstrapping* technique, which has appeared in several different contexts, yields *free* such categories.
We show that **Span**(**Graph**)$_*$ arises in this way, being the free feedback category over **Span**(**Set**). Given that the latter can be seen as an algebra of predicates, the algebra of open transition systems thus arises – roughly speaking – as the result of bootstrapping state to that algebra.
Finally, we generalize feedback categories endowing state spaces with extra structure: this extends the framework from mere transition systems to automata with initial and final states.

[⋆] Di Lavore, Román and 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.

Table of Contents

1 Introduction . . . 3
  1.1 Related Work . . . 6
  1.2 Synopsis . . . 7
2 Preliminaries: Symmetric Monoidal Categories . . . 7
  2.1 Theories of Processes . . . 7
  2.2 Monoidal Equivalence . . . 12
3 Feedback Categories . . . 14
  3.1 Feedback Categories . . . 14
  3.2 Traced Monoidal Categories . . . 16
  3.3 Delay and Feedback . . . 18
  3.4 $\mathsf{St}(\bullet)$, the Free Feedback Category . . . 20
  3.5 Examples . . . 22
4 Span(Graph): an Algebra of Transition Systems . . . 25
  4.1 The Algebra of Spans . . . 25
  4.2 The Algebra of Open Transition Systems . . . 27
  4.3 Span(Graph) as a Feedback Category . . . 30
  4.4 Cospan(Graph) as a Feedback Category . . . 35
  4.5 Syntactical Presentation of Cospan(FinGraph) . . . 37
5 Structured state spaces . . . 39
  5.1 Structured Feedback Categories . . . 39
  5.2 Structured $\mathsf{St}(\bullet)$ Construction . . . 40
  5.3 Categories of Automata . . . 41
  5.4 Automata in Span(Graph) . . . 42
6 Conclusions and Further Work . . . 44
  6.1 Discussion . . . 44
  6.2 Conclusion . . . 45

1 Introduction

Software engineers need models. In fact, models developed in the early years of computer science have been extremely influential on the emergence of software engineering as a discipline. Prominent examples include flowcharts and state machines, and a part of the reason for their impact and longevity is the fact that they are underpinned by relevant and well-understood mathematical theories.

However, while *concurrent* software has been intensively studied since the early 60s, the theoretical research landscape remains quite fragmented. Indeed, Abramsky [1] argues that the reason for the proliferation of models, their sometimes overly locally-optimised techniques, and the difficulty of understanding and relating their expressivity, is the fact that we still do not have a satisfactory understanding of the underlying mathematical principles of concurrency.

A way to identify such principles and arrive at more canonical models is to look for logical or mathematical justifications. An example is the recent discovery and work on of Curry-Howard style connections between calculi for concurrency and fragments of linear logic, which guided the development of session types [15]. Another possible route is to search for models that satisfy some *universal property*.

The latter approach is the remit of this paper: we focus on the **Span**(**Graph**)$_*$ model of concurrency, introduced by Katis, Sabadini and Walters [33] as an algebra of *open transition systems*, and show that it satisfies a universal property: it is the free feedback category over the category of spans of functions.

The free construction is in itself interesting and can be described as a kind of "state-bootstrapping". We thus position our main result within the theoretical context of feedback categories, their relationship with state, and the more restrictive—yet better known—notion of traced monoidal categories. Our exploration of this wider context is justified, given the panoply of related, yet partial, accounts in the literature.

The relationship between feedback and state is well-known by engineers. In fact, a remarkable fact from electronic circuit design is how data-storing components can be built out of a combination of *stateless components* and *feedback*. A famous example is the (set-reset) "NOR latch": a circuit with two stable configurations that *stores* one bit.

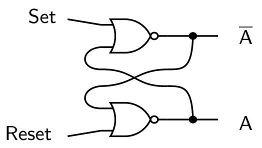


Fig. 1: NOR latch.

The NOR latch is controlled by two inputs, Set and Reset. Activating the first sets the output value to $\mathsf{A} = 1$; activating the second makes the output value return to $\mathsf{A} = 0$. This change is permanent: even when both Set and Reset are deactivated, the feedback loop maintains the last value the circuit was set to[4]—to wit,

[4] In its original description: *"the relay is designed to produce a large and* ***permanent*** *change in the current flowing in an electrical circuit by means of a small electrical stimulus received from the outside"* ([17], emphasis added).

a bit of data has been conjured out of thin air. The results of this paper allow one to see the latch as an instance of a more abstract phenomenon.

Indeed, there is a natural weakening of the notion of traced monoidal categories called *feedback categories* [36]. The construction of the *free* feedback category coincides with a "state-bootstrapping" construction, $\mathsf{St}(\bullet)$, that appears in several different contexts in the literature [7,29,32]. We recall this construction and its mathematical status (Theorem 3.11), which can be summed up by the following intuition:

$$\text{Theory of Processes} + \text{Feedback} = \text{Theory of Stateful Processes.}$$

The **Span**(**Graph**) model of concurrency, introduced in [33], is an algebra of *communicating state machines*, or — equivalently — *open transition systems*.

Let us first explain some terminology. A span $X \to Y$ in a category $\mathbf{C}$ is a pair of morphisms $l\colon A \to X$ and $r\colon A \to Y$ with a common domain (Definition 4.1). When $\mathbf{C}$ has enough structure, spans form a category. This is the case for the category of graphs **Graph**, where objects are graphs and morphisms are, intuitively, pairs of functions that respect the graph structure (Definition 4.6). Summarizing the above, the morphisms of **Span**(**Graph**) are given by pairs of graph homomorphisms, $l\colon G \to X$ and $r\colon G \to Y$, with a common domain G. We think of a span of graphs as a transition system, the graph G, with boundary interfaces X and Y.

Open transition systems interact by synchronization along a common boundary, producing a *simultaneous change of state*. This corresponds to a composition of spans, realized by taking a pullback in **Graph** (see Definition 4.7). The dual algebra of **Cospan**(**Graph**) was introduced in [35] (see Definition 4.17).

Informally, a morphism $X \to Y$ of **Span**(**Graph**) is a state machine with states and transitions, i.e. a finite graph given by the 'head' of the span. The transition system is equipped with left and right interfaces or *communication ports*, X and Y, and every transition is labeled by the effect it produces in *all* its interfaces. Let us focus on some concrete examples.

Let $\mathbb{B} = \{ 0, 1 \}$. We abuse notation by considering $\mathbb{B}$ as a single-vertex graph with two edges, corresponding to the *signals* 0 and 1. Indeed, as we shall see in examples below, it is useful to think of single-vertex graphs as alphabets of signals available on interfaces.

In Figure 2, we depict two open transition systems as arrows of **Span**(**Graph**). The first represents a NOR gate $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$. To give an arrow of this type in **Span**(**Graph**) is to give a span of graph homomorphisms

$$\mathbb{B} \times \mathbb{B} \xleftarrow{l} N \xrightarrow{r} \mathbb{B}.$$

The graphical rendering (Figure 2, left) is a compact representation of the components of this span: the *unlabeled* graph in the bubble is N, and the labels witness the action of two homomorphisms, respectively $l\colon N \to \mathbb{B}\times\mathbb{B}$ and $r\colon N \to \mathbb{B}$. Transitions represent the valid input/output configurations of the NOR gate. For example, the edge with label $(\binom{0}{0}, 1)$, witnesses a transition whose behaviour on the left boundary is $\binom{0}{0}$ and on the right boundary 1. Note that, since the graph N has a single vertex, gates are *stateless* components.

The second component is a span $L = \{\mathsf{Set}, \mathsf{Reset}, \mathsf{Idle}\} \to \{\mathsf{A}, \overline{\mathsf{A}}\} = R$ that models a set-reset latch. The diagram below right (Figure 2), again, is a convenient illustration of the span $L \leftarrow D \to R$. Latches store one bit of information, they are *stateful components*; consequently, their transition graph has two states.

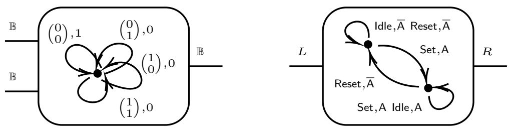


Fig. 2: A NOR gate and set-reset latch, in **Span**(**Graph**).

In both transition systems of Figure 2 the *interfaces* are stateless: indeed, they are determined by a mere set – the self-loops of a single-vertex graph. This is a restriction that occurs rather frequently: in fact, *transition systems with interfaces* are the arrows of the full subcategory of **Span**(**Graph**) on objects that are single-vertex graphs, which we denote by $\mathbf{Span}(\mathbf{Graph})_*$. The objects of $\mathbf{Span}(\mathbf{Graph})_*$ represent interfaces, and a morphism $X \to Y$ encodes a transition system with left interface X and right interface Y. Analogously, the relevant subcategory of **Cospan**(**Graph**) is $\mathbf{Cospan}(\mathbf{Graph})_*$, the full subcategory on sets, or graphs with an empty set of edges.

Definition. $\mathbf{Span}(\mathbf{Graph})_*$ *is the full subcategory of* **Span**(**Graph**) *with objects the single-vertex graphs.*

The problem with $\mathbf{Span}(\mathbf{Graph})_*$ is that it is mysterious from the categorical point of view; the morphisms are graphs, but the boundaries are sets. *Decorated* and *structured* spans and cospans [19,3] are frameworks that capture such phenomena, which occur frequently when composing network structures. Nevertheless, they do not answer the question of *why* they arise naturally.

As stated previously, the main contribution of this paper is the characterization of $\mathbf{Span}(\mathbf{Graph})_*$ in terms of a universal property: it is the free feedback category over the category of spans of functions. We now state this more formally.

Theorem. *The free feedback category over* **Span**(**Set**) *is isomorphic to the full subcategory of* **Span**(**Graph**) *given by single-vertex graphs,* $\mathbf{Span}(\mathbf{Graph})_*$*. That is, there is an isomorphism of categories*

$$\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*.$$

Universal constructions, such as the "state-bootstrapping" $\mathsf{St}(\bullet)$ construction that yields free categories with feedback, characterize the object of interest up to equivalence, making it *the canonical object* satisfying some properties. Recall that Abramsky's concern [1] is that the lack of consensus about the intrinsic primitives of concurrency risks making the results about any particular model of concurrency too dependent on the specific syntax employed. Characterising a model as satisfying a universal property side-steps this concern.

Given that **Span**(**Set**), the category of spans of functions, can be considered an *algebra of predicates* [4,10], the high level intuition that summarizes our main contribution (Theorem 4.12) can be stated as:

$$\text{Algebra of Predicates} + \text{Feedback} = \text{Algebra of Transition Systems.}$$

We similarly prove (in Section 4.4) that the free feedback category over **Cospan**(**Set**) is isomorphic to **Cospan**(**Graph**)$_*$, the full subcategory on discrete graphs of **Cospan**(**Graph**).

Finally, Section 5 shows how the same framework of feedback categories can be extended from transition systems to categories with a structured state space (Theorem 5.6), such as categories of automata. As examples, we recover Mealy deterministic finite automata (Proposition 5.10) and we introduce span automata (Definition 5.11).

1.1 Related Work

This article is an extended version of "A Canonical Algebra of Open Transition Systems" [39], presented at the International Conference on Formal Aspects of Component Software (FACS) 2021. With respect to the conference version, we significantly generalised the framework of feedback categories: Section 5 is completely new material. At the same time, Sections 3 and 4 extend the original manuscript adding new proofs (to propositions 4.9 and 4.10, lemma 4.11, and theorem 4.12) and giving a more complete account of the algebra of spans (Sections 4.1 and 4.2). In an effort to make the paper more self-contained, we also include a new preliminary Section 2, which summarises the necessary concepts from category theory.

Span/**Cospan**(**Graph**) has been used for the modeling of concurrent systems [9,21,22,23,33,35,49,52,53]. Similar approaches to compositional modeling of networks have used *decorated* and *structured cospans* [19,3]. However, these models have not previously been characterized in terms of a universal property.

In [36], the $\mathsf{St}(\bullet)$ construction (under a different name) is exhibited as the free *feedback category*. Feedback categories have been arguably under-appreciated but, at the same time, the $\mathsf{St}(\bullet)$ construction has made multiple appearances as a "state bootstrapping" technique across the literature. The $\mathsf{St}(\bullet)$ construction is used to describe a string diagrammatic syntax for *concurrency theory* in [7]; a variant of it had been previously applied in the setting of *cartesian bicategories* in [32]; and it was again rediscovered to describe a *memoryful geometry of interaction* in [29]. However, a coherent account of both feedback categories and their relation with these stateful extensions has not previously appeared. This motivates our extensive preliminaries in Sections 3.1 and 3.2.

1.2 Synopsis

Section 2 consists of background material on symmetric monoidal categories and equivalences between them. Section 3 contains preliminary discussions on traced monoidal categories and categories with feedback; it explicitly describes $\mathsf{St}(\bullet)$, the free feedback category. It collects mainly expository material. Section 4 exhibits a universal property for the $\mathbf{Span}(\mathbf{Graph})_*$ and $\mathbf{Cospan}(\mathbf{Graph})_*$ models of concurrency and Section 4.5 highlights a specific application. Section 5 extends the framework of feedback categories to capture categories of automata.

2 Preliminaries: Symmetric Monoidal Categories

2.1 Theories of Processes

Resources and processes. We start by setting up an abstract framework for what it means to describe a theory of processes. A theory of processes contains two kinds of components: some *resource types*, which we name $A, B, C, \ldots$; and some *processes*, which we name $f, g, h, \ldots$.

Each process f has an associated input resource type (say, A); and an associated output resource type (say, B). Executing the process f will require some inputs of type A and will produce some outputs of type B. We write this situation as $f\colon A \to B$.

Throughout the paper, we make use of *string diagrams*: a formal diagrammatic syntax for theories of processes [30,40]. In a diagram, every ocurrence of a resource type is represented by a laballed wire; every process is represented by a box, with input wires representing its input type on the left, and output wires representing its output type on the right (Figure 3).

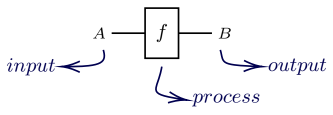
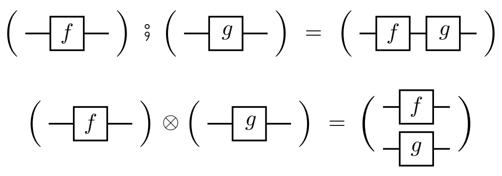
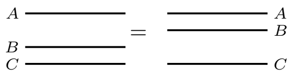


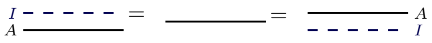
Fig. 3: String diagram for a process $f\colon A \to B$.

Operations in a theory of processes. Theories of processes allow two operations on processes: sequential composition ($\fatsemi$) and parallel composition ($\otimes$). The former is depicted as horizontal concatenation of diagrams, the latter as vertical juxtaposition.



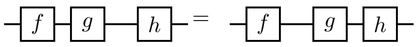
Joining resources. In a theory of processes, resources can be joined. Given a resource type A and a resource type B, we can construct the *joint resource type* $A \otimes B$, which puts together resources of type A and type B. Resource joining may be implemented in diverse ways, depending on the theory of processes. However, it must satisfy some basic axioms:

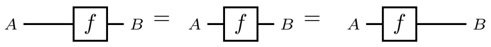
- joining three process resource types together can be done in two ways; these should coincide,
$$A \otimes (B \otimes C) = (A \otimes B) \otimes C, \tag{1}$$


- there must exist a resource type representing the absence of resources, which we call the *unit resource type* I; it must be neutral with respect to process joining
$$A \otimes I = A = A \otimes I. \tag{2}$$


Sequential composition. In a theory of processes, we can compose processes in two different ways. The first is sequential composition: given two processes such that the output type of the first coincides with the input type of the second, say $f \colon A \to B$ and $g \colon B \to C$, their *sequential composition* is the process $(f ⨟ g) \colon A \to C$ that results from executing f and using its output to execute g.

Composing may mean different things in different process theories, but it must always satisfy the following axioms:

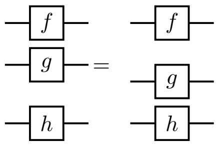
- sequencing together three processes $f \colon A \to B$, $g \colon B \to C$ and $h \colon C \to D$ can be done in two different ways, these should coincide,
$$(f ⨟ g) ⨟ h = f ⨟ (g ⨟ h); \tag{3}$$


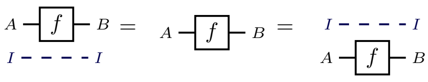
- there must exist a process representing "doing nothing" with a resource A that we write as id_A – the *identity* transformation – which must be neutral with respect to sequential composition,
$$\mathrm{id}_A ⨟ f = f = f ⨟ \mathrm{id}_B. \tag{4}$$


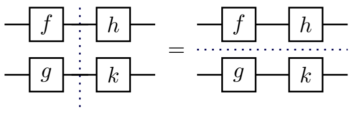
We say that a process $f \colon A \to B$ is *reversible* if it has a reverse counterpart, $f^{-1} \colon B \to A$, such that executing one after the other is the same as having done nothing, $f ⨟ f^{-1} = \mathrm{id}_A$ and $f^{-1} ⨟ f = \mathrm{id}_B$. This is usually called an *isomorphism*. In this situation, we say that A and B are *isomorphic*, and we write that as $A \cong B$.

Parallel composition. The second way of composing two processes is to do so in parallel. Given any two processes $f\colon A \to B$ and $f'\colon A' \to B'$, their parallel composition is a process $(f \otimes f')\colon A \otimes A' \to B \otimes B'$ that results from jointly executing both processes over the joint input resource type, so as to produce the joint output resource type.

The implementation of parallel composition will usually be related to the implementation of resource joining in the same theory. It must satisfy the following axioms:

- composing three processes in parallel can be done in two ways; these should coincide,
$$f \otimes (g \otimes h) = (f \otimes g) \otimes h; \tag{5}$$


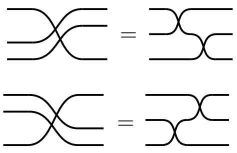
- doing nothing with no resources should be the unit for parallel composition; the identity transformation on the unit resource type I must satisfy
$$f \otimes \mathrm{id}_I = f = \mathrm{id}_I \otimes f; \tag{6}$$


- executing two processes in parallel and then other two processes in parallel must yield the same result as executing in parallel the sequential compositions of both pairs,
$$(f \otimes g) \fatsemi (h \otimes k) = (f \fatsemi h) \otimes (g \fatsemi k). \tag{7}$$


Swapping. Finally, we want to be able to route resources to each specific process. Any theory of processes, given any two resource types A and B, must contain a process $\sigma_{A,B}\colon A \otimes B \to B \otimes A$. This process is called the *swap*, which only permutes the order in which resources are organized. It must satisfy the following axioms.

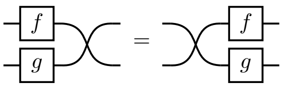
- Swapping twice is the same as swapping once with a joint type,
$$\sigma_{A,B\otimes C} = (\sigma_{A,B} \fatsemi \mathrm{id}_C) \fatsemi (\mathrm{id}_B \otimes \sigma_{A,C}); \tag{8}$$

$$\sigma_{A\otimes B,C} = (\mathrm{id}_A \mathbin{⨟} \sigma_{B,C}) \mathbin{⨟} (\sigma_{A,C} \otimes \mathrm{id}_B). \tag{9}$$



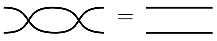
- Swapping two process inputs is the same as swapping the executing place and swapping the output.

$$(f \otimes g) \mathbin{⨟} \sigma_{B,B'} = \sigma_{A,A'} \mathbin{⨟} (g \otimes f). \tag{10}$$



- Swapping and swapping again is the same as doing nothing.

$$\sigma_{A,B} \mathbin{⨟} \sigma_{B,A} = \mathrm{id}_{A\otimes B}. \tag{11}$$



Symmetric monoidal categories. The algebraic structures that capture this notion of process theory are "symmetric monoidal categories" [40]. The resource types are usually called *objects*, while the processes are usually called *morphisms*. Reversible processes are called *isomorphisms*.

Definition 2.1. *A symmetric monoidal category [40] is a tuple*

$$\mathbf{C} = (\mathbf{C}_{\mathrm{obj}}, \mathbf{C}_{\mathrm{mor}}, (⨟), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \sigma),$$

specifying a set of objects, or resource types, $\mathbf{C}_{\mathrm{obj}}$; a set of morphisms, or processes, $\mathbf{C}_{\mathrm{mor}}$; a composition operation; a family of identity morphisms; a tensor operation on objects and morphisms; a unit object and a family of swapping morphisms; satisfying all of the axioms of this section (1-11), possibly up to reversible coherence isomorphisms of the form,

$$\begin{aligned}\alpha_{A,B,C} &\colon (A \otimes B) \otimes C \to A \otimes (B \otimes C),\\ \lambda_A &\colon I \otimes A \to A, \ \textit{and}\\ \rho_A &\colon A \otimes I \to A.\end{aligned}$$

Coherence isomorphisms must commute with all suitably typed processes and must satisfy all possible formal equations between them. We usually denote by $\mathbf{C}(A, B)$ the set of morphisms from A to B.

Note that we do allow the axioms to be satisfied *up to a reversible coherence isomorphism.* For an example, consider the theory of pure functions between sets joined by the cartesian product. It is not true that, given three sets A, B and C, the following two sets are *equal*, $A \times (B \times C) \cong (A \times B) \times C$; they are merely in a one-to-one correspondence. A symmetric monoidal category is *strict* only if these reversible transformations are identities. It was proven by MacLane (his Coherence Theorem, Theorem 2.8 [40]) that the axioms (1-11) are valid for both strict and non-strict monoidal categories.

Example 2.2. The paradigmatic theory of processes uses mathematical sets as types and functions as processes. We can check that the following functions, with the cartesian product, satisfy the axioms (1-11), thus forming a symmetric monoidal category.

$$\begin{aligned}\mathbf{Set} = (&\mathsf{Sets}, \mathsf{Functions}, (\circ), \mathrm{id}, \times, \langle \bullet, \bullet \rangle, 1, (a,(b,c)) \mapsto ((a,b),c), \\ &(a,*) \mapsto a, (*,a) \mapsto a, (a,b) \mapsto (b,a)).\end{aligned}$$

Example 2.3. The theory of linear transformations uses dimensions (natural numbers) as types and matrices over the real numbers as processes. We can check that matrices, with the direct sum, satisfy the axioms (1-11), thus forming a symmetric monoidal category.

$$\mathbf{Mat} = (\mathbb{N}, \mathsf{Matrices}, (\cdot), (+), \oplus, 0, \mathbf{I}, \mathbf{I}, \mathbf{I}, \mathbf{I}, \mathbf{S}),$$

where $\mathbf{I}$ is the identity matrix and $\mathbf{S}$ is the permutation matrix,

$$\mathbf{I}_n = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots_n & \vdots \\ 0 & \dots & 1 \end{pmatrix}; \qquad \mathbf{S}_{n,m} = \begin{pmatrix} 0 & \dots & 0 & 1 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots_n & \vdots \\ 0 & \dots & 0 & 0 & \dots & 1 \\ 1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots_m & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 1 & 0 & \dots & 0 \end{pmatrix}.$$

Example 2.4. It can happen that two theories of processes share the same elements, but differ on how they are combined. The theory of choice in finite sets uses again functions, but instead of the cartesian product, it uses the disjoint union. We can check that the following functions satisfy again the axioms (1-11).

$$\begin{aligned}\mathbf{FinSet} = (&\mathsf{FinSets}, \mathsf{Functions}, (\circ), \mathrm{id}, (+), [\bullet, \bullet], 0, (a|(b|c)) \mapsto ((a|b)|c), \\ &(a|\emptyset) \mapsto a, (\emptyset|a) \mapsto a, (a|b) \mapsto (b|a)).\end{aligned}$$

When designing software, the advantage of an algebraic structure such as monoidal categories is reusability: we can encapsulate the operations of our theory of processes into a separate module, and we can abstractly work with them without knowing the particulars of the theory of processes at hand. The axioms (1-11) are straightforward to check for most theories of processes – even if we will not take the time to do so in this text – but they are a powerful abstraction: once the axioms are satisfied, we can start reasoning with string diagrams.

2.2 Monoidal Equivalence

In this final preliminary section, we recall what it means to have a transformation between monoidal categories (symmetric strong monoidal functor, Definition 2.5), what it means to have two equivalent monoidal categories (monoidal equivalence, Definition 2.7) and the statement of the Coherence Theorem: every monoidal category is equivalent to a strict one (Theorem 2.8).

Monoidal functors. Every time we consider an algebraic structure, it is natural to also consider what is a good notion of transformation between two such algebraic structures. A transformation of algebraic structures should preserve the key ingredients of the algebraic construction. In the case of symmetric monoidal categories, these transformations are called *monoidal functors*, and they preserve the operation of composition.

Definition 2.5. *A* symmetric strong monoidal functor *between two symmetric monoidal categories with coherence isomorphisms*

$$\mathbf{C} = (\mathbf{C}_{\mathrm{obj}}, \mathbf{C}_{\mathrm{mor}}, (\fatsemi), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}}, \sigma^{\mathbf{C}}), \text{ and}$$
$$\mathbf{D} = (\mathbf{D}_{\mathrm{obj}}, \mathbf{D}_{\mathrm{mor}}, (\fatsemi), \mathrm{id}, (\otimes)_{\mathrm{obj}}, (\otimes)_{\mathrm{mor}}, I, \alpha^{\mathbf{D}}, \lambda^{\mathbf{D}}, \rho^{\mathbf{D}}, \sigma^{\mathbf{D}})$$

is a tuple $\mathbf{F} = (F_{\mathrm{obj}}, F_{\mathrm{mor}}, \phi, \varphi)$*, consisting of*

- *a function that assigns objects of the first category to objects of the second category,* $F_{\mathrm{obj}} \colon \mathbf{C}_{\mathrm{obj}} \to \mathbf{D}_{\mathrm{obj}}$*,*
- *and a function that assigns morphisms of the first category to morphisms of the second category,* $F_{\mathrm{mor}} \colon \mathbf{C}_{\mathrm{mor}} \to \mathbf{D}_{\mathrm{mor}}$*.*
- *a coherence isomorphism* $\phi_{A,B} \colon FA \otimes FB \to F(A \otimes B)$*,*
- *and a coherence isomorphism* $\varphi \colon J \to FI$*.*

Traditionally, functions both on objects, F_{obj} *and morphisms,* F_{mor} *are denoted by* F*. The functor must be such that every morphism* $f \colon A \to B$ *is assigned a morphism* $F(f) \colon FA \to FB$*, whose source and target are the images of the original source and target. Moreover, it must satisfy the following axioms,*

- *compositions must be preserved,* $F(f \fatsemi g) = F(f) \fatsemi F(g)$*,*
- *identities must be preserved,* $F(\mathrm{id}_A) = \mathrm{id}_{FA}$*,*
- *tensoring must be transported by the natural transformations, meaning that*

$$F(f \otimes g) = \mu \fatsemi (F(f) \otimes F(g)) \fatsemi \mu^{-1},$$

- *associators, unitors and swaps must be transported by the natural transformations, meaning that*

$$\begin{aligned} F(\alpha^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\mu^{-1} \otimes \mathrm{id}) \fatsemi \alpha^{\mathbf{D}} \fatsemi (\mathrm{id} \otimes \mu) \fatsemi \mu, \\ F(\lambda^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\varphi^{-1} \otimes \mathrm{id}) \fatsemi \lambda^{\mathbf{D}}, \\ F(\rho^{\mathbf{C}}) &= \mu^{-1} \fatsemi (\mathrm{id} \otimes \varphi^{-1}) \fatsemi \rho^{\mathbf{D}}, \\ F(\sigma^{\mathbf{C}}) &= \mu^{-1} \fatsemi \sigma^{\mathbf{D}} \fatsemi \mu. \end{aligned}$$

Example 2.6. For instance, there is a strong monoidal functor translating from the theory of choice in finite sets, $\mathbf{FinSet}_+$ (Example 2.4), to the theory of linear transformations $\mathbf{Mat}$ (Example 2.3) that sends the finite sets $A = \{a_0, \dots, a_{n-1}\}$ and $B = \{b_0, \dots, b_{m-1}\}$ to their cardinalities, n and m; and each function $f\colon A \to B$ to the matrix $F_{ij}\colon n \to m$ that contains a 1 on the entry F_{ij} when $f(a_i) = b_j$, and contains a 0 otherwise.

Definition 2.7. *A* monoidal equivalence *of categories is a symmetric strong monoidal functor $F\colon \mathbf{C} \to \mathbf{D}$ that is*

1. essentially surjective *on objects, meaning that for each $X \in \mathbf{D}_{\text{obj}}$, there exists $A \in \mathbf{C}_{\text{obj}}$ such that $F(A) \cong X$;*
2. essentially injective *on objects, meaning that $F(A) \cong F(B)$ implies $A \cong B$; it can be proven that every monoidal functor is essentially injective, so this condition, though conceptually important, is superfluous;*
3. surjective on morphisms*, or* full*, meaning that for each $g\colon FA \to FB$ there exists some $f\colon A \to B$ such that $F(f) = g$;*
4. injective on morphisms*, or* faithful*, meaning that given any two morphisms $f\colon A \to B$ and $g\colon A \to B$ such that $F(f) = F(g)$, it holds that $f = g$.*

In this situation, we say that $\mathbf{C}$ and $\mathbf{D}$ are equivalent*, and we write that as $\mathbf{C} \cong \mathbf{D}$. Moreover, when the monoidal functor is injective and surjective on objects, we say that $\mathbf{C}$ and $\mathbf{D}$ are* isomorphic*.*

Theorem 2.8 (Coherence theorem, [40, Theorem 2.1, Chapter VII]). *Every monoidal category is monoidally equivalent to a strict monoidal category.*

Let us comment further on how we use the coherence theorem. Each time we have a morphism $f\colon A \to B$ in a monoidal category, we have a corresponding morphism $A \to B$ in its strictification. This morphism can be lifted to the original category to uniquely produce, say, a morphism $(\lambda_A \mathbin{⨟} f \mathbin{⨟} \lambda_B^{-1})\colon I \otimes A \to I \otimes B$. Each time the source and the target are clearly determined, we simply write f again for this new morphism.

The reason to avoid this explicit notation on our definitions and proofs is that it would quickly become verbose and distractive. Equations seem conceptually easier to understand when written assuming the coherence theorem – and they become even clearer when drawn as string diagrams, which implicitly hide these bureaucratic isomorphisms. In fact, in the work of Katis, Sabadini and Walters [36], strictness is assumed from the start for the sake of readibility, even though—as argued above—it is not a necessary assumption.

Theorem 2.8 and Section 2.1 can be summarized by the slogan:

> *"Any theory of processes satisfying the axioms of symmetric monoidal categories (1-11) can be reasoned about using string diagrams".*

3 Feedback Categories

In this section we recall feedback categories, originally introduced in [36], and contrast them with the stronger notion of *traced monoidal categories* in Section 3.2. We discuss the relationship between feedback and delay in Section 3.3. Next, we recall the construction of the *free* feedback category in Section 3.4, and give examples in Section 3.5.

3.1 Feedback Categories

Feedback categories [36] were motivated by examples such as *Elgot automata* [18], *iteration theories* [6] and *continuous dynamical systems* [34]. These categories feature a *feedback operator*, $\mathsf{fbk}(\bullet)$, which takes a morphism $S \otimes A \to S \otimes B$ and *"feeds back"* one of its outputs to one of its inputs of the same type, yielding a morphism $A \to B$ (Figure 4, left). When using string diagrams, we depict the action of the feedback operator as a loop with a double arrowtip (Figure 4, right): string diagrams must be acyclic, and so the feedback operator cannot be confused with a normal wire.

$$\frac{f \colon S \otimes A \to S \otimes B}{\mathsf{fbk}_S(f) \colon A \to B}$$

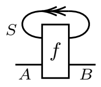


Fig. 4: Type and graphical notation for the operator $\mathsf{fbk}_S(\bullet)$.

Capturing a reasonable notion of feedback requires the operator to interact coherently with the flow imposed by the structure of a symmetric monoidal category. This interaction is expressed by a few straightforward axioms, which we list below.

Definition 3.1. *A* feedback category [36] *is a symmetric monoidal category* $\mathbf{C}$ *endowed with an operator* $\mathsf{fbk}_S \colon \mathbf{C}(S \otimes A, S \otimes B) \to \mathbf{C}(A, B)$*, which satisfies the following axioms (A1-A5, see also Figure 5).*

(A1). Tightening. *Feedback must be natural in* $A, B \in \mathbf{C}$*, its input and output. This is to say that for every morphism* $f \colon S \otimes A \to S \otimes B$ *and every pair of morphisms* $u \colon A' \to A$ *and* $v \colon B \to B'$*,*

$$u \mathbin{⨾} \mathsf{fbk}_S(f) \mathbin{⨾} v = \mathsf{fbk}_S((\mathrm{id} \otimes u) \mathbin{⨾} f \mathbin{⨾} (\mathrm{id} \otimes v)).$$

(A2). Vanishing. *Feedback on the empty tensor product, the unit, does nothing. That is to say that, for every* $f \colon A \to B$*,*

$$\mathsf{fbk}_I(f) = f.$$

(A3). Joining. *Feedback on a monoidal pair is the same as two consecutive applications of feedback. That is to say that, for every morphism* $f \colon S \otimes T \otimes A \to S \otimes T \otimes B$*,*

$$\mathsf{fbk}_T(\mathsf{fbk}_S(f)) = \mathsf{fbk}_{S \otimes T}(f).$$

(A4). Strength. Feedback has the same result if it is taken in parallel with another morphism. That is to say that, for every morphism $f\colon S \otimes A \to S \otimes B$ and every morphism $g\colon A' \to B'$,

$$\mathsf{fbk}_S(f) \otimes g = \mathsf{fbk}_S(f \otimes g).$$

(A5). Sliding. Feedback is invariant to applying an isomorphism "just before" or "just after" the feedback. In other words, feedback is dinatural over the isomorphisms of the category. That is to say that for every $f\colon T \otimes A \to S \otimes B$ and every isomorphism $h\colon S \to T$,

$$\mathsf{fbk}_T(f \fatsemi (h \otimes \mathrm{id})) = \mathsf{fbk}_S((h \otimes \mathrm{id}) \fatsemi f).$$

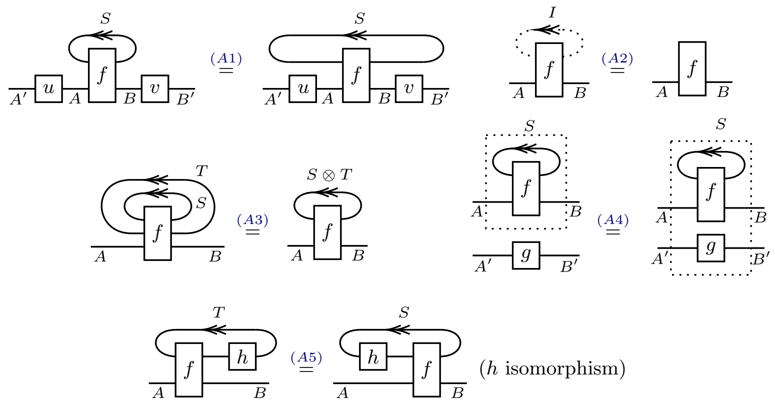


Fig. 5: Diagrammatic depiction of the axioms of feedback.

The natural notion of homomorphism between feedback categories is that of a symmetric monoidal functor that moreover preserves the feedback structure. These are called *feedback functors*.

Definition 3.2. *A feedback functor $F\colon \mathbf{C} \to \mathbf{D}$ between two feedback categories $(\mathbf{C}, \mathsf{fbk}^{\mathbf{C}})$ and $(\mathbf{D}, \mathsf{fbk}^{\mathbf{D}})$ is a strong symmetric monoidal functor such that feedback is transported, that is,*

$$F(\mathsf{fbk}^{\mathbf{C}}_S(f)) = \mathsf{fbk}^{\mathbf{D}}_{F(S)}(\mu \fatsemi Ff \fatsemi \mu^{-1}),$$

where $\mu_{A,B}\colon F(A) \otimes F(B) \to F(A \otimes B)$ is the isomorphism of the strong monoidal functor F. We write $\mathsf{Feedback}$ *for the category of (small) feedback categories and feedback functors. There is a forgetful functor $\mathcal{U}\colon$* $\mathsf{Feedback} \to \mathsf{SymMon}$.

Remark 3.3. Thanks to the coherence theorem (Theorem 2.8), we can present the axioms of a feedback category as in Definition 3.1, omitting associators and unitors. In fact, to be explicit, the statement of the vanishing axiom is

$$\mathsf{fbk}_I(\lambda_A \mathbin{⨾} f \mathbin{⨾} {\lambda_B}^{-1}) = f$$

because the feedback operator, fbk_I, needs to be applied to a morphism $I \otimes A \to I \otimes B$, and the only morphism whose strictification has type $A \to B$ is $(\lambda_A \mathbin{⨾} f \mathbin{⨾} {\lambda_B}^{-1}) \colon I \otimes A \to I \otimes B$ (see Theorem 2.8). Similarly, the joining axiom really states that

$$\mathsf{fbk}_S(\mathsf{fbk}_T(f)) = \mathsf{fbk}_{S\otimes T}(\alpha_{S,T,A} \mathbin{⨾} f \mathbin{⨾} \alpha^{-1}_{S,T,B}).$$

Remark 3.4. Our feedback operator takes a morphism $S \otimes A \to S \otimes B$ with the first component S of the tensor in both the domain and the codomain being the object "fed back". Given that S appears in the first position in both the domain and the codomain, we refer to this as *aligned feedback.*

An alternative definition is possible, and appears in the exposition of traces by Ponto and Shulman [45]. We call this *twisted feedback*: here $\mathsf{fbk}(\bullet)$ is an operator that takes a morphism $S \otimes A \to B \otimes S$—note the position of S in the codomain—and yields a morphism $A \to B$.

$$\frac{f \colon S \otimes A \to B \otimes S}{\mathsf{fbk}_S(f) \colon A \to B}$$

The advantage of using *twisted feedback* is that sequential composition of processes with feedback does not require symmetry of the underlying monoidal category (see [32], where the authors consider a category with twisted feedback). However, parallel composition *does* require symmetry. Given that we study the monoidal category of feedback processes, and aligned feedback diagrams are more readable, we use only aligned feedback in this paper.

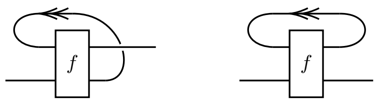


Fig. 6: Twisted vs. aligned feedback

3.2 Traced Monoidal Categories

Feedback categories are a weakening of traced monoidal categories, which have found several applications in computer science. Indeed, since their conception [30] as an abstraction of the *trace* of a matrix in linear algebra, they were used in linear logic and geometry of interaction [1,24,25], programming language semantics [27], semantics of recursion [2] and fixed point operators [28,5].

Between feedback categories and traced monoidal categories there is an intermediate notion called *right traced category* [50]. Here, the sliding axiom applies not only to isomorphisms but rather to arbitrary morphisms. This strengthening is already unsuitable for our purposes (see Remark 3.12). However, the difference in the sliding axiom is not dramatic: we will generalize the notion of feedback category to allow the choice of morphisms that can be "slid" through the feedback loop (Section 5). For example, it is possible to require the sliding axiom for all the morphisms, as in the case of right traced categories, or just isomorphisms, as in the case of feedback categories. The more serious conceptual difference between feedback categories and traced monoidal categories is the "yanking axiom" of traced monoidal categories (in Figure 7). The yanking axiom is incontestably elegant from the geometrical point of view: strings are "pulled", and feedback (the loop with two arrowtips) disappears.

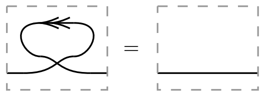


Fig. 7: The yanking axiom.

Strengthening the sliding axiom and adding the yanking axiom yields the definition of traced monoidal category.

Definition 3.5. *A* traced monoidal category *[30,50] is a feedback category that additionally satisfies the* yanking axiom $\mathsf{fbk}(\sigma) = \mathrm{id}$ *and the* sliding axiom*,* $\mathsf{fbk}_T(f \mathbin{⨾} (h \otimes \mathrm{id})) = \mathsf{fbk}_S((h \otimes \mathrm{id}) \mathbin{⨾} f)$*, for an arbitrary morphism* $h\colon S \to T$*. We commonly denote by* $\mathsf{tr}(\bullet)$ *the feedback operator of a traced monoidal category.*

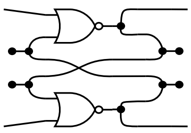


Fig. 8: Diagram for the NOR latch, modeled with a trace in **Span**(**Graph**).

There is scope for questioning the validity of the yanking axiom in many applications that feature feedback. If feedback can disappear without leaving any imprint, that must mean that it is *instantaneous*: its output necessarily mirrors its input.[5] Importantly for our purposes, this implies that a feedback satisfying the yanking equation is "memoryless", or "stateless".

[5] In other words, traces are used to talk about processes in *equilibrium*, processes that have reached a *fixed point*. A theorem by Hasegawa [28] and Hyland [5] corroborates this interpretation: a trace in a cartesian category corresponds to a *fixpoint operator*.

In engineering and computer science, instantaneous feedback is actually a rare concept; a more common notion is that of *guarded feedback*. Consider *signal flow graphs* [51,41]: their categorical interpretation in [8] models feedback not by the usual trace, but by a trace "guarded by a register", that *delays the signal* and violates the yanking axiom (see Remark 7.8 *op.cit.*).

Example 3.6. Let us return to our running example of the NOR latch from Figure 1. We have seen how to model NOR gates in **Span**(**Graph**) in Figure 2, and the algebra of **Span**(**Graph**) does include a trace. However, imitating the real-world behavior of the NOR latch with *just* a trace is unsatisfactory: the trace of **Span**(**Graph**) is built out of stateless components, and tracing stateless components yields a stateless component (see Figure 8, later detailed in Section 4.2).

3.3 Delay and Feedback

As we have discussed previously, the major conceptual difference between feedback categories and traced monoidal categories is the rejection of the yanking axiom. Indeed, a non-trivial delay is what sets apart feedback categories from traced monoidal categories.

We can isolate the delay component in a feedback category. Consider the process that only "feeds back" the input to itself and then just outputs that "fed back" input. The process interpretation of monoidal categories (Section 2.1) allows us to understand this process as delaying its input and returning it as output [16]. This process, $\partial_A \coloneqq \mathsf{fbk}_A(\sigma_{A,A})$, is called the *delay endomorphism* and is illustrated in Figure 9.

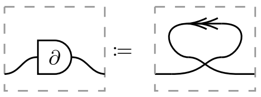


Fig. 9: Definition of *delay*.

If a category has enough structure, feedback can be understood as the combination of *trace* and *delay* in a formal sense. *Compact closed categories* are traced monoidal categories where every object A has a dual $A^\star$ and the trace is constructed from two pieces $\varepsilon\colon A \otimes A^\star \to I$ and $\eta\colon I \to A^\star \otimes A$. While not every traced monoidal category is compact closed, they all embed fully faithfully into a compact closed category.[6] In a compact closed category, a feedback operator is necessarily a trace "guarded" by a *delay*.

Proposition 3.7 (Feedback from delay [7]). *Let* **C** *be a compact closed category with* $\mathsf{fbk}^{\mathbf{C}}$ *a feedback operator that takes a morphism* $S \otimes A \to S \otimes B$ *to a morphism* $A \to B$*, satisfying the axioms of feedback (in Figure 5) but possibly*

[6] This is the **Int** construction from Joyal, Street and Verity [30].

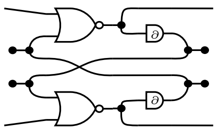


Fig. 10: NOR latch with feedback.

failing to satisfy the yanking axiom (Figure 7) of traced monoidal categories. Then, the feedback operator is necessarily of the form

$$\mathsf{fbk}^{\mathbf{C}}_S(f) \coloneqq (\eta \otimes \mathrm{id}) \mathbin{;} (\mathrm{id} \otimes f) \mathbin{;} (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}) \mathbin{;} (\varepsilon \otimes \mathrm{id})$$

where $\partial_A \colon A \to A$ is a family of endomorphisms satisfying

- $\partial_A \otimes \partial_B = \partial_{A \otimes B}$ *and* $\partial_I = \mathrm{id}$*, and*
- $\partial_A \mathbin{;} h = h \mathbin{;} \partial_B$ *for each isomorphism* $h \colon A \cong B$*.*

In fact, any family of morphisms ∂_A satisfying these properties determines uniquely a feedback operator that has ∂_A as its delay endomorphisms.

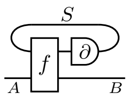


Fig. 11: Feedback from delay.

Proof. Given a family ∂_S satisfying the two properties, we can define a feedback structure, shown in Figure 11, to be $\mathsf{fbk}^{\mathbf{C}}_S(f) \coloneqq (\eta \otimes \mathrm{id}) \mathbin{;} (\mathrm{id} \otimes f) \mathbin{;} (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}) \mathbin{;} (\varepsilon \otimes \mathrm{id})$ and check that it satisfies all the axioms of feedback (Figure 5). Note here that, as expected, the yanking equation is satisfied precisely when delay endomorphisms are identities, $\partial_A = \mathrm{id}_A$.

Let us now show that any feedback operator in a compact closed category is of this form (Figure 12). Indeed,

$$\begin{aligned}
\mathsf{fbk}^{\mathbf{C}}_S(f) &= \mathsf{fbk}^{\mathbf{C}}_S((\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id}) \mathbin{;} (\sigma \otimes \sigma \otimes f) \mathbin{;} (\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id})) \\
&= (\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id}) \mathbin{;} (\mathsf{fbk}^{\mathbf{C}}_S(\sigma) \otimes \sigma \otimes f) \mathbin{;} (\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id}) \\
&= (\eta \otimes \mathrm{id}) \mathbin{;} (\mathrm{id} \otimes f) \mathbin{;} (\mathrm{id} \otimes \mathsf{fbk}^{\mathbf{C}}_S(\sigma) \otimes \mathrm{id}) \mathbin{;} (\varepsilon \otimes \mathrm{id}).
\end{aligned}$$

Here we have used the fact that the trace is constructed by two separate pieces: ε and η; and then the fact that the feedback operator, like trace, can be applied “locally” (see the axioms in Figure 5). □

Example 3.8. Consider again the NOR latch of Figure 1. The algebra of the category **Span**(**Graph**) does include a feedback operator that is *not* a trace – the difference is an additional *stateful* delay component. As we shall see, this notion of feedback is canonical. We shall also see that the delay enables us to capture the real-world behavior of the NOR latch (Figure 10).

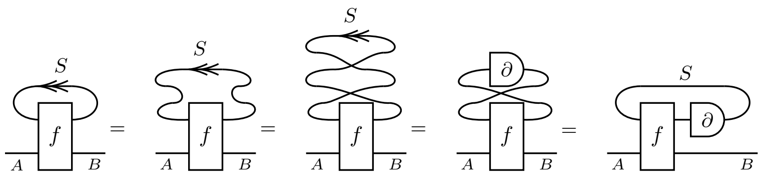


Fig. 12: Feedback in a compact closed category.

The emergence of state from feedback is witnessed by the St(•) construction, which we recall below.

3.4 St(•), the Free Feedback Category

Here we show how to obtain the free feedback category on a symmetric monoidal category. The St(•) construction is a general way of endowing a system with state. It appears multiple times in the literature in slightly different forms: it is used to arrive at a stateful resource calculus in [7]; a variant is used for geometry of interaction in [29]; it coincides with the free feedback category presented in [36]; and yet another, slightly different formulation was given in [32].

Definition 3.9 (Category of stateful processes, [36]). *Let $(\mathbf{C}, \otimes, I)$ be a symmetric monoidal category. We write $\mathsf{St}(\mathbf{C})$ for the category with the objects of $\mathbf{C}$ but where morphisms $A \to B$ are pairs $(S \mid f)$, consisting of a* state space *$S \in \mathbf{C}$ and a morphism $f \colon S \otimes A \to S \otimes B$. We consider morphisms up to isomorphism classes of their state space, and thus*

$$(S \mid f) = (T \mid (h^{-1} \otimes \mathrm{id}) \fatsemi f \fatsemi (h \otimes \mathrm{id})), \qquad \textit{for any isomorphism } h \colon S \cong T.$$

When depicting a stateful process (Figure 13), we mark the state strings.

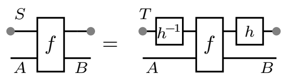


Fig. 13: Equivalence of stateful processes. We depict stateful processes by marking the space state.

We define the *identity stateful process* on $A \in \mathbf{C}$ as $(I \mid \mathrm{id}_{I \otimes A})$. *Sequential composition* of the two stateful processes $(S \mid f) \colon A \to B$ and $(T \mid g) \colon B \to C$ is defined by $(S \mid f) \fatsemi (T \mid g) = (S \otimes T \mid (\sigma \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes f) \fatsemi (\sigma \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g))$, see Figure 14, left. *Parallel composition* of the two stateful processes $(S \mid f) \colon A \to B$ and $(S' \mid f') \colon A' \to B'$ is defined by $(S \mid f) \otimes (S' \mid f') = (S \otimes S' \mid (\mathrm{id} \otimes \sigma \otimes \mathrm{id}) \fatsemi (f \otimes f') \fatsemi (\mathrm{id} \otimes \sigma \otimes \mathrm{id}))$, see Figure 14, right. In both cases, the state spaces of the components are tensored together.

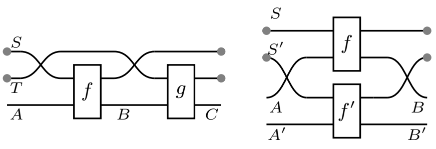


Fig. 14: Sequential and parallel composition of stateful processes.

This defines a symmetric monoidal category. Moreover, the operator

$$\mathsf{store}_T(S \mid f) \coloneqq (S \otimes T \mid f), \text{ for } f \colon S \otimes T \otimes A \to S \otimes T \otimes B,$$

which "stores" some information into the state, makes it a feedback category, see Figure 15.

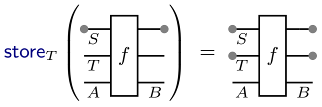


Fig. 15: The $\mathsf{store}(\bullet)$ operation, diagrammatically.

Proposition 3.10. *Sequential composition of stateful processes is associative. That is, for every $f \colon S \otimes A \to S \otimes B$, every $g \colon T \otimes B \to T \otimes C$ and every $h \colon R \otimes C \to R \otimes D$,*

$$((S \mid f) \fatsemi (T \mid g)) \fatsemi (R \mid h) = (S \mid f) \fatsemi ((T \mid g) \fatsemi (R \mid h)).$$

Proof. We can see both morphisms are equal by applying transformations of string diagrams: i.e. the axioms of symmetric monoidal categories (Figure 16).

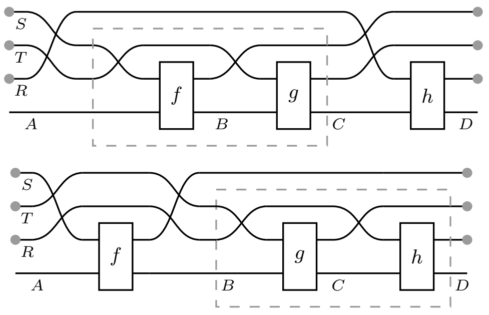


Fig. 16: Associativity of sequential composition.

The state spaces are isomorphic thanks to the associator $\alpha\colon (S \otimes T) \otimes R \to S \otimes (T \otimes R)$. □

Unitality and monoidality of stateful processes follow a similar reasoning. These properties yield the following result.

Theorem 3.11 ([36], Proposition 2.6). *The category $\mathsf{St}(\mathbf{C})$, endowed with the $\mathsf{store}(\bullet)$ operator, is the free feedback category over a symmetric monoidal category $\mathbf{C}$.*

Remark 3.12. Stateful processes are defined *up to isomorphism of the state space*. This is captured by axiom (A5) of feedback categories and, as mentioned in Section 3.2, relaxing it to allow sliding of arbitrary morphisms, would yield a notion of equality of stateful processes that would be too strong for our purposes: it would equate automata with a different number of states and boundary behavior (Example 4.14). Considering stronger notions of equivalence of processes is possible and leads to interesting models of computation [16]. Expanding this line of research is outside the scope of the present manuscript.

Remark 3.13 (Coherence and sliding). There are cases where we do need to be careful about the correct use of associators and unitors. For instance, we could be tempted to conclude that coherence implies that, for any $f\colon ((S\otimes T)\otimes R)\otimes A \to ((S\otimes T)\otimes R)\otimes B$, the following equation holds $((S\otimes T)\otimes R \mid f) = (S\otimes(T\otimes R) \mid f)$ without needing to invoke the equivalence relation of stateful processes. This would allow us to construct the category $\mathsf{St}(\bullet)$ of stateful processes without having to quotient them by the equivalence relation. However, this equality is only enabled by the fact that $\alpha_{S,T,R}$ is an isomorphism: we have

$$((S \otimes T) \otimes R \mid f) = (S \otimes (T \otimes R) \mid \alpha_{S,R,T} \mathbin{⨾} f \mathbin{⨾} \alpha^{-1}_{S,R,T}),$$

even if we write the equation omitting the coherence maps. This is also what will allow us to notate stateful processes diagramatically. We will mark the wires forming the state space; the order in which they are tensored does not matter thanks again to the equivalence relation that we are imposing.

3.5 Examples

All *traced monoidal categories* are feedback categories, since the axioms of feedback are a strict weakening of the axioms of trace. A more interesting source of examples is the $\mathsf{St}(\bullet)$ construction we just defined. We present some examples of state constructions below.

Example 3.14 (Mealy transition systems). A *Mealy deterministic transition system* with boundaries A and B, and state space S was defined [42, §2.1] to be just a function $f\colon S \times A \to S \times B$. It is not difficult to see that, up to isomorphism of the state space, they are morphisms of $\mathsf{St}(\mathbf{Set})$. They compose following Definition 3.9, and form a feedback category $\mathbf{Mealy} \coloneqq \mathsf{St}(\mathbf{Set})$.

Definition 3.15. *A* Mealy transition system *from A to B is a tuple* $\mathbf{M} = (S, t, o)$*, where S is a set called the* state space*, $t\colon S \times A \to S$ is a function called the* transition function*, and $o\colon S \times A \to B$ is a function called the* output function*.*

Two Mealy transition systems are equal whenever their transition functions are equal up to isomorphism of the state space. That is, two deterministic transition systems $\mathbf{M} = (S, t_M, o_M)$ and $\mathbf{N} = (T, t_N, o_N)$ are considered equal whenever there exists an isomorphism $h\colon S \cong T$ between their state spaces such that

$$h(t_M(s, a)) = t_N(h(s), a) \quad \text{and} \quad o_M(s, a) = o_N(s, a).$$

Whenever $t(s_0, a) = s_1$ and $o(s_0, a) = b$, we write $s_0 \overset{a/b}{\to} s_1$. We may also write a transition and output in a single function, $f(s_0, a) = (t(s_0, a), o(s_0, a)) = (s_1, b)$.

The feedback of **Mealy** transition systems transforms input/output pairs into states. Figure 17 is an example: a transition system with a single state becomes a transition system with two states, $\{s_1, s_0\}$. We compute this feedback by transforming each transition $(s_i, i/s_o)$ into a transition $(i/)$ from s_i to s_o.

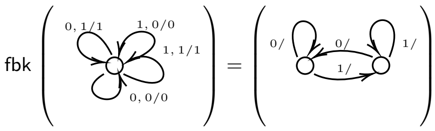


Fig. 17: Feedback of a Mealy transition system. Every transition has a label i/o indicating inputs (i) and outputs (o).

Example 3.16 (Elgot automata). Similarly, when we consider **Set** with the monoidal structure given by the disjoint union, we recover *Elgot automata* [18], which are given by a transition function $S + A \to S + B$. These transition systems motivate the work of Katis, Sabadini and Walters in [32,36].

Definition 3.17. *An* Elgot transition system *with initial states in A and final states in B is a tuple* $\mathbf{E} = (S, p, d)$ *where S is a set called the* state space*, $p\colon A \to S + B$ is a function called* initial step *and $d\colon S \to S + B$ is a function called* iterative step*.*

An Elgot transition system is interpreted as follows. We start by providing an initial state A. We then compute the initial step $p(a)$ which can result either in an internal state $p(a) = s \in S$ or in a final state $p(a) = b \in B$. In the later case, we are done and we return $b \in B$; in the former case, we repeatedly apply the iterative step: $d(p(a)), d(d(p(a))), \ldots$ until we reach a final state.

Example 3.18 (Linear dynamical systems). A *linear dynamical system* with inputs in $\mathbb{R}^n$, outputs in $\mathbb{R}^m$ and state space $\mathbb{R}^k$ is given by a number k, representing the dimension of the state space, and a matrix over the real numbers [31]

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \in \mathbf{Mat}(k+m, k+n).$$

Two linear dynamical systems,

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \text{ and } \begin{pmatrix} A' & B' \\ C' & D \end{pmatrix},$$

are considered equivalent if there is an invertible matrix $H \in \mathbf{Mat}(k, k)$ such that $A' = H^{-1}AH$, $B' = BH$, and $C' = H^{-1}C$.

Linear dynamical systems are morphisms of a feedback category which coincides with $\mathsf{St}(\mathbf{Mat})$, the free feedback category over the category of matrices **Mat** as defined in Example 2.3. The feedback operator is defined by

$$\mathsf{fbk}_l \left(k, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix} \right) = \left(k+l, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix} \right),$$

where $\left(\begin{smallmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{smallmatrix}\right) \in \mathbf{Mat}(k+l+m, k+l+n)$.

4 Span(Graph): an Algebra of Transition Systems

Span(**Graph**) [33] is an algebra of "open transition systems". It has applications in *concurrency theory* and *verification* [32,33,35,37,23], and has been recently applied to biological systems [21,22]. Just as ordinary Petri nets have an underlying (firing) semantics in terms of transition systems, **Span**(**Graph**) is used as a semantic universe for a variant of open Petri nets, see [53,9].

An *open transition system* is a morphism of **Span**(**Graph**): a transition graph endowed with two *boundaries* or *communication ports*. Each transition has an effect on each boundary, and this data is used for synchronization. This conceptual picture actually describes a subcategory, **Span**(**Graph**)$_*$, where boundaries are mere sets: the alphabets of synchronization signals. We shall recall the details of **Span**(**Graph**)$_*$ and prove that it is universal, our main result:

Span(**Graph**)$_*$ is the free feedback category over **Span**(**Set**).

4.1 The Algebra of Spans

Definition 4.1. *A span [4,10] from A to B, both objects of a category* **C**, *is a pair of morphisms with a common domain,*

$$A \xleftarrow{f} E \xrightarrow{g} B.$$

The object E is the "head" of the span, and the morphisms $f\colon E \to A$ and $g\colon E \to B$ are the left and right "legs", respectively.

When the category **C** has pullbacks, we can sequentially compose two spans $A \leftarrow E \to B$ and $B \leftarrow F \to C$ obtaining $A \leftarrow E \times_B F \to C$. Here, $E \times_B F$ is the pullback of E and F along B: for instance, in **Set**, $E \times_B F$ is the subset of $E \times F$ given by pairs that have the same image in B.

Remark 4.2 (Notation for spans). We denote a span $A \overset{f}{\leftarrow} X \overset{g}{\to} B$ in **C** as

$$\{f(x); g(x)\}_{x \in X} \in \mathbf{Span}(A, B),$$

where $x\colon U \to X$, for some object U of **C**, can be thought of as some generalized element that we compose with the two legs: e.g. in the category of sets, when $U = 1$, elements of a set X can be seen as functions $x\colon 1 \to X$. Sometimes, these generalized elements will come with conditions that must be listed with the morphism set. For instance, in Figure 18, a composition of spans has a pullback as its head, so any generalized element of its head is now a pair of morphisms $x\colon U \to X$ and $y\colon U \to Y$ satisfying the extra condition $g(x) = h(y)$:

$$\{f(x); g(x)\}_x \fatsemi \{h(y); k(y)\}_y = \{f(x); k(y)\}_{x,y}^{g(x)=h(y)}.$$

$$\begin{tikzcd}
 & & X \times_B Y \arrow[dl, "\pi_X"'] \arrow[dr, "\pi_Y"] & & \\
 & X \arrow[dl, "f"'] \arrow[dr, "g"] & & Y \arrow[dl, "h"'] \arrow[dr, "k"] & \\
A & & B & & C
\end{tikzcd}$$

Fig. 18: Composition of spans.

In other words, we are saying that the set of generalized elements of the head of the span is $\{x, y \mid g(x) = h(y)\}$. The advantage of this notation is that we can reason in any category with finite limits as we do in the category of sets: using *elements*. Whenever two sets of generalized elements of the head of a span are isomorphic, the Yoneda lemma [40] provides an isomorphism between the heads. That isomorphism makes the two spans equivalent when it commutes with the two legs.

Definition 4.3. *Let* **C** *be a category with pullbacks.* **Span**(**C**) *is the category that has the same objects as* **C** *and isomorphism classes of spans between them as morphisms. That is, two spans are considered* equal *if there is an isomorphism between their heads that commutes with both legs. Dually, if* **C** *is a category with pushouts,* **Cospan**(**C**) *is the category* **Span**($\mathbf{C}^{op}$).

Span(**C**) is a symmetric monoidal category when **C** has products. The parallel composition of $\{f_1(x); g_1(x)\}_{x\in X} \in \mathbf{Span}(A_1, B_1)$ and $\{f_2(y); g_2(y)\}_{y\in Y} \in \mathbf{Span}(A_2, B_2)$ is given by the componentwise product

$$\{(f_1(x), f_2(y)); (g_1(x), g_2(y))\}_{x\in X, y\in Y} \in \mathbf{Span}(A_1 \times A_2, B_1 \times B_2).$$

An example is again **Span**(**Set**).

Remark 4.4 (Variable change). We will be considering spans “up to isomorphism of their head”. This means that, given any isomorphism $\phi\colon X \to Y$, the following two spans are considered equal

$$\{f(\phi(x)); g(\phi(x))\}_{x\in X} = \{f(y); g(y)\}_{y\in Y}.$$

Moreover, if two spans are equal, then such a variable change does necessarily exist.

Example 4.5. Let us now detail some useful constants of the algebra of **Span**(**C**), which we will use to construct the NOR latch circuit from Figure 10.

The Frobenius algebra [10] (−<, >−, ●−, −●) is used for the “wiring”. The following spans are constructed out of diagonals $A \to A \times A$ and units $A \to 1$.

(−<)$_A = \{a; (a, a)\}_{a\in A} \in \mathbf{Span}(A, A \times A)$ (−●)$_A = \{a; *\}_a \in \mathbf{Span}(A, 1)$

(>−)$_A = \{(a, a); a\}_{a\in A} \in \mathbf{Span}(A \times A, A)$ (●−)$_A = \{*; a\}_a \in \mathbf{Span}(1, A)$

These induce a compact closed structure (and thus a trace), as follows:

$$(\bullet\!\!-\!\!\subset)_A = \{*; (a,a)\}_{a\in A} \in \mathbf{Span}(1, A\times A)$$
$$(\supset\!\!-\!\!\bullet)_A = \{(a,a); *\}_{a\in A} \in \mathbf{Span}(A\times A, 1).$$

Finally, we have a braiding making the category symmetric,

$$(\times) = \{(a,b); (b,a)\}_{a\in A, b\in B} \in \mathbf{Span}(A\times B, B\times A).$$

In general, any function $f\colon A\to B$ can be lifted to a span $\{a; f(a)\}_{a\in A} \in \mathbf{Span}(A,B)$ covariantly, and to a span $\{f(a); a\}_{a\in A} \in \mathbf{Span}(B,A)$, contravariantly.

4.2 The Algebra of Open Transition Systems

Definition 4.6. *The category* **Graph** *has graphs $G = (s,t\colon E \rightrightarrows V)$ as objects, i.e. pairs of morphisms from* edges *to* vertices *returning the* source *and* target *of each edge. A morphism of graphs $(e,v)\colon G\to G'$ is given by functions $e\colon E\to E'$ and $v\colon V\to V'$ such that $e\,⨾\,s' = s\,⨾\,v$ and $e\,⨾\,t' = t\,⨾\,v$ (see Figure 19)*[7]*.*

$$\begin{tikzcd}
E \arrow[r, "e"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & E' \arrow[d, bend right, "s'"'] \arrow[d, bend left, "t'"] \\
V \arrow[r, "v"] & V'
\end{tikzcd}$$

Fig. 19: Morphism of graphs.

We now focus on $\mathbf{Span}(\mathbf{Graph})_*$, those spans of graphs that have single vertex graphs $(A \rightrightarrows 1)$ as the boundaries.

Definition 4.7. *An* open transition system *is a morphism of $\mathbf{Span}(\mathbf{Graph})_*$: a span of sets $\{f(e); g(e)\}_{e\in E} \in \mathbf{Span}(A,B)$ where the head is the set of edges of a graph $s,t\colon E \rightrightarrows V$, i.e. the transitions (see Figure 20). Two open transition systems are considered* equal *if there is an isomorphism between their graphs that commutes with the legs. Open transition systems whose graph $E \rightrightarrows 1$ has a single vertex are called* stateless.

$$\begin{tikzcd}
A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & B \arrow[d, bend right] \arrow[d, bend left] \\
1 & V \arrow[l] \arrow[r] & 1
\end{tikzcd}$$

Fig. 20: A morphism of $\mathbf{Span}(\mathbf{Graph})_*$.

[7] Equivalently, **Graph** is the presheaf category on the diagram $(\bullet \rightrightarrows \bullet)$, i.e. the category of functors $(\bullet \rightrightarrows \bullet) \to \mathbf{Set}$ and natural transformations between them.

Sequential composition (the *communicating-parallel operation* of [33]) of two open transition systems with spans

$$\{f(e); g(e)\}_{e\in E} \in \mathbf{Span}(A, B) \text{ and } \{h(e'); k(e')\}_{e'\in E'} \in \mathbf{Span}(B, C)$$

and graphs $s, t\colon E \rightrightarrows S$ and $s', t'\colon E' \rightrightarrows S'$ yields the open transition system with the composite span

$$\{f(e); k(e')\}^{g(e)=h(e')}_{(e,e')\in E\times E'} \in \mathbf{Span}(A, C)$$

and graph $(s \times s', t \times t')\colon E \times_B E' \rightrightarrows S \times S'$. This means that the only allowed transitions are those that synchronize E and E' on the common boundary B.

Parallel composition (the *non communicating-parallel operation* of [33]) of two open transition systems with spans

$$\{f(e); g(e)\}_{e\in E} \in \mathbf{Span}(A, B) \text{ and } \{f'(e'); g'(e')\}_{e'\in E'} \in \mathbf{Span}(A', B')$$

and graphs $s, t\colon E \rightrightarrows V$ and $s', t'\colon E' \rightrightarrows V'$ yields the open transition system with span

$$\{(f(e), f'(e')); (g(e), g(e'))\}_{e,e'\in E\times E'} \in \mathbf{Span}(A \times A', B \times B')$$

and graph $(s \times s', t \times t')\colon E \times E' \rightrightarrows V \times V'$.

Remark 4.8 (Components of $\mathbf{Span}(\mathbf{Graph})_*$*).* Any span in $\mathbf{Span}(A, B)$ can be lifted to $\mathbf{Span}(\mathbf{Graph})_*(A, B)$ by making the head represent the graph $E \rightrightarrows 1$. Apart from the components lifted from $\mathbf{Span}$, which we call *stateless*, we will need to add a single stateful component to model all of $\mathbf{Span}(\mathbf{Graph})_*$: the delay in Figure 21.

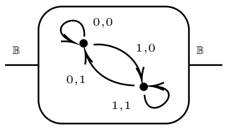


Fig. 21: Delay morphism over the set $\mathbb{B} := \{0, 1\}$.

The delay $(\text{-}\boxed{\partial}\text{-}_A)$ on a given set A is given by the span $\{a_2; a_1\}_{a_1,a_2\in A\times A} \in \mathbf{Span}(A, A)$ together with the graph $\pi_1, \pi_2\colon A \times A \to A$. This is to say that the delay receives on the left what the target of its transition (its next state) will be, while signalling on the right what the source of its transition (its current state) is. This is *not* an arbitrary choice: it is defined as the canonical delay obtained from the feedback structure in $\mathbf{Span}(\mathbf{Graph})_*$ (as in Section 3, $\partial_A = \mathsf{fbk}(\sigma_{A,A})$).

$$(\text{-}\boxed{\partial}\text{-})_A = \mathsf{fbk}(\{(a_1, a_2); (a_2, a_1)\}_{a_1,a_2}).$$

We can use this delay to correctly model a stateful NOR latch from the function $\mathtt{NOR} \colon \mathbb{B} \times \mathbb{B} \to \mathbb{B}$ (as we saw in Figure 2).

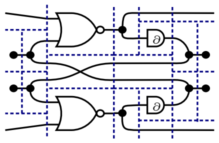


Fig. 22: Decomposing the circuit.

The NOR latch circuit of Figure 10 is the composition of two NOR gates where the outputs of each gate have been copied and fed back as input to the other gate. The algebraic expression, in **Span**(**Graph**)$_*$, of this circuit is obtained by decomposing it into its components, as in Figure 22.

$$\begin{aligned}(\mathrm{id} \otimes \bullet\!\!\prec \otimes \bullet\!\!\prec \otimes \mathrm{id}) \mathbin{⨟} (\mathtt{NOR} \otimes \sigma \otimes \mathtt{NOR}) \mathbin{⨟} (-\!\!\prec \otimes \mathrm{id} \otimes -\!\!\prec)& \\ \mathbin{⨟} (\mathrm{id} \otimes \partial \otimes \mathrm{id} \otimes \partial \otimes \mathrm{id}) \mathbin{⨟} (\mathrm{id} \otimes \succ\!\!\bullet \otimes \succ\!\!\bullet \otimes \mathrm{id})&\end{aligned}$$

The graph obtained from computing this expression, together with its transitions, is shown in Figure 23. This time, our model is indeed stateful. It has four states: two states representing a correctly stored signal, $\overline{\mathsf{A}} = (1, 0)$ and $\mathsf{A} = (0, 1)$; and two states representing transitory configurations $\mathsf{T}_1 = (0, 0)$ and $\mathsf{T}_2 = (1, 1)$.

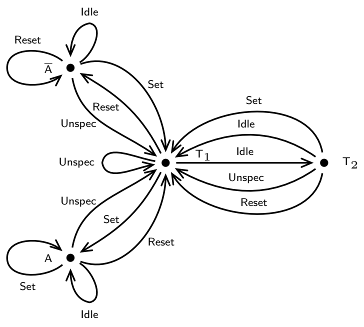


Fig. 23: Span of graphs representing the NOR latch

The *left boundary* can receive a *set* signal, $\mathsf{Set} = \binom{1}{0}$; a *reset* signal, $\mathsf{Reset} = \binom{0}{1}$; none of the two, $\mathsf{Idle} = \binom{0}{0}$; or both of them at the same time, $\mathsf{Unspec} = \binom{1}{1}$, which is known to cause unspecified behavior in a NOR latch. The signal on the *right boundary*, on the other hand, is always equal to the state the transition goes to and does not provide any additional information: we omit it from Figure 23.

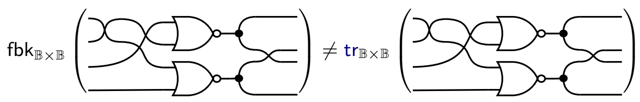


Fig. 24: Applying $\mathsf{fbk}(\bullet)$ over the circuit gives the NOR latch.

Activating the signal Set makes the latch reach the state A in (at most) two transition steps. Activating Reset does the same for $\overline{\mathsf{A}}$. After any of these two cases, deactivating all signals, Idle, keeps the last state.

Moreover, the (real-world) NOR latch has some unspecified behavior that gets also reflected in the graph: activating both Set and Reset at the same time, what we call Unspec, causes the circuit to enter an unstable state where it bounces between the states T_1 and T_2 after an Idle signal. Our modeling has reflected this "unspecified behavior" as expected.

Feedback and trace. In terms of feedback, the circuit of Figure 23 is equivalently obtained as the result of taking feedback over the stateless morphism in Figure 24.

But $\mathbf{Span}(\mathbf{Graph})_*$ is also canonically traced: it is actually compact closed. What changes in the modeling if we would have used the trace instead? As we argued for Figure 8, we obtain a stateless transition system. The valid transitions are

$$\{(\mathsf{Unspec}, \mathsf{T}_1), (\mathsf{Idle}, \mathsf{A}), (\mathsf{Idle}, \overline{\mathsf{A}}), (\mathsf{Set}, \mathsf{A}), (\mathsf{Reset}, \overline{\mathsf{A}})\}.$$

They encode important information: they are the *equilibrium* states of the circuit. However, unlike the previous graph, this one would not get us the correct allowed transitions: under this modeling, our circuit could freely bounce between $(\mathsf{Idle}, \mathsf{A})$ and $(\mathsf{Idle}, \overline{\mathsf{A}})$, which is not the expected behavior of a NOR latch.

The fundamental piece making our modeling succeed the previous time was feedback with *delay*. Next we show that this feedback is canonical.

4.3 Span(Graph) as a Feedback Category

This section presents our main theorem. We introduce a mapping that associates to each stateful span of sets a corresponding span of graphs. This mapping is well-defined and lifts to a functor $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$. Finally, we prove that it is an isomorphism $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*$.

First of all, we need to be able to explicitly compute the composition of stateful spans, following the composition of stateful morphisms in Definition 3.9. This is Proposition 4.9. Then, we will characterize isomorphisms in the category of spans in Proposition 4.10.

Proposition 4.9. *Let* $\mathbf{C}$ *be a category with finite limits. Consider two stateful spans in the category* $\mathsf{St}(\mathbf{Span}(\mathbf{C}))$,

$$\{(\sigma(x), f(x)); (\sigma'(x), g(x))\}_{x \in X} \in \mathbf{Span}(S \times A, S \times B),$$

$$\{(\tau(y), h(y)); (\tau'(y), k(y))\}_{y \in Y} \in \mathbf{Span}(T \times B, T \times C).$$

their composition is then given by the span

$$\{(\sigma(x), \tau(y), f(x)); (\sigma'(x), \tau'(y), k(y))\}^{g(x)=h(y)}_{(x,y)\in X\times Y} \in \mathbf{Span}(S \times T \times A, S \times T \times B),$$

where the head is $X \times_B Y$, the pullback of g and h.

Proof. Using the notation for spans, we apply the definition of sequential composition in a category of stateful processes (Definition 3.9).

$$\begin{aligned}
&(\{(s,t);(t,s)\}_{s,t} \otimes \{a;a\}_a) \fatsemi (\{t;t\}_t \otimes \{(\sigma(x), f(x)); (\sigma'(x), g(x))\}_x) \\
&\qquad \fatsemi (\{(t,s);(s,t)\}_{s,t} \otimes \{b;b\}_b) \fatsemi (\{s;s\}_s \otimes \{(\tau(y), h(y)); (\tau'(y), k(y))\}_y) \\
= \quad & \textit{(Computing tensors)} \\
&\{(s,t,a);(t,s,a)\}_{s,t,a} \fatsemi \{(t,\sigma(x), f(x)); (t,\sigma'(x), g(x))\}_{t,x} \\
&\qquad \fatsemi \{(t,s,b);(s,t,b)\}_{s,t,b} \fatsemi \{(s,\tau(y), h(y)); (s,\tau'(y), k(y))\}_{s,y} \\
= \quad & \textit{(Computing compositions)} \\
&\{(\sigma(x), t, f(x)); (t, \sigma'(x), g(x))\}_{t,x} \fatsemi \{(\tau(y), s, h(y)); (s, \tau'(y), k(y))\}_{s,y} \\
= \quad & \textit{(Computing compositions)} \\
&\{(\sigma(x), \tau(y), f(x)); (\sigma'(x), \tau'(y), k(y))\}^{g(x)=h(y)}_{x,y}.
\end{aligned}$$

This last formula corresponds indeed to the pullback we stated. □

Proposition 4.10. *Let $\mathbf{C}$ be a category with all finite limits. An isomorphism $A \cong B$ in its category of spans, $\mathbf{Span}(\mathbf{C})$, is always of the form*

$$\{a; \phi(a)\}_{a\in A} \in \mathbf{Span}(\mathbf{C})(A, B),$$

where the left leg is an identity and the right leg $\phi \colon A \to B$ is an isomorphism in the base category $\mathbf{C}$.

Proof. Let $\{f(x); g(x)\}_{x\in X} \in \mathbf{Span}(A, B)$ and $\{h(y); k(y)\}_{y\in Y} \in \mathbf{Span}(B, A)$ be mutual inverses. This means that

$$\{f(x); g(x)\}_{x\in X} \fatsemi \{h(y); k(y)\}_{y\in Y} = \{f(x); k(y)\}^{g(x)=h(y)}_{x,y} = \{a; a\}_{a\in A},$$

$$\{h(y); k(y)\}_{y\in Y} \fatsemi \{f(x); g(x)\}_{x\in X} = \{h(y); g(x)\}^{k(y)=f(x)}_{x,y} = \{b; b\}_{b\in B}.$$

In turn, this implies the existence of variable changes $(\alpha_X, \alpha_Y) \colon A \to X \times_B Y$ and $(\beta_Y, \beta_X) \colon B \to Y \times_A X$ such that they are the inverses of (f, k) and (g, h) respectively.

We can thus write the spans as having the identity on the left leg.

$$\{f(x); g(x)\}_{x\in X} = \{f(\alpha_X(a)); g(\alpha_X(a))\}_{a\in A} = \{a; g(\alpha_X(a))\}_{a\in A}.$$

$$\{h(y); k(y)\}_{y\in Y} = \{h(\beta_Y(b)); k(\beta_Y(b))\}_{b\in B} = \{b; k(\beta_Y(b))\}_{b\in B}.$$

Finally, composing them again, we get that $(g \fatsemi \alpha_X)$ and $(k \fatsemi \beta_Y)$ must be mutual inverses, thus isomorphisms. □

We are now ready to prove the main result. The following Lemma 4.11 proves that we can translate stateful spans to spans of graphs. The main Theorem 4.12 follows from it.

Lemma 4.11. *Let* **C** *be a category with all finite limits. The following assignment of stateful processes over* **Span**(**C**) *to morphisms of* **Span**(**Graph**(**C**)) *is well-defined.*

$$K\left(S \;\middle|\; \begin{tikzcd}[ampersand replacement=\&] \& E \arrow[dl, "{(s,f)}"'] \arrow[dr, "{(t,g)}"] \& \\ S \times A \& \& S \times B \end{tikzcd} \right) := \left(\begin{tikzcd}[ampersand replacement=\&] A \arrow[d, shift right] \arrow[d, shift left] \& E \arrow[l, "f"'] \arrow[r, "g"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] \& B \arrow[d, shift right] \arrow[d, shift left] \\ 1 \& S \arrow[l] \arrow[r] \& 1 \end{tikzcd} \right)$$

Proof. We first check that two *isomorphic* spans are sent to *isomorphic* spans of graphs. Let

$$\{(s(e), f(e)); (t(e), g(e))\}_{e \in E} \in \mathbf{Span}(S \times A, S \times B) \text{ and}$$
$$\{(s'(e'), f'(e')); (t'(e'), g'(e'))\}_{e' \in E'} \in \mathbf{Span}(S \times A, S \times B)$$

be two spans that are isomorphic with the variable change $\phi\colon E \cong E'$. Then, (ϕ, id) is an isomorphism of spans of graphs, also making the relevant diagram commute (Figure 25).

$$\begin{tikzcd} & E \arrow[dl, "{(s,f)}"'] \arrow[dr, "{(t,g)}"] \arrow[dd, "\phi"] & \\ S \times A & & S \times B \\ & E' \arrow[ul, "{(s',f')}"] \arrow[ur, "{(t',g')}"'] & \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d, shift right] \arrow[d, shift left] & E \arrow[l, "f"'] \arrow[r, "\phi"] \arrow[rr, bend left, "g"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & E' \arrow[ll, bend right, "f'"'] \arrow[r, "g'"] \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] & B \arrow[d, shift right] \arrow[d, shift left] \\ 1 & S \arrow[l] \arrow[r, "\mathrm{id}"] \arrow[rr, bend right] & S \arrow[ll, bend left] \arrow[r] & 1 \end{tikzcd}$$

Fig. 25: Isomorphic spans result in isomorphic spans of graphs.

We show now that the assignment preserves the equivalence relation of stateful processes. Isomorphisms in a category of spans are precisely spans whose two legs are isomorphisms (by Proposition 4.10, or the more general result of [44]). This means that an isomorphism in **Span**(**Set**) can be always rewritten as $\{s; \phi(s)\}_{s \in S} \in \mathbf{Span}(S, T)$, where the left leg is an identity and the right leg is $\phi\colon S \to T$, some isomorphism. Its inverse can be written analogously as $T \leftarrow S \to S$. In order to prove that the quotient relation induced by the feedback is preserved, we need to check that equivalent spans of sets are sent to isomorphic spans of graphs. If two spans are equivalent with the variable change $\phi\colon S \cong T$, then the corresponding graphs are isomorphic with the isomorphism of graphs (id, ϕ), see Figure 26.

$$\begin{tikzcd}
 & \cdot \arrow[dl, "\phi\times\mathrm{id}"'] \arrow[d, dash] & & E \arrow[dl, "{(s,a)}"'] \arrow[d, "{(t,b)}"] & & \cdot \arrow[dl, dash] \arrow[d, "\phi\times\mathrm{id}"] \\
T\times A & S\times A & & S\times B & & T\times B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "a"'] \arrow[r, "\mathrm{id}"] \arrow[rr, bend left, "b"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & E \arrow[r, "b"] \arrow[ll, bend right, "a"'] \arrow[d, bend right, "s\fatsemi\phi"'] \arrow[d, bend left, "t\fatsemi\phi"] & B \arrow[d, bend right] \arrow[d, bend left] \\
1 & S \arrow[l] \arrow[r, "h"] \arrow[rr, bend right] & T \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

Fig. 26: Equivalent spans result in isomorphic spans of graphs.

$\square$

Theorem 4.12. *There exists an isomorphism of categories* $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*$. *That is, the free feedback category over* $\mathbf{Span}(\mathbf{Set})$ *is isomorphic to the full subcategory of* $\mathbf{Span}(\mathbf{Graph})$ *given by single-vertex graphs.*

Proof. We prove that there is a fully faithful functor $K\colon \mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$ defined on objects as $K(A) = (A \rightrightarrows 1)$ and defined on morphisms as in Lemma 4.11.

We now show that K is functorial, preserving composition and identities. The identity morphism on A in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ has state space 1, so it is a span $1 \times A \leftarrow A \to 1 \times A$ and it is sent to the identity span on the graph $A \rightrightarrows 1$.

Composition is also preserved. Let us consider two stateful spans

$$\{(s(e), a(e)); (s'(e), b(e))\}_{e\in E} \in \mathbf{Span}(S \times A, S \times B) \text{ and}$$
$$\{(t(f), b'(f)); (t'(f), c(f))\}_{f\in F} \in \mathbf{Span}(S \times B, S \times C)$$

By Proposition 4.9, their composition is given by the span

$$\{(s(e), t(f), a(e)); (s'(e), t'(f), c(f))\}^{b(e)=b'(f)}_{(e,f)\in E\times F} \in \mathbf{Span}(S \times T \times A, S \times T \times C)$$

where the head $E \times_B F$ is the pullback of b and b'.

$$\begin{tikzcd}
 & & E\times_B F \arrow[dl, "\pi_E"'] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr, "\pi_F"] & & \\
 & E \arrow[dl, "a"'] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr, "b"'] & S\times T \arrow[dl, "\pi_S"] \arrow[dr, "\pi_T"'] & F \arrow[dl, "b'"] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr, "c"] & \\
A \arrow[d, shift left] \arrow[d, shift right] & S \arrow[dl] \arrow[dr] & B \arrow[d, shift left] \arrow[d, shift right] & T \arrow[dl] \arrow[dr] & C \arrow[d, shift left] \arrow[d, shift right] \\
1 & & 1 & & 1
\end{tikzcd}$$

Fig. 27: Pullback of graphs.

We have composed two stateful spans and we want to show that the graph corresponding to their composition is the pullback of the graphs corresponding to them. Computing a pullback of graphs can be done separately on edges and vertices, as graphs form a presheaf category (see Figure 27). Note how the resulting graph is precisely the graph corresponding, under the assignment K, to the stateful span computed above.

We have shown that K is a functor. The final step is to show that it is fully-faithful. We can see that it is full: every span of single-vertex graphs given by $\{f(e); g(e)\}_{e\in E} \in \mathbf{Span}(A, B)$ and $s, t \colon E \rightrightarrows S$ is the image of some span, namely

$$\{s(e), f(e); t(e), g(e)\}_{e\in E} \in \mathbf{Span}(S \times A, S \times B).$$

Let us check it is also faithful. Suppose that two morphisms in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$, $S \times A \leftarrow E \rightarrow S \times B$ and $S' \times A \leftarrow E' \rightarrow S' \times B$, are sent to equivalent spans of graphs, i.e. there exist $h \colon E \cong E'$ and $k \colon S' \cong S$ making the diagrams in Figure 28 commute.

$$\begin{tikzcd}
A \arrow[d, shift right] \arrow[d, shift left] & E \arrow[l, "a"'] \arrow[r, "h"] \arrow[rr, bend left, "b"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & E' \arrow[r, "b'"] \arrow[ll, bend right, "a'"'] \arrow[d, shift right, "s'"'] \arrow[d, shift left, "t'"] & B \arrow[d, shift right] \arrow[d, shift left] \\
1 & S \arrow[l] \arrow[r, "k"] \arrow[rr, bend right] & S' \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

Fig. 28: Equivalent spans of graphs.

The isomorphism k makes the following spans equivalent as stateful processes.

$$S \times A \leftarrow E \rightarrow S \times B$$
$$S' \times A \leftarrow E \rightarrow S' \times B$$

Moreover, the isomorphism h makes the following spans equivalent as spans, showing faithfullness of K.

$$S' \times A \leftarrow E \rightarrow S' \times B$$
$$S' \times A \leftarrow E' \rightarrow S' \times B$$

We have shown that there exists a fully-faithful functor from the free feedback category over $\mathbf{Span}(\mathbf{Set})$ to the category $\mathbf{Span}(\mathbf{Graph})$ of spans of graphs. The functor restricts to an equivalence between $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ and the full subcategory of $\mathbf{Span}(\mathbf{Graph})$ on single-vertex graphs. It is moreover bijective on objects, giving an isomorphism of categories. □

Example 4.13. The characterization $\mathbf{Span}(\mathbf{Graph})_* \cong \mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ that we prove in Theorem 4.12 lifts the inclusion $\mathbf{Set} \rightarrow \mathbf{Span}(\mathbf{Set})$ to a feedback preserving functor $\mathbf{Mealy} \rightarrow \mathbf{Span}(\mathbf{Graph})_*$. This inclusion embeds a deterministic transition system into the graph that determines it.

Example 4.14. Following from Remark 3.12, we present an example of spans of graphs that would be equated if we assumed the sliding axiom (A5) of feedback categories for arbitrary morphisms rather than just isomorphisms. Consider the spans of sets $\alpha \colon W \to V$ and $h \colon V \to W \times \mathbb{B}$ as in Figure 29, where $V = \{v_1, v_2\}$ and $W = \{w\}$. Depending on where the feedback operation is applied, we obtain two different spans of graphs, $g = \mathsf{fbk}_V(h ⨟ (\alpha \otimes \mathrm{id}))$ and $f = \mathsf{fbk}_W((\alpha \otimes \mathrm{id}) ⨟ h)$: the first one contains an additional transition. If we were to impose that the sliding axiom holds for non-isomorphisms, we could erase this additional transition, and obtain that $f = g$ by sliding α through the feedback loop.

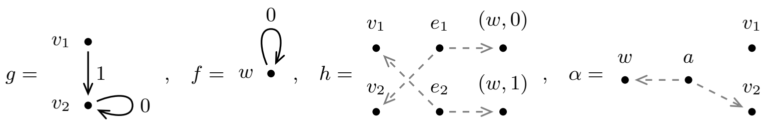


Fig. 29: Spans of graphs that would be equated by a stronger notion of equivalence: $g = \mathsf{fbk}(h ⨟ (\alpha \otimes \mathrm{id})) \sim \mathsf{fbk}((\alpha \otimes \mathrm{id}) ⨟ h) = f$.

4.4 Cospan(Graph) as a Feedback Category

Theorem 4.12 can be generalized to any category $\mathbf{C}$ with finite limits, where we can define graphs and spans of them.

A graph in a category $\mathbf{C}$ is given by two objects, E and V, and two morphisms in $\mathbf{C}$, the source and the target $s, t \colon E \to V$. A morphism of graphs $\alpha \colon G \to G'$ in $\mathbf{C}$ is a pair of morphisms, $\alpha_E \colon E \to E'$ and $\alpha_V \colon V \to V'$, in $\mathbf{C}$ that commute with the sources and the targets. In categorical terms, these can be reformulated as functors and natural transformations.

Definition 4.15. *Let $\mathbf{C}$ be a category with finite limits. A graph in $\mathbf{C}$ is a functor from the diagram $(\bullet \rightrightarrows \bullet)$ to $\mathbf{C}$. A morphism of graphs in $\mathbf{C}$ is a natural transformation between the corresponding functors. Graphs in $\mathbf{C}$ form a category $\mathbf{Graph}(\mathbf{C})$.*

Categories of functors into $\mathbf{C}$ have all the limits that $\mathbf{C}$ has [40]. We can then form the category $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))$ and take its full subcategory on objects of the form $A \rightrightarrows 1$, i.e. $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$, to obtain:

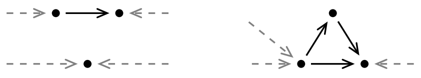
Theorem 4.16. *There exists an isomorphism of categories $\mathsf{St}(\mathbf{Span}(\mathbf{C})) \cong \mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$. That is, the free feedback category over $\mathbf{Span}(\mathbf{C})$ is equivalent to the full subcategory on $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))$ given by single-vertex graphs.*

$\mathbf{Cospan}(\mathbf{Graph})_*$ is the dual algebra to $\mathbf{Span}(\mathbf{Graph})_*$. Its morphisms represent graphs with discrete boundaries: while, in $\mathbf{Span}(\mathbf{Graph})_*$, each transition in the graph is assigned a boundary behavior, a morphism in $\mathbf{Cospan}(\mathbf{Graph})_*$

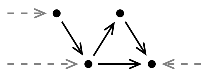
is a graph where some vertices are marked as left boundary or right boundary vertices. This allows graphs to be composed by identifying these boundary vertices.

Definition 4.17. *A* graph with discrete boundaries $g \colon X \to Y$ *is given by a graph* $G = (s, t \colon E \rightrightarrows V)$ *and two functions,* $l \colon X \to V$ *and* $r \colon Y \to V$*, marking the boundary vertices.*

Example 4.18. We represent the legs of a cospan as dashed arrows pointing to some vertices of the apex graph.



The composition of the above cospans of graphs is given by


,

where the vertices in the common boundary have been identified.

Cospan(**Graph**)$_*$ can be also characterized as a free feedback category. We know that **Cospan**(**Set**) $\cong$ **Span**(**Set**^{op}), we note that **Graph**(**Set**^{op}) $\cong$ **Graph**^{op}(**Set**) (which has the effect of flipping edges and vertices), and we can use Theorem 4.16 because **Set** has all finite colimits. The explicit assignment is similar to the one shown in Lemma 4.11.

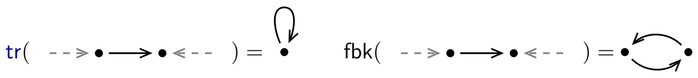
$$K\left(S \;\middle|\; \begin{tikzcd} & S & \\ E+A \arrow[ur, "{[t|a]}"] & & E+B \arrow[ul, "{[s|b]}"'] \end{tikzcd} \right) := \left(\begin{tikzcd} A \arrow[r, "a"] & S & B \arrow[l, "b"'] \\ 0 \arrow[r] \arrow[u, bend left] \arrow[u, bend right] & E \arrow[u, bend left, "t"] \arrow[u, bend right, "s"'] & 0 \arrow[l] \arrow[u, bend left] \arrow[u, bend right] \end{tikzcd} \right)$$

Corollary 4.19. *There is an isomorphism*

$$\mathsf{St}(\mathbf{Cospan}(\mathbf{Set})) \cong \mathbf{Cospan}(\mathbf{Graph})_*.$$

Cospan(**Graph**) is also compact closed and, in particular, traced. As in the case of **Span**(**Graph**), the feedback structure given by the universal property is different from the trace. In the case of **Cospan**(**Graph**), tracing has the effect of identifying the output and input vertices of the graph; while feedback adds an additional edge from the output to the input vertices.

Example 4.20. Tracing the cospan of a one-edge graph identifies the two vertices making it into a self-loop. On the other hand, taking feedback of the same cospan has the effect of adding another edge from the right boundary to the left one.



4.5 Syntactical Presentation of Cospan(FinGraph)

The observation in Proposition 3.7 has an important consequence in the case of finite sets. We write **FinGraph** for **Graph**(**FinSet**). **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [38], meaning that any morphism written out of the operations of a special commutative Frobenius monoid and the structure of a symmetric monoidal category is precisely a cospan of finite sets. Figure 30 represents the generators and the axioms of the generic special commutative Frobenius monoid.

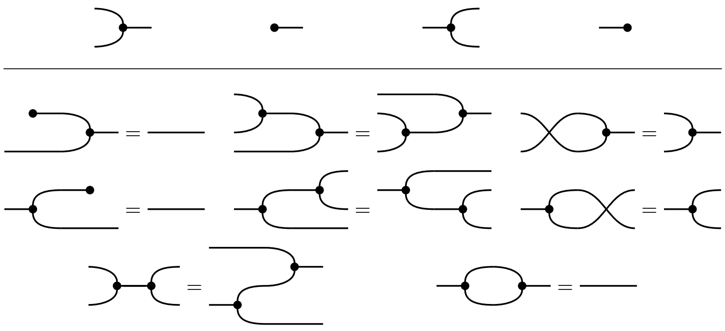


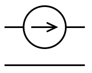
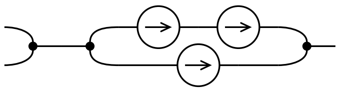
Fig. 30: Generators and axioms of the generic special commutative Frobenius monoid.

But we also know that **Cospan**(**FinSet**), with an added generator to its PROP structure [7] is St(**Cospan**(**FinSet**)), or, equivalently, **Cospan**(**FinGraph**). This means that any morphism written out of the operations of a special commutative Frobenius algebra plus a freely added generator of type $(-\boxed{\partial}-) \colon 1 \to 1$ is a morphism in **Cospan**(**FinGraph**)$_*$. This way, we recover one of the main results of [48] as a direct corollary of our characterization.

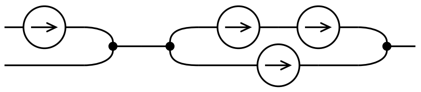
Proposition 4.21 (Proposition 3.2 of [48]). **Cospan**(**FinGraph**)$_*$ *is the generic special commutative Frobenius monoid with an added generator.*

Proof. It is known that the category **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [38]. Adding a free generator $(-\boxed{\partial}-) \colon 1 \to 1$ to its PROP structure corresponds to adding a family $(-\boxed{\partial}-)_n \colon n \to n$ with the conditions on Proposition 3.7. Now, Proposition 3.7 implies that adding such a generator to **Cospan**(**FinSet**) results in St(**Cospan**(**FinSet**)). Finally, we use Theorem 4.12 to conclude that St(**Cospan**(**FinSet**)) $\cong$ **Cospan**(**FinGraph**)$_*$. □

Example 4.22. The delay generator $-\boxed{\partial}- \colon 1 \to 1$ in **Cospan**(**FinGraph**)$_*$ can be interpreted as a single edge. Thus, we draw it as $-\!\!\bigcirc\!\!\!\!\!\!\rightarrow\!\!-$ $: 1 \to 1$. The cospans of graphs in Example 4.18 are represented by the string diagrams

 and  .

Their composition is

 .

5 Structured state spaces

This section extends the framework of feedback categories from mere transition systems to automata with initial and final states. In order to achieve this, we generalize the feedback construction to deal with a richer structure. The key ingredient in the generalization of feedback categories is a close examination of the sliding axiom: deciding which processes can be "slid" determines the notion of equality we want to apply.

5.1 Structured Feedback Categories

In order to capture automata, the state space S needs to be equipped with "extra structure". This is achieved by letting the state space live in a different category $\mathbf{S}$ from the base category $\mathbf{C}$ and by having a way of "forgetting" the extra structure it carries through a monoidal functor $\mathsf{R}\colon \mathbf{S} \to \mathbf{C}$.

A structured feedback operator on $\mathbf{C}$, then, takes a morphism acting on a structured state space (Figure 31).

$$\frac{f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B}{\mathsf{fbk}_S(f)\colon A \to B}$$

Fig. 31: Type of the operator $\mathsf{fbk}_S(\bullet)$.

Definition 5.1 (Structured feedback category). *A structured feedback category is a symmetric monoidal category $(\mathbf{C}, \otimes, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}})$ together with a symmetric monoidal category, $(\mathbf{S}, \boxtimes, J, \alpha^{\mathbf{S}}, \lambda^{\mathbf{S}}, \rho^{\mathbf{S}})$, representing structured state spaces, and a symmetric monoidal functor $(\mathsf{R}, \varepsilon, \mu)\colon \mathbf{S} \to \mathbf{C}$ endowed with an operator $\mathsf{fbk}_S\colon \mathbf{C}(\mathsf{R}S \otimes A, \mathsf{R}S \otimes B) \to \mathbf{C}(A, B)$, which satisfies the following axioms (B1-B5).*

(B1). Tightening. *For every $S \in \mathbf{S}$, every morphism $f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$ and every pair of morphisms $u\colon A' \to A$ and $v\colon B \to B'$,*

$$u \mathbin{⨟} \mathsf{fbk}_S(f) \mathbin{⨟} v = \mathsf{fbk}_S((\mathrm{id} \otimes u) \mathbin{⨟} f \mathbin{⨟} (\mathrm{id} \otimes v)).$$

(B2). Vanishing. *For every $f\colon A \to B$,*

$$\mathsf{fbk}_J((\varepsilon^{-1} \otimes \mathrm{id}) \mathbin{⨟} f \mathbin{⨟} (\varepsilon \otimes \mathrm{id})) = f.$$

(B3). Joining. *For every $S, T \in \mathbf{S}$ and every morphism $f\colon \mathsf{R}S \otimes \mathsf{R}T \otimes A \to \mathsf{R}S \otimes \mathsf{R}T \otimes B$,*

$$\mathsf{fbk}_T(\mathsf{fbk}_S(f)) = \mathsf{fbk}_{S \otimes T}((\mu^{-1} \otimes \mathrm{id}) \mathbin{⨟} f \mathbin{⨟} (\mu \otimes \mathrm{id})).$$

(B4). Strength. *For every $S \in \mathbf{S}$, every morphism $f\colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$ and every morphism $g\colon A' \to B'$,*

$$\mathsf{fbk}_S(f) \otimes g = \mathsf{fbk}_S(f \otimes g).$$

(B5). Sliding. For every $S, T \in \mathbf{S}$, every $f \colon \mathsf{R}T \otimes A \to \mathsf{R}S \otimes B$ and every $h \colon S \to T$ in $\mathbf{S}$,

$$\mathsf{fbk}_T(f \,⨾\, (\mathsf{R}h \otimes \mathrm{id})) = \mathsf{fbk}_S((\mathsf{R}h \otimes \mathrm{id}) \,⨾\, f).$$

Remark 5.2. The sliding axiom encodes the fact that applying a transformation to the state space $h \colon S \to T$ just before computing the next state should be essentially the same as applying the same transformation to the state space just before retrieving the current state. In the particular case where all transformations are asked to be reversible (i.e. isomorphisms)[8], this sliding axiom (B5) particularizes to the sliding axiom of plain feedback categories (A5).

Definition 5.3 (Structured feedback functor). *A structured feedback functor $(F, G) \colon \mathbf{C} \to \mathbf{C}'$ between two structured feedback categories $(\mathbf{C}, \mathbf{S}, \mathsf{R}, \mathsf{fbk})$ and $(\mathbf{C}', \mathbf{S}', \mathsf{R}', \mathsf{fbk}')$ is a pair of symmetric monoidal functors, (F, ε, μ) and $(G, \varepsilon^G, \mu^G)$, with types $F \colon \mathbf{C} \to \mathbf{C}'$ and $G \colon \mathbf{S} \to \mathbf{S}'$ such that $R \,⨾\, F = G \,⨾\, R'$ and*

$$F(\mathsf{fbk}_S(f)) = \mathsf{fbk}'_{GS}(\mu \,⨾\, Ff \,⨾\, \mu^{-1}).$$

We write $\mathsf{SFeedback}$ *for the category of (small) structured feedback categories and structured feedback functors. There is a forgetful functor $\mathcal{U} \colon \mathsf{SFeedback} \to \mathsf{SymMon}$.*

5.2 Structured St(•) Construction

In the same way that the free feedback category was realized by stateful processes, the free structured feedback category is realized by stateful processes with a structured state space $S \in \mathbf{S}$. A functor $\mathsf{R} \colon \mathbf{S} \to \mathbf{C}$ forgets the extra structure of this space.

Following the analogy, stateful processes with structured state space are pairs $(S \mid f)$ consisting of a structured state space $S \in \mathbf{S}$ and a morphism $f \colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$. We shall consider morphisms up to sliding of their state space, as in Figure 32.

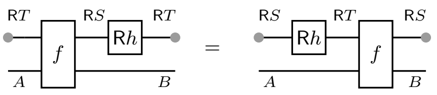


Fig. 32: Equivalence of structured stateful processes. We depict structured stateful processes by marking the state space.

Definition 5.4 (Category of structured stateful processes). *Consider a pair of symmetric monoidal categories $(\mathbf{C}, \otimes, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}})$ and $(\mathbf{S}, \boxtimes, J, \alpha^{\mathbf{S}}, \lambda^{\mathbf{S}}, \rho^{\mathbf{S}})$ and a symmetric monoidal functor $(\mathsf{R}, \varepsilon, \mu) \colon \mathbf{S} \to \mathbf{C}$. We write $\mathsf{St}(\mathbf{C}, \mathsf{R})$ for the*

[8] Here, $\mathbf{S}$ is the subcategory of isomorphisms of $\mathbf{C}$ and R is the inclusion functor.

category with the objects of $\mathbf{C}$ *but where morphisms* $A \to B$ *are pairs* $(S \mid f)$ *consisting of a state space* $S \in \mathbf{S}$ *and a morphism* $f \colon \mathsf{R}S \otimes A \to \mathsf{R}S \otimes B$*. We consider morphisms up to equivalence of their state spaces, where the equivalence relation is generated by*

$$(S \mid (\mathsf{R}h \otimes \mathrm{id}) ⨾ f) \sim_{\mathbf{S}} (T \mid f ⨾ (\mathsf{R}h \otimes \mathrm{id})) \text{ for any } h \colon S \to T.$$

Identities, composition, monoidal product and the feedback operator $\mathsf{store}(\bullet)$ *are defined in analogous ways as for stateful processes (Definition 3.9). When depicting a structured stateful process (Figure 32) we mark the state strings.*

Remark 5.5. In other words, structured stateful processes are elements of the following coproduct quotiented by the smallest equivalence relation $(\sim_{\mathbf{S}})$ satisfying sliding: $((\mathsf{R}h \otimes \mathrm{id}) ⨾ f) \sim_{\mathbf{S}} (f ⨾ (\mathsf{R}h \otimes \mathrm{id}))$.

$$\mathsf{St}(\mathbf{C}, \mathsf{R})(X, Y) \coloneqq \left(\sum_{S \in \mathbf{S}} \mathbf{C}(\mathsf{R}S \otimes A, \mathsf{R}S \otimes B) \right) / (\sim_{\mathbf{S}}).$$

This quotient is a particular form of colimit called a *coend* [40].

Repeating the proof from Katis, Sabadini and Walters [33] in this generalized setting, we can show that structured stateful processes form the free structured feedback category.

Theorem 5.6. *The category* $\mathsf{St}(\mathbf{C}, \mathsf{R})$*, endowed with the* $\mathsf{store}(\bullet)$ *operator, is the free structured feedback category over a symmetric monoidal category* $\mathbf{C}$ *with a symmetric monoidal functor* $\mathsf{R} \colon \mathbf{S} \to \mathbf{C}$*.*

5.3 Categories of Automata

Let us present classical automata as an example of the construction of structured feedback. Automata have a structured state space where a particular state is considered the "initial state", and a subset of states are considered "final". We can canonically recover a suitable category of automata as the free feedback category over these structured spaces.

Definition 5.7. *An* automaton state space (S, i_S, f_S) *is a finite set* S *together with an* initial state $i_S \in S$ *and a subset of final states* $f_S \colon S \to 2$*. The product of two automaton state spaces,* (S, i_S, f_S) *and* (T, i_T, f_T)*, is the state space* $(S \times T, (i_S, i_T), f_S \wedge f_T)$*, where the final states are pairs of final states,* $(f_S \wedge f_T)(s,t) = f_S(s) \wedge f_T(t)$*. A morphism of automaton state spaces* $\alpha \colon (S, i_S, f_S) \to (T, i_T, f_T)$ *is a function* $\alpha \colon S \to T$ *such that* $i_S ⨾ \alpha = i_T$ *and* $f_S = \alpha ⨾ f_T$*. Automaton state spaces form a symmetric monoidal category,* $\mathbf{AutSt}$*.*

Remark 5.8. As a consequence, an isomorphism of automata state spaces

$$\alpha \colon (S, i_S, f_S) \cong (T, i_T, f_T)$$

is an isomorphism $\alpha \colon S \cong T$ such that $i_S ⨾ \alpha = i_T$ and $f_S = \alpha ⨾ f_T$. These form a subcategory $\mathbf{Iso}(\mathbf{AutSt})$ with forgetful functors $\mathsf{U}_{\mathsf{Iso}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{FinSet}$ and $\mathsf{U}_{\mathsf{Aut}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{Span}(\mathbf{FinSet})$.

Definition 5.9. A Mealy deterministic finite automaton (S, A, B, i_S, f_S, t_S) *is given by a finite set of states S, a finite alphabet of input symbols A and a finite alphabet of output symbols B, an initial state $i_S \in S$, a set of final states $f_S \colon S \to 2$, and a transition function $t_S \colon S \times A \to S \times B$. The product of two deterministic finite automata,*

$$(S, A, B, i_S, f_S, t_S) \text{ and } (S', A', B', i_{S'}, f_{S'}, t_{S'}),$$

is the automaton $(S \times T, A, C, (i_S, i_{S'}), (f_S \wedge f_{S'}), (t_S \times t_{S'}))$, where the transition function computes a pair of independent transitions,

$$(t_S \times t_{S'})(s, s', a, a') = (t_S(s, a), t_{S'}(s', a')).$$

The sequential synchronization of two deterministic finite automata,

$$(S, A, B, i_S, f_S, t_S) \text{ and } (T, B, C, i_T, f_T, t_T),$$

is the automaton $(S \times T, A, C, (i_S, i_T), (f_S \wedge f_T), (t_S \wedge t_T))$, where the transition function $(t_S \wedge t_T)$ uses the output of the first transition as the input of the second

$$(t_S \wedge t_T)(s, t, a) = (s', t', c) \text{ where } t_S(s, a) = (s', b) \text{ and } t_T(t, b) = (t', c).$$

We consider Mealy deterministic automata up to isomorphism of their state space. Mealy deterministic finite automata form a symmetric monoidal category, **MealyAut**, *with sequential composition and product as defined above.*

This construction, together with the results of Section 5.2, leads to the following result.

Proposition 5.10. *The category of Mealy deterministic finite automata is the free structured feedback category over the isomorphisms of automaton state spaces $\mathsf{U}_{\mathsf{Aut}} \colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{FinSet}$, that is,*

$$\mathbf{MealyAut} \cong \mathsf{St}(\mathbf{FinSet}, \mathsf{U}_{\mathsf{Aut}}).$$

5.4 Automata in Span(Graph)

Our final construction is the canonical category of automata over the algebra of predicates given by spans. These automata are analogous to the previous transition systems in $\mathbf{Span}(\mathbf{Graph})_*$, but their state space contains an initial state and a set of final states. A similar definition has appeared previously in the literature for the modeling of Petri nets [46].

Definition 5.11. *A span automaton with left labels A and right labels B*

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X)$$

is given by a finite set of states S_X, a finite set of transitions E_X with source and target $s_X, t_X \colon E_X \to S_X$ and with left $l_X \colon E_X \to A$ and right $r_X \colon E_X \to B$

labels, an initial state $i_X \in S_X$, and a subset of final states $f_X\colon S_X \to 2$. The product of two span automata

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X) \text{ and}$$
$$\mathbf{Y} = (E_Y, S_Y, A', B', s_Y, t_Y, l_X, r_X, i_Y, f_Y),$$

is given by the component-wise product

$$\mathbf{X} \otimes \mathbf{Y} \coloneqq (E_X \times E_Y, S_X \times S_Y, A \times A', B \times B', \\ s_X \times s_Y, t_X \times t_Y, l_X \times l_Y, r_X \times r_Y, i_X \times i_Y, f_X \wedge f_Y).$$

The composition of two span automata

$$\mathbf{X} = (E_X, S_X, A, B, s_X, t_X, l_X, r_X, i_X, f_X) \text{ and}$$
$$\mathbf{Y} = (E_Y, S_Y, B, C, s_Y, t_Y, l_X, r_X, i_Y, f_Y),$$

is given by a pullback

$$\mathbf{X} \mathbin{⨾} \mathbf{Y} \coloneqq (E_X \times_B E_Y, S_X \times S_Y, A, C, s_X \times s_Y, t_X \times t_Y, l_X, r_Y, i_X \times i_Y, f_X \wedge f_Y),$$

where $E_X \times_B E_Y$ is the pullback of $E_X \overset{r_X}{\to} B \overset{l_Y}{\leftarrow} E_Y$. We consider span automata up to isomorphism of their state space. Span automata form a symmetric monoidal category **SpanAut** *with sequential composition and the monoidal product defined above.*

This construction, together with the results of Section 5.2, leads to the following result.

Proposition 5.12. *The category of span automata is the free structured feedback category over the category of spans with the inclusion functor from automaton state spaces, $\mathsf{U}_{\mathsf{AutSpan}}\colon \mathbf{Iso}(\mathbf{AutSt}) \to \mathbf{Span}(\mathbf{FinSet})$, that is,*

$$\mathbf{SpanAut} \cong \mathsf{St}(\mathbf{Span}(\mathbf{FinSet}), \mathsf{U}_{\mathsf{AutSpan}}).$$

Example 5.13. By the universal property of the $\mathsf{St}(\bullet)$ construction, each Mealy automaton in **MealyAut** functorially induces a span automaton in **SpanAut** whose graph is the graph of the Mealy automaton.

Consider the following Mealy automaton $\mathbf{X}$ in Figure 33, left. It has a state space $S_X = \{0, 1, 2\}$, left labels $A = \{a, b\}$ and trivial right labels $B = \mathbf{1}$, with initial state $i_X = 0$ and a unique final state $f_X(2) = \mathbf{true}$, while $f_X(0) = f_X(1) = \mathbf{false}$. Its transition function is given by $t_X(0, a) = 1$, $t_X(0, b) = 2$, $t_X(1, a) = 2$, $t_X(1, b) = 1$, $t_X(2, a) = 1$ and $t_X(2, b) = 1$.

The corresponding span automaton $\mathbf{sX}$ (Figure 33, right) is not only the transition system, but also the markings for initial and final state. Explicitly, its set of edges – or transitions – is given by tuples

$$E_{sX} = \{(0, a, 1), (0, b, 0), (1, a, 1), (1, b, 2), (2, a, 1), (2, b, 0)\},$$

its state space is again $S_{sX} = \{0, 1, 2\}$, its left and right boundaries are again $A = \{a, b\}$ and $B = \mathbf{1}$, with initial state $i_X = 0$ and a unique final state $f_X(2) = \mathbf{true}$, while $f_X(0) = f_X(1) = \mathbf{false}$.

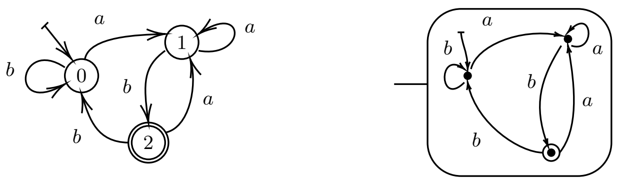


Fig. 33: Mealy automaton and associated span automaton.

6 Conclusions and Further Work

6.1 Discussion

We began this manuscript by pointing out the fragmented landscape of models for concurrent software. We have now formally proven that any theory of predicates with a notion of feedback that accepts reasonable axioms (A1-A5) must already contain a simulation of **Span**(**Graph**)$_*$: the algebra of open transition systems of Katis, Sabadini and Walters. This can be considered as being surprising given the dearth of results that establish the canonicity of models of concurrency. In this section we frame our contribution by expanding on its relevance for the wider research landscape.

Why *open* transition systems? Transition systems are the mainstay of concurrency theory: the firing semantics of a Petri net or the unfolding of the behaviour of a process algebra term are both important examples. In recent years significant effort [46,20,7] has been devoted to developing the mathematical foundations of *compositional* modelling. Roughly speaking, this means the study of semantic foundations that include the ability to compose/decompose systems – understood in a general sense – into more primitive components in a homomorphic/functorial way, so that the behaviour of the composite is determined by that of the components. Conceptually, this is a strengthening of the foundational computer science principle of *modularity*: our models do not merely need modular descriptions, these descriptions have to correspond homomorphically to a modular description of their behaviours. The **Span**(**Graph**)$_*$ algebra is an application of the principle of compositionality for transition systems.

We live in an age of complexity, with enormous implications for modern software development. Software is nowadays rarely about sandbox development of a software product intended to be run on a single machine. Rather, software engineering is evolving to a discipline that deals complex networks consisting of interactions of libraries, services, APIs both locally and in the cloud. We believe that compositional approaches, given that they are a tool to tame this complexity, will become increasingly important in the future.

Conceptual implications From an applied point of view, our results inform a minimal software architecture for a library that constructs and analyzes these modular transition systems, that are canonical models of concurrency. For instance, an object-oriented programmer may

- define a specific class for "theory of resources", providing methods for the fundamental operations of composing, tensoring, identities and swapping, and providing subclasses for "resources" and "processes", if necessary;
- implement an abstract method that computes the $\mathsf{St}(\bullet)$ construction: this method will take a theory of processes and instantiate the free feedback category as a new theory of processes;
- work uniformly with multiple theories of processes and the theories of automata they generate, providing auxiliary methods (for reachability, connectedness, joining, . . .) that work across different theories of transition systems;
- thus, thanks to our results, obtain at the end of the day a graph – in $\mathbf{Span}(\mathbf{Graph})$ – that can be understood as a transition system and analyzed with the same library.

A functional programmer may want to follow the architecture of the public Haskell implementation of the ideas of this paper [47], where our running example (Figure 1) is showcased.

The problem we solve is one of abstraction. When faced with the task of implementing models for transition systems, we could be tempted to simply implement each one of them separately. Our result shows that a much more succinct choice is possible. Reducing the lines of code is important: transition systems are usually formally analyzed to prove correctness; the less lines there are to analyze in the core implementation, the easiest it will be to be sure of its correctness.

Open transition systems allow us to construct transition systems from a few building blocks; compositionality eases both implementation and verification. Specifically, this could open an avenue to study the problem of compositional verification, where the automatic analysis of a global system must be modularly reduced to the analysis of its components.

6.2 Conclusion

We have characterized $\mathbf{Span}(\mathbf{Graph})_*$, an algebra of open transition systems, as the free feedback category over the category of spans of functions. To do so, we have used the $\mathsf{St}(\bullet)$ construction, characterized as the free feedback category in [36]. We have given this characterization more generally, for any category $\mathbf{C}$ with finite limits: the category $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$ of spans of graphs in $\mathbf{C}$ is the free feedback category over the category of spans in $\mathbf{C}$. Finally, we have defined a generalization of feedback categories to capture automata with initial and final states.

Further work will look at timed [11] and probabilistic [13,14] versions of the Cospan/Span model to connect it with recent work on modeling probabilistic programs with feedback categories [16]. We also plan to investigate the relationship between generalized feedback categories (Section 5) to approaches based on guarded recursion [26] and coalgebras [12,43].

References

1. Samson Abramsky. What are the fundamental structures of concurrency? We still don't know! *CoRR*, abs/1401.4973, 2014. URL: http://arxiv.org/abs/1401.4973, arXiv:1401.4973.
2. Jirí Adámek, Stefan Milius, and Jiri Velebil. Elgot algebras. *Log. Methods Comput. Sci.*, 2(5), 2006. doi:10.2168/LMCS-2(5:4)2006.
3. John C. Baez and Kenny Courser. Structured cospans. *CoRR*, abs/1911.04630, 2019.
4. Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
5. Nick Benton and Martin Hyland. Traced premonoidal categories. *RAIRO Theor. Informatics Appl.*, 37(4):273–299, 2003. doi:10.1051/ita:2003020.
6. Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993. doi:10.1007/978-3-642-78034-9.
7. Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. doi:10.1145/3290338.
8. Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear Relations on Streams. *Information and Computation*, 252:2–29, 2017. doi:10.1016/j.ic.2016.03.002.
9. 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.
10. Aurelio Carboni and Robert F. C. Walters. Cartesian Bicategories I. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987.
11. Alessandra Cherubini, Nicoletta Sabadini, and Robert F. C. Walters. Timing in the Cospan/Span model. *Electronic Notes in Theoretical Computer Science*, 104:81–97, 2004.
12. Ranald Clouston, Ales Bizjak, Hans Bugge Grathwohl, and Lars Birkedal. Programming and reasoning with guarded recursion for coinductive types. In Andrew M. Pitts, editor, *Foundations of Software Science and Computation Structures - 18th International Conference, FoSSaCS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9034 of *Lecture Notes in Computer Science*, pages 407–421. Springer, 2015. doi:10.1007/978-3-662-46678-0_26.
13. Luisa de Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of Markov processes. *Applied Categorical Structures*, 19(1):425–437, 2011.
14. Luisa de Francesco Albasini, Nicoletta Sabadini, and Robert F. C. Walters. The compositional construction of Markov processes II. *RAIRO-Theoretical Informatics and applications*, 45(1):117–142, 2011.
15. Mariangiola Dezani-Ciancaglini and Ugo de'Liguoro. Sessions and session types: An overview. In Cosimo Laneve and Jianwen Su, editors, *Web Services and Formal Methods, 6th International Workshop, WS-FM 2009, Bologna, Italy, September 4-5, 2009, Revised Selected Papers*, volume 6194 of *Lecture Notes in Computer Science*, pages 1–28. Springer, 2009. doi:10.1007/978-3-642-14458-5_1.
16. 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, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3531130.3533365.

17. William Henry Eccles and Frank Wilfred Jordan. Improvements in ionic relays. *British patent number: GB 148582*, 1918.
18. Calvin C. Elgot. Monadic computation and iterative algebraic theories. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 175–230. Elsevier, 1975.
19. Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.
20. Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality.* Cambridge University Press, 2019.
21. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, Filippo Schiavio, and Simone Tini. CospanSpan(Graph): a compositional description of the heart system. *Fundam. Informaticae*, 171(1-4):221–237, 2020.
22. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, and Simone Tini. Compositional modeling of biological systems in CospanSpan(Graph). In *Proc. of ICTCS 2020.* CEUR-WS, To appear.
23. Alessandro Gianola, Stefano Kasangian, and Nicoletta Sabadini. Cospan/Span(Graph): an Algebra for Open, Reconfigurable Automata Networks. In Filippo Bonchi and Barbara König, editors, *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, volume 72 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.CALCO.2017.2.
24. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
25. Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.
26. Sergey Goncharov and Lutz Schröder. Guarded traced categories. In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 313–330. Springer, 2018. doi:10.1007/978-3-319-89366-2_17.
27. Masahito Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. In Philippe de Groote, editor, *Typed Lambda Calculi and Applications, Third International Conference on Typed Lambda Calculi and Applications, TLCA '97, Nancy, France, April 2-4, 1997, Proceedings*, volume 1210 of *Lecture Notes in Computer Science*, pages 196–213. Springer, 1997. doi:10.1007/3-540-62688-3_37.
28. Masahito Hasegawa. The uniformity principle on traced monoidal categories. In Richard Blute and Peter Selinger, editors, *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002*, volume 69 of *Electronic Notes in Theoretical Computer Science*, pages 137–155. Elsevier, 2002. doi:10.1016/S1571-0661(04)80563-2.
29. Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014. doi:10.1145/2603088.2603124.

30. André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447 – 468, 04 1996. doi:10.1017/S0305004100074338.
31. Rudolf Emil Kalman, Peter L. Falb, and Michael A. Arbib. *Topics in mathematical system theory*, volume 1. McGraw-Hill New York, 1969.
32. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.
33. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Span(Graph): A Categorial Algebra of Transition Systems. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings*, volume 1349 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 1997. doi:10.1007/BFb0000479.
34. 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.
35. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A formalization of the IWIM model. In António Porto and Gruia-Catalin Roman, editors, *Coordination Languages and Models, 4th International Conference, COORDINATION 2000, Limassol, Cyprus, September 11-13, 2000, Proceedings*, volume 1906 of *Lecture Notes in Computer Science*, pages 267–283. Springer, 2000. doi:10.1007/3-540-45263-X_17.
36. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *RAIRO-Theor. Informatics Appl.*, 36(2):181–194, 2002. doi:10.1051/ita:2002009.
37. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A Process Algebra for the Span(Graph) Model of Concurrency. *arXiv preprint arXiv:0904.3964*, 2009.
38. Stephen Lack. Composing PROPs. *Theory and Applications of Categories*, 13(9):147–163, 2004.
39. Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. A Canonical Algebra of Open Transition Systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*, volume 13077 of *Lecture Notes in Computer Science*, pages 63–81. Springer, 2021. doi:10.1007/978-3-030-90636-8_4.
40. Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.
41. S. J. Mason. Feedback Theory - Some properties of signal flow graphs. *Proceedings of the Institute of Radio Engineers*, 41(9):1144–1156, 1953. doi:10.1109/JRPROC.1953.274449.
42. George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.
43. Stefan Milius and Tadeusz Litak. Guard your daggers and traces: Properties of guarded (co-) recursion. *Fundamenta Informaticae*, 150(3-4):407–449, 2017.
44. Duško Pavlović. Maps I: relative to a factorisation system. *Journal of Pure and Applied Algebra*, 99(1):9–34, 1995.
45. Kate Ponto and Michael Shulman. Traces in symmetric monoidal categories. *Expositiones Mathematicae*, 32(3):248–273, 2014. URL: http://dx.doi.org/10.1016/J.EXMATH.2013.12.003, doi:10.1016/j.exmath.2013.12.003.

46. Julian Rathke, Pawel Sobocinski, and Owen Stephens. Compositional Reachability in Petri Nets. In Joël Ouaknine, Igor Potapov, and James Worrell, editors, *Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings*, volume 8762 of *Lecture Notes in Computer Science*, pages 230–243. Springer, 2014. doi:10.1007/978-3-319-11439-2_18.
47. Mario Román. Span graph via the state construction, 2022. URL: https://github.com/mroman42/feedback-span-graph.
48. Robert Rosebrugh, Nicoletta Sabadini, and Robert F. C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15(6):164–177, 2005.
49. Nicoletta Sabadini, Filippo Schiavio, and Robert F. C. Walters. On the geometry and algebra of networks with state. *Theor. Comput. Sci.*, 664:144–163, 2017.
50. Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010. doi:10.1007/978-3-642-12821-9_4.
51. Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines*. Bell Telephone Laboratories, 1942.
52. Paweł Sobociński. A non-interleaving process calculus for multi-party synchronisation. In *2nd Interaction and Concurrency Experience: Structured Interactions, (ICE 2009)*, volume 12 of *EPTCS*, 2009. URL: http://users.ecs.soton.ac.uk/ps/papers/ice09.pdf, doi:10.4204/eptcs.12.6.
53. Paweł Sobociński. Representations of Petri net interactions. In *Concurrency Theory, 21th International Conference, (CONCUR 2010)*, volume 6269 of *Lecture Notes in Computer Science*, pages 554–568. Springer, 2010. doi:10.1007/978-3-642-15375-4_38.

2. Monoidal Streams for Dataflow Programming

Elena Di Lavore, Giovanni de Felice, Mario Román
IEEE Symposium on Logic in Computer Science (LiCS, 2023)
Distinguished paper, Kleene Award to the best student paper

Abstract: We introduce monoidal streams: a generalization of causal stream functions to monoidal categories. In the same way that streams provide semantics to dataflow programming with pure functions, monoidal streams provide semantics to dataflow programming with theories of processes represented by a symmetric monoidal category. At the same time, monoidal streams form a feedback monoidal category, which can be used to interpret signal flow graphs. As an example, we study a stochastic dataflow language.

Declaration: *Hereby I declare that my contribution to this manuscript was to: provide the main theorem and its proof and write most of the paper. The original idea was worked out with Elena Di Lavore, who also wrote some parts of the paper. The theorem characterizing stochastic processes was worked out together with Giovanni di Felice.*

Monoidal Streams for Dataflow Programming

Elena Di Lavore
Giovanni de Felice
Mario Román

Abstract

We introduce monoidal streams: a generalization of causal stream functions to monoidal categories. They have a feedback structure that gives semantics to signal flow graphs. In the same way that streams provide semantics to dataflow programming with pure functions, monoidal streams provide semantics to dataflow programming with theories of processes represented by a symmetric monoidal category. As an example, we study a stochastic dataflow language.

Keywords: Monoidal stream, Stream, Monoidal category, Dataflow programming, Feedback, Signal flow graph, Coalgebra, Stochastic process.

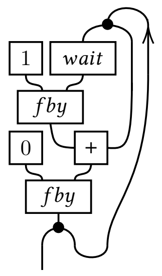
1 Introduction

Dataflow languages. Dataflow (or *stream-based*) programming languages, such as Lucid [35, 77], follow a paradigm in which every declaration represents an infinite list of values: a *stream* [8, 75]. The following program in a Lucid-like language (Figure 1) computes the Fibonacci sequence, thanks to a Fby ("followed by") operator.

$$fib = 0 \ \textsc{Fby}\ (fib + (1\ \textsc{Fby}\ \textsc{Wait}(fib)))$$

Figure 1. *The Fibonacci sequence is 0 followed by the Fibonacci sequence plus the Fibonacci sequence preceded by a 1.*

The control structure of dataflow programs is inspired by *signal flow graphs* [8, 57, 69]. Signal flow graphs are diagrammatic specifications of processes with feedback loops, widely used in control system engineering. In a dataflow program, feedback loops represent how the current value of a stream may depend on its previous values. For instance, the previous program (Figure 1) corresponds to the signal flow graph in Figure 2.



$$\begin{aligned} \mathsf{fib} = \mathsf{fbk}(&\mathsf{copy};\\ &\partial(1 \times \mathsf{wait}) \times \mathrm{id};\\ &\partial(\mathsf{fby}) \times \mathrm{id};\\ &\partial(+);\\ &0 \times \mathrm{id};\\ &\mathsf{fby};\\ &\mathsf{copy}) \end{aligned}$$

Figure 2. Fibonacci: signal flow graph and morphism.

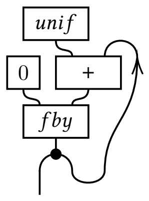
Monoidal categories. Any theory of processes that *compose sequentially and in parallel*, satisfying reasonable axioms, forms a *monoidal category*. Examples include functions [49], probabilistic channels [19, 30], partial maps [21], database queries [14], linear resource theories [23] and quantum processes [2]. Signal flow graphs are the graphical syntax for *feedback monoidal categories* [15, 16, 26, 33, 46]: they are the *string diagrams* for any of these theories, extended with *feedback*.

Yet, semantics of dataflow languages have been mostly restricted to theories of pure functions [8, 24, 25, 58, 76]: what are called *cartesian* monoidal categories. We claim that this restriction is actually inessential; dataflow programs may take semantics in non-cartesian monoidal categories, exactly as their signal flow graphs do.

The present work provides this missing semantics: we construct *monoidal streams* over a symmetric monoidal category, which form a *feedback monoidal category*. Monoidal streams model the values of a *monoidal dataflow language*, in the same way that streams model the values of a classical dataflow language. This opens the door to stochastic, effectful, or quantum dataflow languages. In particular, we give semantics and string diagrams for a *stochastic dataflow programming language*, where the following code can be run.

$$walk = 0\ \textsc{Fby}\ (\textsc{Uniform}(-1,1) + walk)$$

Figure 3. *A stochastic dataflow program. A random walk is 0 followed by the random walk plus a stochastic stream of steps to the left (-1) or to the right (1), sampled uniformly.*



$$\begin{aligned} \mathsf{walk} = \mathsf{fbk}(&\\ &\partial(\mathsf{unif}) \otimes \mathrm{id};\\ &0 \otimes \partial(+);\\ &\mathsf{fby};\\ &\mathsf{copy}) \end{aligned}$$

Figure 4. Random walk: signal flow graph and morphism.

1.1 Contributions

Our main novel contribution is the definition of a feedback monoidal category of *monoidal streams* over a symmetric monoidal category (Stream, Definition 5.1 and theorem 5.10).

Monoidal streams form a final coalgebra; for sufficiently well-behaved monoidal categories (Definition 4.9), we give an explicit construction of this coalgebra (Definition 4.7).

In cartesian categories, the *causal functions* of Uustalu and Vene [75] (see also [72]) are a particular case of our monoidal streams (Theorems 6.1 and 6.3). In the category of stochastic functions, our construction captures the notion of *controlled stochastic process* [27, 66] (Theorem 7.2).

In order to arrive to this definition, we unify the previous literature: we characterize the cartesian "intensional stateful sequences" of Katsumata and Sprunger with a final coalgebra (Theorem 2.5), and then "extensional stateful sequences" in terms of the "feedback monoidal categories" of Katis, Sabadini and Walters [46] (Theorem 3.8). We justify observational equivalence with a refined fixpoint equation that employs *coends* (Theorem 4.11). We strictly generalize "stateful sequences" from the cartesian to the monoidal case.

Finally, we extend a type theory of symmetric monoidal categories with a feedback operator (Section 8) and we use it as a stochastic dataflow programming language.

1.2 Related work

Coalgebraic streams. Uustalu and Vene [75] provide elegant *comonadic* semantics for a (cartesian) Lucid-like programming language. We shall prove that their exact technique cannot be possibly extended to arbitrary monoidal categories (Theorem 6.1). However, we recover their semantics as a particular case of our monoidal streams (Theorem 6.3).

Stateful morphism sequences. Sprunger and Katsumata constructed the category of *stateful sequences* in the cartesian case [73]. Our work is based on an unpublished work by Román [65] that first exteneded this definition to the symmetric monoidal case, using *coends* to justify *extensional equality*. Shortly after, Carette, de Visme and Perdrix [18] rederived this construction and applied it to the case of completely positive maps between Hilbert spaces, using (a priori) a slightly different notion of equality. We synthetise some of this previous work, we justify it for the first time using coalgebra and we particularize it to some cases of interest.

Feedback. Feedback monoidal categories are a weakening of *traced monoidal categories*. The construction of the free such categories is originally due to Katis, Sabadini and Walters [46]. Feedback monoidal categories and their free construction have been repurposed and rediscovered multiple times in the literature [13, 32, 38, 44]. Di Lavore et al. [26] summarize these uses and introduce *delayed feedback*.

General and dependent streams. Our work concerns *synchronous* streams: those where, at each point in time $t = 0, 1, \dots$, the stream process takes exactly one input and produces exactly one output. This condition is important in certain contexts like, for instance, real-time embedded systems; but it is not always present. The study of asynchronous stream transformers and their universal properties is considerably different [1], and we refer the reader to the recent work of Garner [31] for a discussion on *non-synchronous* streams. Finally, when we are concerned with *dependent streams* indexed by time steps, a possible approach, when our base category is a topos, is to use the *topos of trees* [9].

Categorical dataflow programming. Category theory is a common tool of choice for dataflow programming [32, 56, 67]. In particular, profunctors and coends are used by Hildebrandt, Panangaden and Winskel [37] to generalise a model of non-deterministic dataflow, which has been the main focus [51, 54, 60] outside cartesian categories.

1.3 Synopsis

This manuscript contains three main definitions in terms of universal properties (*intensional, extensional and observational streams*, Definitions 2.2, 3.7 and 4.1); and three explicit constructions for them (*intensional, extensional and observational sequences*, Definitions 2.4, 2.8 and 4.7). Each definition is refined into the next one: each construction is a quotienting of the previous one.

Sections 1.4 and 2.3 contain expository material on coalgebra and dinaturality. Section 2 presents intensional monoidal streams. Section 3 introduces extensional monoidal streams in terms of feedback monoidal categories. Section 4 introduces the definitive *observational* equivalence and defines *monoidal streams*. Section 5 constructs the feedback monoidal category of monoidal streams. Sections 6 and 7 present two examples: cartesian and stochastic streams. Section 8 introduces a type theory for feedback monoidal categories.

1.4 Prelude: Coalgebra

In this preparatory section, we introduce some background material on coalgebra [3, 41, 67]. Coalgebra is the category-theoretic study of stateful systems and infinite data-structures, such as streams. These structures arise as *final coalgebras*: universal solutions to certain functor equations.

Let us fix an endofunctor $F \colon \mathsf{C} \to \mathsf{C}$ through the section.

Definition 1.1. A *coalgebra* (Y, β) is an object $Y \in \mathsf{C}$, together with a morphism $\beta \colon Y \to FY$. A *coalgebra morphism* $g \colon (Y, \beta) \to (Y', \beta')$ is a morphism $g \colon Y \to Y'$ such that $g ; \beta' = \beta ; Fg$.

Coalgebras for an endofunctor form a category with coalgebra morphisms between them. A *final coalgebra* is a final object in this category. As such, final coalgebras are unique up to isomorphism when they exist.

Definition 1.2. A *final coalgebra* is a coalgebra (Z, γ) such that for any other coalgebra (Y, β) there exists a unique coalgebra morphism $g \colon (Y, \beta) \to (Z, \gamma)$.

Our interest in final coalgebras derives from the fact that they are canonical fixpoints of an endofunctor. Specifically, Lambek's theorem (Theorem 1.4) states that whenever the final coalgebra exists, it is a fixpoint.

Definition 1.3. A *fixpoint* is a coalgebra (Y, β) such that $\beta \colon Y \to FY$ is an isomorphism. A *fixpoint morphism* is a coalgebra morphism between fixpoints: fixpoints and fixpoint morphisms form a full subcategory of the category of coalgebras. A *final fixpoint* is a final object in this category.

Theorem 1.4 (Lambek, [50])**.** *Final coalgebras are fixpoints. As a consequence, when they exist, they are final fixpoints.*

The last question before continuing is how to explicitly construct a final coalgebra. This is answered by Adamek's theorem (Theorem 1.5). The reader may be familiar with Kleene's theorem for constructing fixpoints [74]: the least fixpoint of a monotone function $f \colon X \to X$ in a directed-complete partial order $(X, \leqslant)$ is the supremum of the chain $\bot \leqslant f(\bot) \leqslant f(f(\bot)) \leqslant \dots$, where $\bot$ is the least element of the partial order, whenever this supremum is preserved by f. This same result can be categorified into a fixpoint theorem for constructing final coalgebras: the directed-complete poset becomes a category with ω-chain limits; the monotone function becomes an endofunctor; and the least element becomes the final object.

Theorem 1.5 (Adamek, [4])**.** *Let* D *be a category with a final object* 1 *and* ω*-shaped limits. Let* $F \colon \mathsf{D} \to \mathsf{D}$ *be an endofunctor. We write* $L = \lim_n F^n 1$ *for the limit of the following* ω*-chain, which is called the* terminal sequence *of* F*.*

$$1 \xleftarrow{!} F1 \xleftarrow{F!} FF1 \xleftarrow{FF!} FFF1 \xleftarrow{FFF!} \dots$$

Assume that F *preserves this limit, meaning that the canonical morphism* $FL \to L$ *is an isomorphism. Then,* L *is the final* F*-coalgebra.*

2 Intensional Monoidal Streams

This section introduces a preliminary definition of *monoidal stream* in terms of a fixpoint equation (in Figure 5). In later sections, we refine both this definition and its characterization into the definitive notion of *monoidal stream.*

Let $(\mathsf{C}, \otimes, I)$ be a fixed symmetric monoidal category.

2.1 The fixpoint equation

Classically, type-variant streams have a neat coinductive definition [41, 67] that says:

> *"A stream of type* $\mathbb{A} = (A_0, A_1, \dots)$ *is an element of* A_0 *together with a stream of type* $\mathbb{A}^+ = (A_1, A_2, \dots)$*".*

Formally, streams are the final fixpoint of the equation

$$\mathrm{S}(A_0, A_1, \dots) \cong A_0 \times \mathrm{S}(A_1, A_2, \dots);$$

and this fixpoint is computed to be $\mathrm{S}(\mathbb{A}) = \prod_{n \in \mathbb{N}}^{\infty} A_n$.

In the same vein, we want to introduce not only streams but *stream processes* over a fixed theory of processes.

> *"A stream process from* $\mathbb{X} = (X_0, X_1, \dots)$ *to* $\mathbb{Y} = (Y_0, Y_1, \dots)$ *is a process from* X_0 *to* Y_0 *communicating along a channel* M *with a stream process from* $\mathbb{X}^+ = (X_1, X_2, \dots)$ *to* $\mathbb{Y}^+ = (Y_1, Y_2, \dots)$*."*

Streams are recovered as stream processes on an empty input, so we take this more general slogan as our preliminary definition of *monoidal stream* (in Definition 2.2). Formally, they are the final fixpoint of the equation in Figure 5.

$$\mathrm{T}(\mathbb{X}, \mathbb{Y}) \cong \sum_{M \in \mathsf{C}} \mathrm{hom}(X_0, M \otimes Y_0) \times \mathrm{T}(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Figure 5. Fixpoint equation for intensional streams.

Remark 2.1 (Notation). Let $\mathbb{X} \in [\mathbb{N}, \mathsf{C}]$ be a sequence of objects $(X_0, X_1, \dots)$. We write $\mathbb{X}^+$ for its *tail* $(X_1, X_2, \dots)$. Given $M \in \mathsf{C}$, we write $M \cdot \mathbb{X}$ for the sequence $(M \otimes X_0, X_1, X_2, \dots)$; As a consequence, we write $M \cdot \mathbb{X}^+$ for $(M \otimes X_1, X_2, X_3, \dots)$.

Definition 2.2. The set of *intensional monoidal streams* $\mathrm{T} \colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathsf{Set}$, depending on inputs and outputs, is the final fixpoint of the equation in Figure 5.

Remark 2.3 (Initial fixpoint). There exists an obvious fixpoint for the equation in Figure 5: the constant empty set. This solution is the *initial* fixpoint, a minimal solution. The *final* fixpoint will be realized by the set of *intensional sequences.*

2.2 Intensional sequences

We now construct the set of intensional streams explicitly (Theorem 2.5). For this purpose, we generalize the "stateful morphism sequences" of Katsumata and Sprunger [73] from cartesian to arbitrary symmetric monoidal categories (Definition 2.4). We derive a novel characterization of these "sequences" as the desired final fixpoint (Theorem 2.5).

In the work of Katsumata and Sprunger, a stateful sequence is a sequence of morphisms $f_n \colon M_{n-1} \times X_n \to M_n \times Y_n$ in a cartesian monoidal category. These morphisms represent a process at each point in time $n = 0, 1, 2, \dots$. At each step n, the process takes an input X_n and, together with the stored memory M_{n-1}, produces some output Y_n and writes to a new memory M_n. The memory is initially empty, with $M_{-1} = 1$ being the final object by convention. We extend this definition to any symmetric monoidal category.

Definition 2.4. Let $\mathbb{X}$ and $\mathbb{Y}$ be two sequences of objects representing inputs and outputs, respectively. An *intensional sequence* is a sequence of objects $(M_0, M_1, \dots)$ together with a sequence of morphisms

$$(f_n \colon M_{n-1} \otimes X_n \to M_n \otimes Y_n)_{n \in \mathbb{N}},$$

where, by convention, $M_{-1} = I$ is the unit of the monoidal category. In other words, the set of intensional sequences is

$$\mathrm{Int}(\mathbb{X}, \mathbb{Y}) := \sum_{M \in [\mathbb{N}, \mathsf{C}]} \prod_{n=0}^{\infty} \mathrm{hom}(M_{n-1} \otimes X_n, M_n \otimes Y_n).$$

We now prove that intensional sequences are the final fixpoint of the equation in Figure 5. The following Theorem 2.5 serves two purposes: it gives an explicit final solution to this fixpoint equation and it gives a novel universal property to intensional sequences.

Theorem 2.5. *Intensional sequences are the explicit construction of intensional streams, $\mathbb{T} \cong \mathrm{Int}$. In other words, they are a fixpoint of the equation in Figure 5, and they are the final such one.*

Proof sketch. It is known that categories of functors over sets, such as $[[\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}]$, have all limits. Adamek's theorem (Theorem 1.5) states that, if the following limit is a fixpoint, it is indeed the final one.

$$\lim_{n \in \mathbb{N}} \sum_{M_0,\ldots,M_n} \prod_{t=0}^{n} \mathrm{hom}(M_{t-1} \otimes X_t, M_t \otimes Y_t) \tag{1}$$

Connected limits commute with coproducts and the limit of the nth-product is the infinite product. Thus, Equation (1) is isomorphic to $\mathrm{Int}(\mathbb{X}, \mathbb{Y})$. It only remains to show that $\mathrm{Int}(\mathbb{X}, \mathbb{Y})$ is a fixpoint, which means it should be isomorphic to the following expression.

$$\sum_{M_0} \mathrm{hom}(X_0, M_0 \otimes Y_0) \times \sum_{M \in [\mathbb{N}, \mathsf{C}]} \prod_{n=1}^{\infty} \mathrm{hom}(M_{n-1} \otimes X_n, M_n \otimes Y_n). \tag{2}$$

Cartesian products distribute over coproducts, so Equation (2) is again isomorphic to $\mathrm{Int}(\mathbb{X}, \mathbb{Y})$. □

2.3 Interlude: Dinaturality

During the rest of this text, we deal with two different definitions of what it means for two processes to be equal: *extensional* and *observational equivalence*, apart from pure *intensional equality*. Fortunately, when working with functors of the form $P \colon \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$, the so-called *endoprofunctors*, we already have a canonical notion of equivalence.

Endoprofunctors $P \colon \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$ can be thought as indexing families of processes $P(M, N)$ by the types of an input channel M and an output channel N. A process $p \in P(M, N)$ writes to a channel of type N and then reads from a channel of type M.

Now, assume we also have a transformation $r \colon N \to M$ translating from output to input types. Then, we can *plug the output to the input*: the process p writes with type N, then r translates from N to M, and then p uses this same output as its input M. This composite process can be given two sligthly different descriptions; the process could

- translate *after writing*, $P(M, r)(p) \in P(M, M)$, or
- translate *before reading*, $P(r, N)(p) \in P(N, N)$.

These two processes have different types. However, with the output plugged to the input, it does not really matter when to apply the translation. These two descriptions represent the same process: they are *dinaturally equivalent*.

Definition 2.6 (Dinatural equivalence)**.** For any functor $P \colon \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$, consider the set

$$S_P = \sum_{M \in \mathsf{C}} P(M, M).$$

Dinatural equivalence, $(\sim)$, on the set S_P is the smallest equivalence relation satisfying $P(M, r)(p) \sim P(r, N)(p)$ for each $p \in P(M, N)$ and each $r \in \mathrm{hom}(N, M)$.

Coproducts quotiented by dinatural equivalence construct a particular form of colimit called a *coend*. Under the process interpretation of profunctors, taking a coend means *plugging an output to an input* of the same type.

Definition 2.7 (Coend, [53, 55])**.** Let $P \colon \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$ be a functor. Its *coend* is the coproduct of $P(M, M)$ indexed by $M \in \mathsf{C}$, quotiented by dinatural equivalence.

$$\int^{M \in \mathsf{C}} P(M, M) = \left(\sum_{M \in \mathsf{C}} P(M, M) \middle/ \sim \right).$$

That is, the coend is the colimit of the diagram with a *cospan* $P(M, M) \leftarrow P(M, N) \rightarrow P(N, N)$ for each $f \colon N \to M$.

2.4 Towards extensional memory channels

Let us go back to intensional monoidal streams. Consider a family of processes $f_n \colon M_{n-1} \otimes X_n \to Y_n \otimes N_n$ reading from memories of type M_n but writing to memories of type N_n. Assume we also have processes $r_n \colon N_n \to M_n$ translating from output to input memory. Then, we can consider the process that does f_n, translates from memory N_n to memory M_n and then does f_{n+1}. This process is described by two different intensional sequences,

- $(f_n ; (r_n \otimes \mathrm{id}) \colon M_{n-1} \otimes X_n \to M_n \otimes Y_n)_{n \in \mathbb{N}}$, and
- $((r_{n-1} \otimes \mathrm{id}); f_n \colon N_{n-1} \otimes X_n \to N_n \otimes Y_n)_{n \in \mathbb{N}}$.

These two intensional sequences have different types for the memory channels. However, in some sense, they represent *the same process description*. If we do not care about what exactly it is that we save to memory, we should consider two such processes to be equal (as in Figure 6, where "the same process" can keep two different values in memory). Indeed, dinaturality in the memory channels M_n is the smallest equivalence relation $(\sim)$ satisfying

$$(f_n ; (r_n \otimes \mathrm{id}))_{n \in \mathbb{N}} \sim ((r_{n-1} \otimes \mathrm{id}); f_n)_{n \in \mathbb{N}}.$$

This is precisely the quotienting that we perform in order to define *extensional sequences*.

Definition 2.8. *Extensional equivalence* of intensional sequences, $(\sim)$, is dinatural equivalence in the memory channels M_n. An *extensional sequence* from $\mathbb{X}$ to $\mathbb{Y}$ is an equivalence class

$$\langle f_n \colon M_{n-1} \otimes X \to M_n \otimes Y \rangle_{n \in \mathbb{N}}$$

of intensional sequences under extensional equivalence.

In other words, the set of extensional sequences is the set of intensional sequences substituting the coproduct by a coend,

$$\mathrm{Ext}(\mathbb{X}, \mathbb{Y}) = \int^{M \in [\mathbb{N}, \mathsf{C}]} \prod_{i=0}^{\infty} \mathrm{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

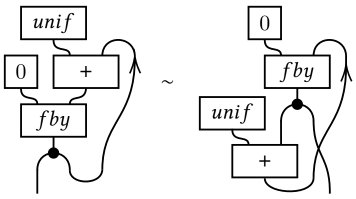


Figure 6. Extensionally equivalent walks keeping different memories: the *current* position vs. the *next* position.

3 Extensional Monoidal Streams

In this section, we introduce *extensional monoidal streams* in terms of a universal property: *extensional streams are the morphisms of the free delayed-feedback monoidal category* (Theorem 3.8).

Feedback monoidal categories come from the work of Katis, Sabadini and Walters [46]. They are instrumental to our goal of describing and composing signal flow graphs: they axiomatize a graphical calculus that extends the well-known string diagrams for monoidal categories with *feedback loops* [26, 46]. Constructing the *free feedback monoidal category* (Definition 3.4) will lead to the main result of this section: extensional sequences are the explicit construction of extensional streams (Theorem 3.8).

We finish the section by exploring how extensional equivalence may not be enough to capture true observational equality of processes (Example 3.10).

3.1 Feedback monoidal categories

Feedback monoidal categories are symmetric monoidal categories with a "feedback" operation that connects outputs to inputs. They have a natural axiomatization (Definition 3.1) that has been rediscovered independently multiple times, with only slight variations [10, 13, 45, 46]. It is weaker than that of *traced monoidal categories* [26] while still satisfying a normalization property (Theorem 3.5). We present a novel definition that generalizes the previous ones by allowing the feedback operator to be guarded by a monoidal endofunctor.

Definition 3.1. A *feedback monoidal category* is a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ endowed with a monoidal endofunctor $\mathrm{F} \colon \mathsf{C} \to \mathsf{C}$ and an operation

$$\mathrm{fbk}_S \colon \mathrm{hom}(\mathrm{F}(S) \otimes X, S \otimes Y) \to \mathrm{hom}(X, Y)$$

for all S, X and Y objects of C; this operation needs to satisfy the following axioms.

(A1). Tightening: $u \,;\mathrm{fbk}_S(f)\,; v = \mathrm{fbk}_S((\mathsf{id}_{\mathrm{F}S} \otimes u)\,; f\,; (\mathsf{id}_S \otimes v))$.
(A2). Vanishing: $\mathrm{fbk}_I(f) = f$.
(A3). Joining: $\mathrm{fbk}_T(\mathrm{fbk}_S(f)) = \mathrm{fbk}_{S\otimes T}(f)$
(A4). Strength: $\mathrm{fbk}_S(f) \otimes g = \mathrm{fbk}_S(f \otimes g)$.
(A5). Sliding: $\mathrm{fbk}_S((\mathrm{F}h \otimes \mathsf{id}_X)\,; f) = \mathrm{fbk}_T(f\,; (h \otimes \mathsf{id}_Y))$.

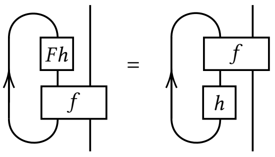


Figure 7. The sliding axiom (A5).

A *feedback functor* is a symmetric monoidal functor that preserves the feedback structure (Appendix, Definition A.9).

Remark 3.2 (Wait or trace). In a feedback monoidal category $(\mathsf{C}, \mathrm{fbk})$, we construct the morphism $\mathsf{wait}_X \colon X \to FX$ as a feedback loop over the symmetry, $\mathsf{wait}_X = \mathrm{fbk}(\sigma_{X,X})$. A *traced monoidal category* [43] is a feedback monoidal category guarded by the identity functor such that $\mathsf{wait}_X = \mathrm{id}_X$.

The "state construction", $\mathsf{St}(\bullet)$, realizes the *free* feedback monoidal category. As it happens with feedback monoidal categories, this construction has appeared in the literature in slightly different forms. It has been used for describing a "memoryful geometry of interaction" [38], "stateful resource calculi" [13], and "processes with feedback" [44, 46].

The idea in all of these cases is the same: we allow the morphisms of a monoidal category to depend on a "state space" S, possibly guarded by a functor. Adding a state space is equivalent to freely adding feedback [26].

Definition 3.3. A *stateful morphism* is a pair (S, f) consisting of a "state space" $S \in \mathsf{C}$ and a morphism $f \colon \mathrm{F}S \otimes X \to S \otimes Y$. We say that two stateful morphisms are *sliding equivalent* if they are related by the smallest equivalence relation satisfying $(S, (\mathrm{F}r \otimes \mathrm{id})\,; h) \sim (T, h\,; (r \otimes \mathrm{id}))$ for each $h \colon X \otimes \mathrm{F}T \to S \otimes Y$ and each $r \colon S \to T$.

In other words, sliding equivalence is *dinaturality in* S.

Definition 3.4 ($\mathsf{St}(\bullet)$ construction, [26, 46])**.** We write $\mathsf{St}_{\mathrm{F}}(\mathsf{C})$ for the symmetric monoidal category that has the same objects as C and whose morphisms from X to Y are stateful morphisms $f \colon \mathrm{F}S \otimes X \to S \otimes Y$ *up to sliding.*

$$\mathrm{hom}_{\mathsf{St}_{\mathrm{F}}(\mathsf{C})}(X, Y) := \int^{S \in \mathsf{C}} \mathrm{hom}_{\mathsf{C}}(\mathrm{F}S \otimes X, S \otimes Y).$$

Theorem 3.5 (see [46])**.** $\mathsf{St}_{\mathrm{F}}(\mathsf{C})$ *is the free feedback monoidal category over* (C, F).

3.2 Extensional monoidal streams

Monoidal streams should be, in some sense, the minimal way of adding feedback to a theory of processes. The output of this feedback, however, should be delayed by one unit of time: the category $[\mathbb{N}, \mathsf{C}]$ is naturally equipped with a *delay* endofunctor that shifts a sequence by one. Extensional monoidal streams form the free delayed-feedback category.

Definition 3.6 (Delay functor)**.** Let $\partial\colon [\mathbb{N}, \mathsf{C}] \to [\mathbb{N}, \mathsf{C}]$ be the endofunctor defined on objects $\mathbb{X} = (X_0, X_1, \dots)$, as $\partial(\mathbb{X}) = (I, X_0, X_1, \dots)$; and on morphisms $\mathbb{f} = (f_0, f_1, \dots)$ as $\partial(\mathbb{f}) = (\mathrm{id}_I, f_0, f_1, \dots)$.

Definition 3.7. The set of *extensional monoidal streams*, depending on inputs and outputs, $\mathsf{R}\colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathsf{Set}$, is the hom-set of the free feedback monoidal category over $([\mathbb{N}, \mathsf{C}], \partial)$.

We characterize now extensional streams in terms of extensional sequences and the $\mathsf{St}(\bullet)$-construction.

Theorem 3.8. *Extensional sequences are the explicit construction of extensional streams,* $\mathsf{R} \cong \mathsf{Ext}$.

Proof. Note that $\mathsf{Ext}(\mathbb{X}, \mathbb{Y}) = \mathsf{St}_\partial([\mathbb{N}, \mathsf{C}])(\mathbb{X}, \mathbb{Y})$. That is, the extensional sequences we defined using dinaturality coincide with the morphisms of $\mathsf{St}_\partial([\mathbb{N}, \mathsf{C}])$, the free feedback monoidal category over $([\mathbb{N}, \mathsf{C}], \partial)$ in Definition 3.4. □

As a consequence, the calculus of signal flow graphs given by the syntax of feedback monoidal categories is sound and complete for extensional equivalence over $[\mathbb{N}, \mathsf{C}]$.

3.3 Towards observational processes

Extensional sequences were an improvement over intensional sequences because they allowed us to equate process descriptions that were *essentially the same*. However, we could still have two processes that are "observationally the same" without them being described in the same way.

Remark 3.9 (Followed by). As we saw in the Introduction, *"followed by"* is a crucial operation in dataflow programming. Any sequence can be decomposed as $\mathbb{X} \cong X_0 \cdot \partial(\mathbb{X}^+)$.[1] We call *"followed by"* to the coherence map in $[\mathbb{N}, \mathsf{C}]$ that witnesses this decomposition.

$$\mathsf{fby}_{\mathbb{X}}\colon X_0 \cdot \partial(\mathbb{X}^+) \to \mathbb{X}$$

In the case of constant sequences $\mathbb{X} = (X, X, \dots)$, we have that $\mathbb{X}^+ = \mathbb{X}$; which means that "followed by" has type $\mathsf{fby}_{\mathbb{X}}\colon X \cdot \partial\mathbb{X} \to \mathbb{X}$.

Example 3.10. Consider the extensional stateful sequence, in any cartesian monoidal category, that saves the first input to memory without ever producing an output. *Observationally,* this is no different from simply discarding the first input,

[1]This can also be seen as the isomorphism making "sequences" a final coalgebra. That is, the first slogan we saw in Section 2.1.

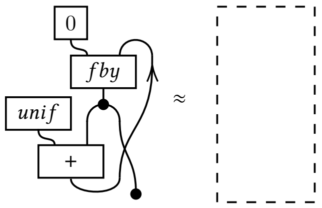


Figure 8. *Observationally*, a silent process does nothing.

$(\downarrow)_X\colon X \to 1$. However, in principle, we cannot show that these are *extensionally equal*, that is, $\mathsf{fbk}(\mathsf{fby}_{\mathbb{X}}) \neq (\downarrow)_X$.

More generally, discarding the result of any stochastic or deterministic signal flow graph is, *observationally*, the same as doing nothing (Figure 8, consequence of Theorem 7.2).

4 Observational Monoidal Streams

In this section, we introduce our definitive *monoidal streams*: *observational streams* (Definition 4.1). Their explicit construction is given by observational sequences: extensional sequences quotiented by observational equivalence.

Intuitively, two processes are observationally equal if they are "equal up to stage n", for any $n \in \mathbb{N}$. We show that, in sufficiently well-behaved monoidal categories (which we call *productive*, Definition 4.9), the set of observational sequences given some inputs and outputs is the final coalgebra of a fixpoint equation (Figure 9). The name "observational equivalence" is commonly used to denote equality on the final coalgebra: Theorem 4.11 justifies our use of the term.

4.1 Observational streams

We saw in Section 2 that we can define intensional sequences as a solution to a fixpoint equation. We now consider the same equation, just substituting the coproduct for a coend.

Definition 4.1 (Observational streams)**.** The set of *observational monoidal streams*, depending on inputs and outputs, is the functor $\mathsf{Q}\colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathsf{Set}$ given by the final fixpoint of the equation in Figure 9.

$$\mathsf{Q}(\mathbb{X}, \mathbb{Y}) \cong \int^{M \in \mathsf{C}} \mathrm{hom}(X_0, M \otimes Y_0) \times \mathsf{Q}(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Figure 9. Fixpoint equation for observational streams.

The explicit construction of this final fixpoint will be given by observational sequences (Theorem 4.11).

4.2 Observational sequences

We said that observational equivalence is "equality up to stage n", so our first step will be to define what it means to truncate an extensional sequence at any given $n \in \mathbb{N}$.

Definition 4.2 (n-Stage process)**.** An *n-stage process* from inputs $\mathbb{X} = (X_0, X_1, \dots)$ to outputs $\mathbb{Y} = (Y_0, Y_1, \dots)$ is an element of the set

$$\mathsf{Stage}_n(\mathbb{X},\mathbb{Y}) = \int^{M_0,\dots,M_n} \prod_{i=0}^{n} \mathrm{hom}(M_{i-1}\otimes X_i, M_i \otimes Y_i).$$

Remark 4.3. In other words, n-stage processes are n-tuples $(f_i \colon M_{i-1}\otimes X_i \to M_i \otimes Y_i)_{i=0}^{n}$, for some choice of M_i *up to dinaturality*, that we write as

$$\langle f_0 | f_1 | \dots | f_n | \in \mathsf{Stage}_n(\mathbb{X},\mathbb{Y}).$$

In this notation, *dinaturality* means that morphisms can *slide past the bars*. That is, for any $r_i \colon N_i \to M_i$ and any tuple, dinaturality says that

$$\langle f_0; (r_0\otimes \mathrm{id}) | f_1; (r_1 \otimes \mathrm{id}) | \dots | f_n; (r_n \otimes \mathrm{id})| \\ = \langle f_0 | (r_0 \otimes \mathrm{id}); f_1 | \dots | (r_{n-1}\otimes \mathrm{id}); f_n | .$$

Note that the last r_n is removed by dinaturality.

Definition 4.4 (Truncation)**.** The k*-truncation* of an extensional sequence $\langle f_n \colon M_{n-1}\otimes X_n \to M_n \otimes Y_n\rangle \in \mathsf{Ext}(\mathbb{X},\mathbb{Y})$ is $\langle f_0 | \dots | f_k | \in \mathsf{Stage}_k(\mathbb{X},\mathbb{Y})$. Truncation is well-defined under dinatural equivalence (Remark 4.3).

For $k \leqslant n$, the k*-truncation* of an n-stage process given by $\langle f_0 | f_1 | \dots | f_n | \in \mathsf{Stage}_n(\mathbb{X},\mathbb{Y})$ is $\langle f_0 | \dots | f_k | \in \mathsf{Stage}_k(\mathbb{X},\mathbb{Y})$. This induces functions $\pi_{n,k} \colon \mathsf{Stage}_n(\mathbb{X},\mathbb{Y}) \to \mathsf{Stage}_k(\mathbb{X},\mathbb{Y})$, with the property that $\pi_{n,m};\pi_{m,k} = \pi_{n,k}$.

Definition 4.5 (Observational equivalence)**.** Two extensional stateful sequences

$$\langle f \rangle_{n\in\mathbb{N}}, \langle g \rangle_{n\in\mathbb{N}} \in \int^{M\in[\mathbb{N},\mathsf{C}]} \prod_{i=0}^{\infty} \mathrm{hom}(M_{i-1}\otimes X_i, Y_i \otimes M_i)$$

are *observationally equivalent* when all their n-stage truncations are equal. That is, $\langle f_0 | \dots | f_n | = \langle g_0 | \dots | g_n |$, for each $n \in \mathbb{N}$. We write this as $f \approx g$.

Remark 4.6. Formally, this is to say that the sequences $\langle f \rangle_{n\in\mathbb{N}}$ and $\langle g \rangle_{n\in\mathbb{N}}$ have the same image on the limit

$$\lim_n \mathsf{Stage}_n(\mathbb{X},\mathbb{Y}),$$

over the chain $\pi_{n,k} \colon \mathsf{Stage}_n(\mathbb{X},\mathbb{Y}) \to \mathsf{Stage}_k(\mathbb{X},\mathbb{Y})$.

Definition 4.7. An *observational sequence* from $\mathbb{X}$ to $\mathbb{Y}$ is an equivalence class

$$[\langle f_n \colon M_{n-1}\otimes X_n \to M_n \otimes Y_n \rangle_{n\in\mathbb{N}}]_{\approx}$$

of extensional sequences under observational equivalence. In other words, the set of observational sequences is

$$\mathsf{Obs}(\mathbb{X},\mathbb{Y}) \cong \left(\int^{M\in[\mathbb{N},\mathsf{C}]} \prod_{i=0}^{\infty} \mathrm{hom}(M_{i-1}\otimes X_i, M_i \otimes Y_i) \right) \Big/ \approx$$

4.3 Productive categories

The interaction between extensional and observational equivalence is of particular interest in some well-behaved categories that we call *productive categories*. In productive categories, observational sequences are the final fixpoint of an equation (Theorem 4.11), analogous to that of Section 2.

An important property of programs is *termination*: a terminating program always halts in a finite amount of time. However, some programs (such as servers, drivers) are not actually intended to terminate but to produce infinite output streams. A more appropriate notion in these cases is that of *productivity*: a program that outputs an infinite stream of data is productive if each individual component of the stream is produced in finite time. To quip, *"a productive stream is a terminating first component followed by a productive stream"*.

The first component of our streams is only defined *up to some future*. It is an equivalence class $\alpha \in \mathsf{Stage}_1(\mathbb{X},\mathbb{Y})$, with representatives $\alpha_i \colon X_0 \to M_i \otimes Y_0$. But, if it does terminate, there is a process $\alpha_0 \colon X_0 \to M_0 \otimes Y_0$ in our theory representing the process just until Y_0 is output.

Definition 4.8 (Terminating component)**.** A 1-stage process $\alpha \in \mathsf{Stage}_1(\mathbb{X},\mathbb{Y})$ is *terminating relative to* C if there exists $\alpha_0 \colon X_0 \to M_0 \otimes Y_0$ such that each one of its representatives, $\langle \alpha_i | = \alpha$, can be written as $\alpha_i = \alpha_0; (s_i \otimes \mathrm{id})$ for some $s_i \colon M_0 \to M_i$.

The morphisms s_i represent what is unique to each representative, and so we ask that, for any $u \colon M_0 \otimes A \to U \otimes B$ and $v \colon M_0 \otimes A \to V \otimes B$, the equality $\langle \alpha_i \otimes \mathrm{id}_A; u \otimes \mathrm{id}_{Y_0} | = \langle \alpha_j \otimes \mathrm{id}_A; v \otimes \mathrm{id}_{Y_0} |$ implies $\langle s_i \otimes \mathrm{id}_A; u | = \langle s_j \otimes \mathrm{id}_A; v |$.

Definition 4.9 (Productive category)**.** A symmetric monoidal category $(\mathsf{C}, \otimes, I)$ is *productive* when every 1-stage process is terminating relative to C.

Remark 4.10. Cartesian monoidal categories are productive (Proposition C.1). *Markov categories* [30] with *conditionals* and *ranges* are productive (Theorem A.21). Free symmetric monoidal categories and compact closed categories are always productive.

Theorem 4.11. *Observational sequences are the explicit construction of observational streams when the category is productive. More precisely, in a productive category, the final fixpoint of the equation in Figure 9 is given by the set of observational sequences,* Obs.

Proof sketch. The terminal sequence for this final coalgebra is given by $\mathsf{Stage}_n(\mathbb{X},\mathbb{Y})$. In productive categories, we can prove that the limit $\lim_n \mathsf{Stage}_n(\mathbb{X},\mathbb{Y})$ is a fixpoint of the equation in Figure 9 (Lemma D.6). Finally, in productive categories, observational sequences coincide with this limit (Theorem D.7). □

5 The Category of Monoidal Streams

We are ready to construct Stream: the feedback monoidal category of monoidal streams. Let us recast the definitive notion of monoidal stream (Definition 4.1) coinductively.

Definition 5.1. A *monoidal stream* $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ is a triple consisting of

- $M(f) \in \mathsf{Obj}(\mathsf{C})$, the *memory*,
- $\mathsf{now}(f) \in \hom(X_0, M(f) \otimes Y_0)$, the *first action*,
- $\mathsf{later}(f) \in \mathsf{Stream}(M(f) \cdot \mathbb{X}^+, \mathbb{Y}^+)$, the *rest of the action*,

quotiented by dinaturality in M.

Explicitly, monoidal streams are quotiented by the equivalence relation $f \sim g$ generated by

- the existence of $r \colon M(g) \to M(f)$,
- such that $\mathsf{now}(f) = \mathsf{now}(g); r$,
- and such that $r \cdot \mathsf{later}(f) \sim \mathsf{later}(g)$.

Here, $r \cdot \mathsf{later}(f) \in \mathsf{Stream}(M(g) \cdot \mathbb{X}^+, \mathbb{Y}^+)$ is obtained by precomposition of the first action of $\mathsf{later}(f)$ with r.

Remark 5.2. This is a coinductive definition of the functor

$$\mathsf{Stream} \colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathsf{Set}.$$

In principle, arbitrary final coalgebras do not need to exist. Moreover, it is usually difficult to explicitly construct such coalgebras [4]. However, in productive categories, this coalgebra does exist and is constructed by observational sequences. From now on, we reason *coinductively* [48], a style particularly suited for all the following definitions.

5.1 The symmetric monoidal category of streams

The definitions for the operations of sequential and parallel composition are described in two steps. We first define an operation that takes into account an extra *memory channel* (Figure 10); we use this extra generality to strengthen the *coinduction hypothesis*. We then define the desired operation as a particular case of this coinductively defined one.

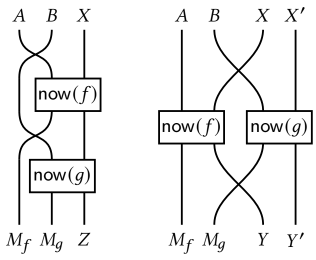


Figure 10. String diagrams for the first action of sequential and parallel composition with memories.

Definition 5.3 (Sequential composition). Given two streams $f \in \mathsf{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$ and $g \in \mathsf{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$, we compute $(f^A ; g^B) \in \mathsf{Stream}((A \otimes B) \cdot \mathbb{X}, \mathbb{Z})$, their *sequential composition with memories A and B*, as

- $M(f^A ; g^B) = M(f) \otimes M(g)$,
- $\mathsf{now}(f^A ; g^B) = \sigma ; (\mathsf{now}(f) \otimes \mathrm{id}) ; \sigma ; (\mathsf{now}(g) \otimes \mathrm{id})$,
- $\mathsf{later}(f^A ; g^B) = \mathsf{later}(f)^{M(f)} ; \mathsf{later}(g)^{M(g)}$.

We write $(f ; g)$ for $(f^I ; g^I) \in \mathsf{Stream}(\mathbb{X}, \mathbb{Z})$; the *sequential composition* of $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ and $g \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$.

Definition 5.4. The *identity* $\mathrm{id}_{\mathbb{X}} \in \mathsf{Stream}(\mathbb{X}, \mathbb{X})$ is defined by $M(\mathrm{id}_{\mathbb{X}}) = I$, $\mathsf{now}(\mathrm{id}_{\mathbb{X}}) = \mathrm{id}_{X_0}$, and $\mathsf{later}(\mathrm{id}_{\mathbb{X}}) = \mathrm{id}_{\mathbb{X}^+}$.

Definition 5.5 (Parallel composition). Given two streams $f \in \mathsf{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$ and $g \in \mathsf{Stream}(B \cdot \mathbb{X}', \mathbb{Y}')$, we compute $(f^A \otimes g^B) \in \mathsf{Stream}((A \otimes B) \cdot (\mathbb{X} \otimes \mathbb{X}'), \mathbb{Y} \otimes \mathbb{Y}')$, their *parallel composition with memories A and B*, as

- $M(f^A \otimes g^B) = M(f) \otimes M(g)$,
- $\mathsf{now}(f^A \otimes g^B) = \sigma; (\mathsf{now}(f) \otimes \mathsf{now}(g)); \sigma$,
- $\mathsf{later}(f^A \otimes g^B) = \mathsf{later}(f)^{M(f)} \otimes \mathsf{later}(g)^{M(g)}$.

We write $(f \otimes g)$ for $(f^I \otimes g^I) \in \mathsf{Stream}(\mathbb{X} \otimes \mathbb{X}', \mathbb{Y} \otimes \mathbb{Y}')$; we call it the *parallel composition* of $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ and $g \in \mathsf{Stream}(\mathbb{X}', \mathbb{Y}')$.

Definition 5.6 (Memoryless and constant streams). Each sequence $\mathbb{f} = (f_0, f_1, \ldots)$, with $f_n \colon X_n \to Y_n$, induces a stream $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ defined by $M(f) = I$, $\mathsf{now}(f) = f_0$, and $\mathsf{later}(f) = \mathbb{f}^+$. Streams of this form are called *memoryless*, i.e. their memories are given by the monoidal unit.

Moreover, each morphism $f_0 \colon X \to Y$ induces a *constant* memoryless stream that we also call $f \in \mathsf{Stream}(X, Y)$, defined by $M(f) = I$, $\mathsf{now}(f) = f_0$, and $\mathsf{later}(f) = f$.

Theorem 5.7. *Monoidal streams over a productive symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a symmetric monoidal category* Stream *with a symmetric monoidal identity-on-objects functor from* $[\mathbb{N}, \mathsf{C}]$.

Proof. Appendix, Theorem E.5. □

5.2 Delayed feedback for streams

Monoidal streams form a *delayed feedback* monoidal category. Given some stream in $\mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, we can create a new stream in $\mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ that passes the output in $\mathbb{S}$ as a memory channel that gets used as the input in $\partial\mathbb{S}$. As a consequence, the category of monoidal streams has a graphical calculus given by that of feedback monoidal categories. This graphical calculus is complete for extensional equivalence (as we saw in Theorem 3.8).

Definition 5.8 (Delay functor). The functor from Definition 3.6 can be lifted to a monoidal functor $\partial \colon \mathsf{Stream} \to \mathsf{Stream}$ that acts on objects in the same way. It acts on morphisms by sending a stream $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ to the stream given by $M(\partial f) = I$, $\mathsf{now}(\partial f) = \mathrm{id}_I$ and $\mathsf{later}(\partial f) = f$.

Definition 5.9 (Feedback operation). Given any morphism of the form $f \in \mathsf{Stream}(N \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, we define $\mathsf{fbk}(f^N) \in \mathsf{Stream}(N \cdot \mathbb{X}, \mathbb{Y})$ as

- $M(\mathsf{fbk}(f^N)) = M(f) \otimes S_0$,
- $\mathsf{now}(\mathsf{fbk}(f^N)) = \mathsf{now}(f)$ and

- $\mathsf{later}(\mathrm{fbk}(f^N)) = \mathrm{fbk}(\mathsf{later}(f)^{M(f)\otimes S_0})$.

We write $\mathrm{fbk}(f) \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ for $\mathrm{fbk}(f^I)$, the feedback of $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$

Theorem 5.10. *Monoidal streams over a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ form a ∂-feedback monoidal category $(\mathsf{Stream}, \mathrm{fbk})$.*

Proof. Appendix, Theorem E.16. □

Corollary 5.11. *There is a "semantics" identity-on-objects feedback monoidal functor $\mathrm{Sm}\colon \mathsf{St}_\partial[\mathbb{N}, \mathsf{C}] \to \mathsf{Stream}$ from the free ∂-feedback monoidal category to the category of monoidal streams. Every extensional stateful sequence $\langle f_n \colon M_{n-1} \otimes X_n \to Y_n \otimes M_n \rangle_{n\in\mathbb{N}}$ gives a monoidal stream $\mathrm{Sm}(f)$, which is defined by $M(\mathrm{Sm}(f)) = M_0$,*

$$\mathsf{now}(\mathrm{Sm}(f)) = f_0, \text{ and } \mathsf{later}(\mathrm{Sm}(f)) = \mathrm{Sm}(f^+),$$

and this is well-defined. Moreover, this functor is full when C is productive; it is not generally faithful.

Proof. We construct Sm from Theorems 3.5 and 5.10. Moreover, when C is productive, by Theorem 4.11, monoidal streams are extensional sequences quotiented by observational equivalence, giving the fullness of the functor. □

6 Cartesian Streams

Dataflow languages such as Lucid or Lustre [35, 77] can be thought of as using an underlying cartesian monoidal structure: we can copy and discard variables and resources without affecting the order of operations. These abilities correspond exactly to cartesianity thanks to Fox's theorem (Theorem C.5, see [29]).

6.1 Causal stream functions

In the cartesian case, there is available literature on the categorical semantics of dataflow programming languages [8, 24, 25, 58, 76]. Uustalu and Vene [75] provide elegant comonadic semantics to a Lucid-like programming language using the non-empty list comonad. In their framework, *streams* with types $\mathbb{X} = (X_0, X_1, \ldots)$ are families of elements $1 \to X_n$. *Causal stream functions* from $\mathbb{X} = (X_0, X_1, \ldots)$ to $\mathbb{Y} = (Y_0, Y_1, \ldots)$ are families of functions $f_n \colon X_0 \times \cdots \times X_n \to Y_n$. Equivalently, they are, respectively, the states $(\mathbb{1} \to \mathbb{X})$ and morphisms $(\mathbb{X} \to \mathbb{Y})$ of the cokleisli category of the comonad $\mathsf{List}^+\colon [\mathbb{N}, \mathsf{Set}] \to [\mathbb{N}, \mathsf{Set}]$ defined by

$$(\mathsf{List}^+(\mathbb{X}))_n \coloneqq \prod_{i=0}^{n} X_i.$$

This comonad can be extended to other base categories, $\mathsf{List}^+\colon [\mathbb{N}, \mathsf{C}] \to [\mathbb{N}, \mathsf{C}]$ *only* as long as C is cartesian. Indeed, we can prove that the mere existence of such a comonad implies cartesianity of the base category. For this, we introduce a refined version of Fox's theorem (Theorem C.7).

Theorem 6.1. *Let $(\mathsf{C}, \otimes, I)$ be a symmetric monoidal category. Let $\mathsf{List}^+\colon [\mathbb{N}, \mathsf{C}] \to [\mathbb{N}, \mathsf{C}]$ be the functor defined by*

$$\mathsf{List}^+(X)_n \coloneqq \bigotimes_{i=0}^{n} X_i.$$

This functor is monoidal, with oplaxators $\psi_0^+\colon \mathsf{List}^+(I) \to I$ and $\psi_{X,Y}\colon \mathsf{List}^+(X \otimes Y) \to \mathsf{List}^+(X) \otimes \mathsf{List}^+(Y)$ given by symmetries, associators and unitors.

The monoidal functor $\mathsf{List}^+\colon [\mathbb{N}, \mathsf{C}] \to [\mathbb{N}, \mathsf{C}]$ has a monoidal comonad structure if and only if its base monoidal category $(\mathsf{C}, \otimes, I)$ is cartesian monoidal.

Proof sketch. The cartesian structure can be shown to make List^+ an opmonoidal comonad. Conversely, the opmonoidal comonad structure implies that every object should have a natural and uniform counital comagma structure. By our refined statement of Fox's theorem (Theorem C.7), this implies cartesianity. See Appendix, Theorem C.4. □

This means that we cannot directly extend Uustalu and Vene's approach to the monoidal case. However, we prove in the next section that our definition of monoidal streams particularizes to their *causal stream functions* [73, 75].

6.2 Cartesian monoidal streams

The main claim of this section is that, in a cartesian monoidal category, monoidal streams instantiate to *causal stream functions* (Theorem 6.3). Let us fix such a category, $(\mathsf{C}, \times, 1)$.

The first observation is that the universal property of the cartesian product simplifies the fixpoint equation that defines monoidal streams. This is a consequence of the following chain of isomorphisms, where we apply a Yoneda reduction to simplify the coend.

$$\begin{aligned}
&\mathsf{Stream}(\mathbb{X}, \mathbb{Y}) \cong \\
&\int^M \hom(X_0, M \times Y_0) \times \mathsf{Stream}(M \cdot \mathbb{X}^+, \mathbb{Y}) \cong \\
&\int^M \hom(X_0, M) \times \hom(X_0, Y_0) \times \mathsf{Stream}(M \cdot \mathbb{X}^+, \mathbb{Y}) \cong \\
&\hom(X_0, Y_0) \times \mathsf{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+).
\end{aligned}$$

Explicitly, the Yoneda reduction works as follows: the first action of a stream $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ can be uniquely split as $\mathsf{now}(f) = (f_1, f_2)$ for some $f_1 \colon X_0 \to Y_0$ and $f_2 \colon X_0 \to M(f)$. Under the *dinaturality* equivalence relation, $(\sim)$, we can always find a unique representative with $M = X_0$ and $f_2 = \mathrm{id}_{X_0}$.

The definition of monoidal streams in the cartesian case is thus simplified (Definition 6.2). From there, the explicit construction of cartesian monoidal streams is straightforward.

Definition 6.2 (Cartesian monoidal streams). The set of *cartesian monoidal streams*, given inputs $\mathbb{X}$ and outputs $\mathbb{Y}$, is the terminal fixpoint of the equation

$$\mathsf{Stream}(\mathbb{X}, \mathbb{Y}) \cong \hom(X_0, Y_0) \times \mathsf{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

In other words, a cartesian monoidal stream $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$ is a pair consisting of

- $\mathsf{fst}(f) \in \mathrm{hom}(X_0, Y_0)$, the *first action*, and
- $\mathsf{snd}(f) \in \mathsf{Stream}(X_0 \cdot \mathbb{X}^+, \mathbb{Y}^+)$, the *rest of the action*.

Theorem 6.3. *In the cartesian case, the final fixpoint of the equation in Figure 9 is given by the set of causal functions,*

$$\mathsf{Stream}(\mathbb{X}, \mathbb{Y}) = \prod_{n \in \mathbb{N}}^{\infty} \mathrm{hom}(X_0 \times \cdots \times X_n, Y_n).$$

That is, the category Stream *of monoidal streams coincides with the cokleisli monoidal category of the non-empty list monoidal comonad* $\mathsf{List}^+ \colon [\mathbb{N}, \mathsf{C}] \to [\mathbb{N}, \mathsf{C}]$.

Proof. By Adamek's theorem (Theorem 1.5). □

Corollary 6.4. *Let* $(\mathsf{C}, \times, 1)$ *be a cartesian monoidal category. The category* Stream *is cartesian monoidal.*

6.3 Example: the Fibonacci sequence

Consider $(\mathsf{Set}, \times, 1)$, the cartesian monoidal category of small sets and functions. And let us go back to the morphism $\mathsf{fib} \in \mathsf{Stream}(1, \mathbb{N})$ that we presented in the Introduction (Figure 2). By Theorem 6.3, a morphism of this type is, equivalently, a sequence of natural numbers. Using the previous definitions in Sections 5 and 6, we can explicitly compute this sequence to be $\mathsf{fib} = [0, 1, 1, 2, 3, 5, 8, \dots]$ (see the Appendix, Example G.1).

7 Stochastic Streams

Monoidal categories are well suited for reasoning about probabilistic processes. Several different categories of probabilistic channels have been proposed in the literature [6, 19, 59]. They were largely unified by Fritz [30] under the name of *Markov categories*. For simplicity, we work in the discrete stochastic setting, i.e. in the Kleisli category of the finite distribution monad, Stoch, but we will be careful to isolate the relevant structure of Markov categories that we use.

The main result of this section is that *controlled stochastic processes* [27, 66] are precisely monoidal streams over Stoch. That is, controlled stochastic processes are the canonical solution over Stoch of the fixpoint equation in Figure 9.

7.1 Stochastic processes

We start by recalling the notion of *stochastic process* from probability theory and its "controlled" version. The latter is used in the context of *stochastic control* [27, 66], where the user has access to the parameters or optimization variables of a probabilistic model.

A *discrete stochastic process* is defined as a collection of random variables $Y_1, \dots Y_n$ indexed by discrete time. At any time step n, these random variables are distributed according to some $p_n \in \mathrm{D}(Y_1 \times \cdots \times Y_n)$. Since the future cannot influence the past, the marginal of p_{n+1} over Y_{n+1} must equal p_n. When this occurs, we say that the family of distributions $(p_n)_{n \in \mathbb{N}}$ is *causal.*

More generally, there may be some additional variables $X_1, \dots, X_n$ which we have control over. In this case, a *controlled stochastic process* is defined as a collection of *controlled random variables* distributing according to $f_n \colon X_1 \times \cdots \times X_n \to \mathrm{D}(Y_1, \dots, Y_n)$. Causality ensures that the marginal of f_{n+1} over Y_{n+1} must equal f_n.

Definition 7.1. Let $\mathbb{X}$ and $\mathbb{Y}$ be sequences of sets. A *controlled stochastic process* $f \colon \mathbb{X} \to \mathbb{Y}$ is a sequence of functions

$$f_n \colon X_n \times \cdots \times X_1 \to \mathrm{D}(Y_n \times \cdots \times Y_1)$$

satisfying *causality* (the *marginalisation property*). That is, such that f_n coincides with the marginal distribution of f_{n+1} on the first n variables, making the diagram in Figure 11 commute.

$$\begin{tikzcd}
X_0 \times \cdots \times X_{n+1} \arrow[r, "f_{n+1}"] \arrow[d, "\pi_{0,\dots,n}"'] & D(Y_0 \times \cdots \times Y_{n+1}) \arrow[d, "D\pi_{0,\dots,n}"] \\
X_0 \times \cdots \times X_n \arrow[r, "f_n"] & D(Y_0 \times \cdots \times Y_n)
\end{tikzcd}$$

Figure 11. Marginalisation for stochastic processes.

Controlled stochastic processes with componentwise composition, identities and tensoring, are the morphisms of a symmetric monoidal category StochProc.

Stochastic monoidal streams and stochastic processes not only are the same thing but they compose in the same way: they are *isomorphic* as categories.

Theorem 7.2. *The category of stochastic processes* StochProc *is monoidally isomorphic to the category* Stream *over* Stoch.

Proof sketch. Appendix, Theorem A.29. The proof of this result is non-trivial and relies on a crucial property concerning ranges in Stoch. The proof is moreover written in the language of Markov categories where the property of ranges can be formulated in full abstraction. □

We expect that the theorem above can be generalised to interesting categories of probabilistic channels over measurable spaces (such as the ones covered in [19, 30, 59]).

Corollary 7.3. StochProc *is a feedback monoidal category.*

7.2 Examples

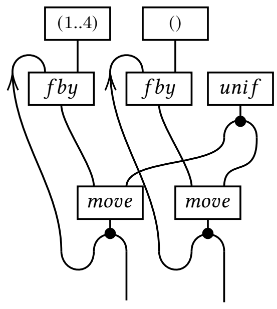
We have characterized in two equivalent ways the notion of controlled stochastic process. This yields a categorical semantics for probabilistic dataflow programming: we may use the syntax of feedback monoidal categories to specify simple stochastic programs and evaluate their semantics in StochProc.

Example 7.4 (Random Walk). Recall the morphism $\mathsf{walk} \in \mathsf{Stream}(1, \mathbb{Z})$ that we depicted back in Figure 4.

Here, $\mathsf{unif} \in \mathsf{Stream}(1, \{-1, 1\})$, is a uniform random generator that, at each step, outputs either 1 or (-1). The output of this uniform random generator is then added to the current position, and we declare the starting position to be 0. Our implementation of this morphism, following the definitions from Section 5 (Example G.2) is, by Theorem 7.2, a discrete stochastic process, and it produces samples like the following ones.

$$[0, 1, 0, -1, -2, -1, -2, -3, -2, -3, \dots]$$
$$[0, 1, 2, 1, 2, 1, 2, 3, 4, 5, \dots]$$
$$[0, -1, -2, -1, -2, -1, 0, -1, 0, -1, \dots]$$

Example 7.5 (Ehrenfest model). The Ehrenfest model [47, §1.4] is a simplified model of particle diffusion.



ehr =
$(1...4) \otimes (); \mathsf{fbk}(\sigma;$
$\mathsf{fby} \otimes \mathsf{fby} \otimes \mathsf{unif};$
$\mathrm{id} \otimes \mathrm{id} \otimes \mathsf{copy}; \sigma;$
$\mathsf{move} \otimes \mathsf{move};$
$\mathsf{copy})$

Figure 12. Ehrenfest model: sig. flow graph and morphism.

Assume we have two urns with 4 balls, labelled from 1 to 4. Initially, the balls are all in the first urn. We randomly (and uniformly) pick an integer from 1 to 4, and the ball labelled by that number is removed from its box and placed in the other box. We iterate the procedure, with independent uniform selections each time.

Our implementation of this morphism, following the definitions from Section 5 (Example G.3) yields samples such as the following.

$$[([2,3,4],[1]),\quad ([3,4],[1,2]),\quad ([1,3,4],[2]),$$
$$([1,4],[2,3]),\quad ([1],[2,3,4]),\quad ([],[1,2,3,4]),$$
$$([2],[1,3,4]),\quad \dots]$$

8 A dataflow programming language

In this section, we introduce the syntax for two Lucid-like dataflow programming languages and their semantics in monoidal streams. The first one is *deterministic* and it takes semantics in the feedback monoidal category of set-based monoidal streams. The second one is *stochastic* and it takes semantics in the feedback monoidal category of stochastic processes, or stochastic monoidal streams.

We do so by presenting a type theory for *feedback* monoidal categories (similar to [36, 71]). Terms of the type theory represent programs in our language.

8.1 Type theory for monoidal categories

We start by considering a *type theory for symmetric monoidal categories* over some generators forming a multigraph $\mathcal{G}$. Instead of presenting a type theory from scratch, we extend the basic type theory for symmetric monoidal categories described by Shulman [70]. Details are in the Appendix (Appendix F). Here, we only illustrate it with an example.

Gen
$$\frac{f \in \mathcal{G}(A_1, \dots, A_n; B) \qquad \Gamma_1 \vdash x_1 : A_1 \dots \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1, \dots, \Gamma_n) \vdash f(x_1, \dots, x_n) : B}$$

Pair
$$\frac{\Gamma_1 \vdash x_1 : A_1 \ \dots \ \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1, ..., \Gamma_n) \vdash [x_1, ..., x_n] : A_1 \otimes ... \otimes A_n}$$

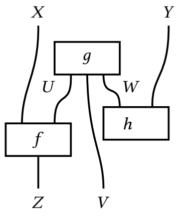
Var
$$\frac{}{x : A \vdash x : A}$$

Split
$$\frac{\Delta \vdash m : A_1 \otimes \cdots \otimes A_n \qquad \Gamma, x_1 : A_1, \dots, x_n : A_n \vdash z : C}{\mathrm{Shuf}(\Gamma, \Delta) \vdash \textsc{Split}\ m \to [x_1, \dots, x_n]\ \textsc{in}\ z : C}$$

Figure 13. Type theory of symm. monoidal categories [70].

The type theory for symmetric monoidal categories is *linear* [34, 52, 68] in the sense that any introduced variable must be used exactly once. This is for a good reason: monoidal categories represent linear theories of processes, where *copying* and *discarding* may not be allowed in general.

Example 8.1. In a monoidal category, let $f : X \otimes U \to Z$, $g : I \to U \otimes V \otimes W$ and $h : V \otimes Y \to I$. The following is a string diagram together with its term in the type theory.



$\textsc{Split}\ g \to [u, v, w]\ \textsc{in}$
$\textsc{Split}\ h(w, y) \to []\ \textsc{in}$
$[f(x, u), v]$

8.2 Adding feedback

We now extend the theory with delay and feedback. We start by considering a ∂ operator on types, which extends to contexts inductively as $\partial[] = []$ and $\partial(\Gamma, x{:}A) = \partial\Gamma, (x : \partial A)$. We provide formation rules for introducing delay and feedback. These need to satisfy equalities making Delay a functor and Fbk a feedback operator.

Delay
$$\frac{\Gamma \vdash x : A}{\partial\Gamma \vdash x : \partial A}$$

Fbk
$$\frac{\Gamma, s : \partial S \vdash x(s) : S \otimes A}{\Gamma \vdash \textsc{Fbk}\ s.\ x(s) : A}$$

As in Remark 3.2, we define $\textsc{Wait}(x) \equiv \textsc{Fbk}\ y\ \textsc{in}\ [x, y]$.

8.3 Adding generators

In both versions of the language (deterministic and stochastic), we include "copy" and "followed by" operations, representing the corresponding monoidal streams. Copying does not need to be natural (in the stochastic case, it will not be) and it does not even need to form a comonoid.

$$\textsc{Copy}\ \frac{\Gamma \vdash x : A}{\Gamma \vdash \textsc{Copy}(x) : A \otimes A} \qquad \textsc{Fby}\ \frac{\Gamma \vdash x : A \qquad \Delta \vdash y : \partial(A)}{\mathrm{Shuf}(\Gamma, \Delta) \vdash x \ \textsc{Fby}\ y : A}$$

In fact, recursive definitions make sense only when we have a *copy* operation, that allows us to rewrite the definition as a feedback that ends with a copy. That is,

$$M = x(M) \quad \text{means} \quad M = \textsc{Fbk}\ m \ \textsc{in}\ \textsc{Copy}(x(m)).$$

Moreover, in the deterministic version of our language we allow non-linearity: a variable can occur multiple times, implicitly copying it.

8.4 Examples

Example 8.2. Recall the example from the introduction (and Section 6.3).

$$fib = 0\ \textsc{Fby}\ (fib + (1\ \textsc{Fby}\ \textsc{Wait}\ fib))$$

Its desugaring, following the previous rules, is below.

$$\begin{aligned} fib = {} & \textsc{Fbk}\ f\ \textsc{in}\ \textsc{Copy} \\ & \quad (0\ \textsc{Fby} \\ & \qquad \textsc{Split}\ \textsc{Copy}(f) \longrightarrow [f_1, f_2]\ \textsc{in} \\ & \qquad (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2))\ ) \end{aligned}$$

Example 8.3 (Ehrenfest model). The Ehrenfest model described in Figure 12 has the following specification in the programming language.

$$\begin{aligned} urns = {} & [(1,2,3,4),()]\ \textsc{Fby} \\ & \textsc{Split}\ urns \longrightarrow [u_1, u_2]\ \textsc{in} \\ & \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \longrightarrow [n_1, n_2]\ \textsc{in} \\ & [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] \end{aligned}$$

Sampling twice from the same distribution is different from copying a single sample, and Split allows us to express this difference: instead of calling the Uniform distribution twice, this program calls it once and then copies the result.

9 Conclusions

Monoidal streams are a common generalization of streams, causal functions and stochastic processes. In the same way that streams give semantics to dataflow programming [35, 77] with plain functions, monoidal streams give semantics to dataflow programming with monoidal theories of processes. Signal flow graphs are a common tool to describe control flow in dataflow programming. Signal flow graphs are also the natural string diagrams of feedback monoidal categories. Monoidal streams form a feedback monoidal category, and signal flow graphs are a formal syntax to describe and reason about monoidal streams. The second syntax we present comes from the type theory of monoidal categories, and it is inspired by the original syntax of dataflow programming. We have specifically studied stochastic dataflow programming, but the same framework allows for *linear*, *quantum* and *effectful* theories of resources.

The literature on dataflow and feedback is rich enough to provide multiple diverging definitions and approaches. What we can bring to this discussion are universal constructions. Universal constructions justify some mathematical object as *the canonical object* satisfying some properties. In our case, these exact properties are extracted from three, arguably under-appreciated, but standard category-theoretic tools: *dinaturality*, *feedback*, and *coalgebra*. *Dinaturality*, profunctors and coends, sometimes regarded as highly theoretical developments, are the natural language to describe how processes communicate and compose. *Feedback*, sometimes eclipsed by trace in the mathematical literature, keeps appearing in multiple variants across computer science. *Coalgebra* is the established tool to specify and reason about stateful systems.

9.1 Further work

Other theories. Many interesting examples of theories of processes are not monoidal but just *premonoidal categories* [42, 62]. For instance, the kleisli categories of arbitrary monads, where effects (e.g. reading and writing to a global state) do not need to commute. Premonoidal streams can be constructed by restricting dinaturality to their *centres*. Another important source of theories of processes that we have not covered is that of *linearly distributive* and **-autonomous categories* [11, 12, 22, 68].

Within monoidal categories, we would like to make monoidal streams explicit in the cases of partial maps [21] for dataflow programming with different clocks [76], nondeterministic maps [17, 51] and quantum processes [18]. A final question we do not pursue here is *expressivity*: the class of functions a monoidal stream can define.

The 2-categorical view. We describe the morphisms of a category as a final coalgebra. However, it is also straightforward to describe the 2-endofunctor that should give rise to this category as a final coalgebra itself.

Implementation of the type theory. Justifying that the output of monoidal streams is the expected one requires some computations, which we have already implemented separately in the Haskell programming language (Appendix, Appendix G). Agda has similar foundations and supports the coinductive definitions of this text (Section 5). It is possible to implement a whole interpreter for a Lucid-like stochastic programming language with a dedicated parser, but that requires some software engineering effort that we postpone for further work.

References

[1] Martín Abadi and Michael Isard. Timely Dataflow: A Model. In Susanne Graf and Mahesh Viswanathan, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 35th IFIP WG 6.1 International Conference, FORTE 2015, Held as Part of the 10th International Federated Conference on Distributed Computing Techniques, DisCoTec 2015, Grenoble, France, June 2-4, 2015, Proceedings*, volume 9039 of *Lecture Notes in Computer Science*, pages 131–145. Springer, 2015. doi:10.1007/978-3-319-19195-9_9.

[2] Samson Abramsky and Bob Coecke. Categorical quantum mechanics. *Handbook of quantum logic and quantum structures*, 2:261–325, 2009. arXiv:0808.1023.

[3] Jiří Adámek. Introduction to coalgebra. *Theory and Applications of Categories*, 14(8):157–199, 2005.

[4] Jiří Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015(4):589–602, 1974. URL: http://eudml.org/doc/16649.

[5] Jiří Adámek. On Terminal Coalgebras Derived from Initial Algebras. In Markus Roggenbach and Ana Sokolova, editors, *8th Conference on Algebra and Coalgebra in Computer Science, CALCO 2019, June 3-6, 2019, London, United Kingdom*, volume 139 of *LIPIcs*, pages 12:1–12:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CALCO.2019.12.

[6] John C. Baez, Brendan Fong, and Blake S. Pollard. A Compositional Framework for Markov Processes. *Journal of Mathematical Physics*, 57(3):033301, March 2016. arXiv:1508.06448, doi:10.1063/1.4941578.

[7] Jon Beck. Distributive laws. In *Seminar on triples and categorical homology theory*, pages 119–140. Springer, 1969.

[8] Albert Benveniste, Paul Caspi, Paul Le Guernic, and Nicolas Halbwachs. Data-flow synchronous languages. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency, Reflections and Perspectives, REX School/Symposium, Noordwijkerhout, The Netherlands, June 1-4, 1993, Proceedings*, volume 803 of *Lecture Notes in Computer Science*, pages 1–45. Springer, 1993. doi:10.1007/3-540-58043-3_16.

[9] Lars Birkedal, Rasmus Ejlers Møgelberg, Jan Schwinghammer, and Kristian Støvring. First steps in synthetic guarded domain theory: Step-indexing in the topos of trees. In *Proceedings of the 26th Annual IEEE Symposium on Logic in Computer Science, LICS 2011, June 21-24, 2011, Toronto, Ontario, Canada*, pages 55–64. IEEE Computer Society, 2011. doi:10.1109/LICS.2011.16.

[10] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993. doi:10.1007/978-3-642-78034-9.

[11] Richard Blute. Linear logic, coherence, and dinaturality. *Theor. Comput. Sci.*, 115(1):3–41, 1993. doi:10.1016/0304-3975(93)90053-V.

[12] Richard F Blute, J Robin B Cockett, Robert AG Seely, and Todd H Trimble. Natural deduction and coherence for weakly distributive categories. *Journal of Pure and Applied Algebra*, 113(3):229–296, 1996.

[13] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019. doi:10.1145/3290338.

[14] Filippo Bonchi, Jens Seeber, and Paweł Sobociński. Graphical conjunctive queries. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.CSL.2018.13.

[15] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. A categorical semantics of signal flow graphs. In *International Conference on Concurrency Theory*, pages 435–450. Springer, 2014.

[16] Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. Full abstraction for signal flow graphs. *ACM SIGPLAN Notices*, 50(1):515–526, 2015.

[17] Manfred Broy and Gheorghe Ştefănescu. The algebra of stream processing functions. *Theoretical Computer Science*, 258(1-2):99–129, 2001.

[18] 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*, pages 1–13. IEEE, 2021. doi:10.1109/LICS52264.2021.9470553.

[19] 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.

[20] J. Robin B. Cockett, Xiuzhan Guo, and Pieter Hofstra. Range Categories I: General theory. *Theory and Applications of Categories*, 26(17):412–452, 2012.

[21] J. Robin B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1-2):223–259, 2002. doi:10.1016/S0304-3975(00)00382-0.

[22] J Robin B Cockett and Robert AG Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

[23] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Inf. Comput.*, 250:59–86, 2016. doi:10.1016/j.ic.2016.02.008.

[24] Patrick Cousot. Syntactic and semantic soundness of structural dataflow analysis. In Bor-Yuh Evan Chang, editor, *Static Analysis - 26th International Symposium, SAS 2019, Porto, Portugal, October 8-11, 2019, Proceedings*, volume 11822 of *Lecture Notes in Computer Science*, pages 96–117. Springer, 2019. doi:10.1007/978-3-030-32304-2_6.

[25] Antonin Delpeuch. A complete language for faceted dataflow programs. In John Baez and Bob Coecke, editors, *Proceedings Applied Category Theory 2019, ACT 2019, University of Oxford, UK, 15-19 July 2019*, volume 323 of *EPTCS*, pages 1–14, 2019. doi:10.4204/EPTCS.323.1.

[26] Elena Di Lavore, Alessandro Gianola, Mario Román, Nicoletta Sabadini, and Paweł Sobociński. A canonical algebra of open transition systems. In Gwen Salaün and Anton Wijs, editors, *Formal Aspects of Component Software*, pages 63–81, Cham, 2021. Springer International Publishing.

[27] Wendell Helms Fleming and Raymond W. Rishel. *Deterministic and Stochastic Optimal Control*. Number vol 1 in Applications of Mathematics. Springer-Verlag, Berlin ; New York, 1975.

[28] Brendan Fong and David I Spivak. Supplying bells and whistles in symmetric monoidal categories. *arXiv preprint arXiv:1908.02633*, 2019.

[29] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976.

[30] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. URL: http://arxiv.org/abs/1908.07021, arXiv:1908.07021.

[31] Richard Garner. Stream processors and comodels. *arXiv preprint arXiv:2106.05473*, 2021.

[32] Simon J. Gay and Rajagopal Nagarajan. Intensional and extensional semantics of dataflow programs. *Formal Aspects Comput.*, 15(4):299–318, 2003. doi:10.1007/s00165-003-0018-1.

[33] Dan R. Ghica, George Kaye, and David Sprunger. Full abstraction for digital circuits, 2022. arXiv:2201.10456.

[34] Jean-Yves Girard. Linear logic. *Theoretical computer science*, 50(1):1–101, 1987.

[35] Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE. *IEEE Trans. Software Eng.*, 18(9):785–793, 1992. doi:10.1109/32.159839.

[36] Masahito Hasegawa. *Models of sharing graphs: a categorical semantics of let and letrec*. PhD thesis, University of Edinburgh, UK, 1997. URL: http://hdl.handle.net/1842/15001.

[37] Thomas Hildebrandt, Prakash Panangaden, and Glynn Winskel. A relational model of non-deterministic dataflow. In *International Conference on Concurrency Theory*, pages 613–628. Springer, 1998.

[38] Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014. doi:10.1145/2603088.2603124.

[39] Paul Hudak, Simon L. Peyton Jones, Philip Wadler, Brian Boutel, Jon Fairbairn, Joseph H. Fasel, María M. Guzmán, Kevin Hammond, John Hughes, Thomas Johnsson, Richard B. Kieburtz, Rishiyur S. Nikhil, Will Partain, and John Peterson. Report on the Programming Language Haskell, A Non-strict, Purely Functional Language. *ACM SIGPLAN Notices*, 27(5):1, 1992. doi:10.1145/130697.130699.

[40] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. doi:10.1016/S0167-6423(99)00023-4.

[41] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*, volume 59 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2016. doi:10.1017/CBO9781316823187.

[42] Alan Jeffrey. Premonoidal categories and flow graphs. *Electron. Notes Theor. Comput. Sci.*, 10:51, 1997. doi:10.1016/S1571-0661(05)80688-7.

[43] André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447 – 468, 04 1996. doi:10.1017/S0305004100074338.

[44] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.

[45] 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.

[46] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *RAIRO-Theor. Informatics Appl.*, 36(2):181–194, 2002. doi:10.1051/ita:2002009.

[47] Frank P. Kelly. *Reversibility and stochastic networks.* Cambridge University Press, 2011.

[48] Dexter Kozen and Alexandra Silva. Practical coinduction. *Mathematical Structures in Computer Science*, 27(7):1132–1152, 2017. doi:10.1017/S0960129515000493.

[49] J. Lambek. Cartesian closed categories and typed λ-calculi. In Guy Cousineau, Pierre-Louis Curien, and Bernard Robinet, editors, *Combinators and Functional Programming Languages*, Lecture Notes in Computer Science, pages 136–175, Berlin, Heidelberg, 1986. Springer. doi:10.1007/3-540-17184-3_44.

[50] Joachim Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103(2):151–161, 1968.

[51] Edward A. Lee and Eleftherios Matsikoudis. The semantics of dataflow with firing. *From Semantics to Computer Science: Essays in Honour of Gilles Kahn*, pages 71–94, 2009.

[52] Patrick Lincoln and John C. Mitchell. Operational aspects of linear lambda calculus. In *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS '92), Santa Cruz, California, USA, June 22-25, 1992*, pages 235–246. IEEE Computer Society, 1992. doi:10.1109/LICS.1992.185536.

[53] Fosco Loregian. *(Co)end Calculus.* London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. doi:10.1017/9781108778657.

[54] Nancy A. Lynch and Eugene W. Stark. A proof of the Kahn principle for input/output automata. *Information and Computation*, 82(1):81–92, 1989.

[55] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1978. doi:10.1007/978-1-4757-4721-8.

[56] Konstantinos Mamouras. Semantic foundations for deterministic dataflow and stream processing. In Peter Müller, editor, *Programming Languages and Systems - 29th European Symposium on Programming, ESOP 2020, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2020, Dublin, Ireland, April 25-30, 2020, Proceedings*, volume 12075 of *Lecture Notes in Computer Science*, pages 394–427. Springer, 2020. doi:10.1007/978-3-030-44914-8_15.

[57] S. J. Mason. Feedback Theory - Some properties of signal flow graphs. *Proceedings of the Institute of Radio Engineers*, 41(9):1144–1156, 1953. doi:10.1109/JRPROC.1953.274449.

[58] José Nuno Oliveira. *The formal semantics of deterministic dataflow programs.* PhD thesis, University of Manchester, UK, 1984. URL: http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.376586.

[59] Prakash Panangaden. The Category of Markov Kernels. *Electronic Notes in Theoretical Computer Science*, 22:171–187, January 1999. doi:10.1016/S1571-0661(05)80602-4.

[60] Prakash Panangaden and Eugene W. Stark. Computations, residuals, and the power of indeterminacy. In *International Colloquium on Automata, Languages, and Programming*, pages 439–454. Springer, 1988.

[61] Ross Paterson. A new notation for arrows. In Benjamin C. Pierce, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, pages 229–240. ACM, 2001. doi:10.1145/507635.507664.

[62] John Power. Premonoidal categories as categories with algebraic structure. *Theor. Comput. Sci.*, 278(1-2):303–321, 2002. doi:10.1016/S0304-3975(00)00340-6.

[63] John Power and Hayo Thielecke. Closed freyd- and kappa-categories. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, volume 1644 of *Lecture Notes in Computer Science*, pages 625–634. Springer, 1999. doi:10.1007/3-540-48523-6_59.

[64] John Power and Hiroshi Watanabe. Distributivity for a monad and a comonad. In Bart Jacobs and Jan J. M. M. Rutten, editors, *Coalgebraic Methods in Computer Science, CMCS 1999, Amsterdam, The Netherlands, March 20-21, 1999*, volume 19 of *Electronic Notes in Theoretical Computer Science*, page 102. Elsevier, 1999. doi:10.1016/S1571-0661(05)80271-3.

[65] Mario Román. Comb diagrams for discrete-time feedback. *CoRR*, abs/2003.06214, 2020. arXiv:2003.06214.

[66] Sheldon M. Ross. *Stochastic processes*, volume 2. John Wiley & Sons, 1996.

[67] Jan J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249(1):3–80, 2000. doi:10.1016/S0304-3975(00)00056-6.

[68] Robert A.G. Seely. *Linear logic, *-autonomous categories and cofree coalgebras.* Ste. Anne de Bellevue, Quebec: CEGEP John Abbott College, 1987.

[69] Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines.* Bell Telephone Laboratories, 1942.

[70] Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. URL: https://mikeshulman.github.io/catlog/catlog.pdf.

[71] Michael Shulman. A practical type theory for symmetric monoidal categories, 2021. arXiv:1911.00818.

[72] David Sprunger and Bart Jacobs. The differential calculus of causal functions. *CoRR*, abs/1904.10611, 2019. URL: http://arxiv.org/abs/1904.10611, arXiv:1904.10611.

[73] 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, pages 1–12. IEEE, 2019. doi:10.1109/LICS.2019.8785670.

[74] Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical theory of domains*, volume 22 of *Cambridge tracts in theoretical computer science.* Cambridge University Press, 1994.

[75] Tarmo Uustalu and Varmo Vene. The essence of dataflow programming. In Kwangkeun Yi, editor, *Programming Languages and Systems, Third Asian Symposium, APLAS 2005, Tsukuba, Japan, November 2-5, 2005, Proceedings*, volume 3780 of *Lecture Notes in Computer Science*, pages 2–18. Springer, 2005. doi:10.1007/11575467_2.

[76] Tarmo Uustalu and Varmo Vene. Comonadic notions of computation. In Jiří Adámek and Clemens Kupke, editors, *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, volume 203 of *Electronic Notes in Theoretical Computer Science*, pages 263–284. Elsevier, 2008. doi:10.1016/j.entcs.2008.05.029.

[77] William W Wadge, Edward A Ashcroft, et al. *Lucid, the dataflow programming language*, volume 303. Academic Press London, 1985.

A Monoidal categories

Definition A.1 ([55]). A **monoidal category**,

$$(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho),$$

is a category C equipped with a functor $\otimes \colon \mathsf{C} \times \mathsf{C} \to \mathsf{C}$, a unit $I \in \mathsf{C}$, and three natural isomorphisms: the associator $\alpha_{A,B,C} \colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, the left unitor $\lambda_A \colon I \otimes A \cong A$ and the right unitor $\rho_A \colon A \otimes I \cong A$; such that $\alpha_{A,I,B} ; (\mathrm{id}_A \otimes \lambda_B) = \rho_A \otimes \mathrm{id}_B$ and $(\alpha_{A,B,C} \otimes \mathrm{id}) ; \alpha_{A,B \otimes C,D} ; (\mathrm{id}_A \otimes \alpha_{B,C,D}) = \alpha_{A \otimes B,C,D} ; \alpha_{A,B,C \otimes D}$. A monoidal category is *strict* if α, λ and ρ are identities.

Definition A.2 (Monoidal functor, [55]). Let

$$(\mathsf{C}, \otimes, I, \alpha^{\mathsf{C}}, \lambda^{\mathsf{C}}, \rho^{\mathsf{C}}) \text{ and } (\mathsf{D}, \boxtimes, J, \alpha^{\mathsf{D}}, \lambda^{\mathsf{D}}, \rho^{\mathsf{D}})$$

be monoidal categories. A *monoidal functor* (sometimes called *strong monoidal functor*) is a triple (F, ε, μ) consisting of a functor $F \colon \mathsf{C} \to \mathsf{D}$ and two natural isomorphisms $\varepsilon \colon J \cong F(I)$ and $\mu \colon F(A \otimes B) \cong F(A) \boxtimes F(B)$; such that

- the associators satisfy
$$\begin{aligned} &\alpha^{\mathsf{D}}_{FA,FB,FC} ; (\mathrm{id}_{FA} \otimes \mu_{B,C}) ; \mu_{A,B \otimes C} \\ &= (\mu_{A,B} \otimes \mathrm{id}_{FC}) ; \mu_{A \otimes B,C} ; F(\alpha^{\mathsf{C}}_{A,B,C}), \end{aligned}$$
- the left unitor satisfies
$$(\varepsilon \otimes \mathrm{id}_{FA}) ; \mu_{I,A} ; F(\lambda^{\mathsf{C}}_A) = \lambda^{\mathsf{D}}_{FA}$$
- the right unitor satisfies
$$(\mathrm{id}_{FA} \otimes \varepsilon) ; \mu_{A,I} ; F(\rho^{\mathsf{C}}_{FA}) = \rho^{\mathsf{D}}_{FA}.$$

A monoidal functor is a *monoidal equivalence* if it is moreover an equivalence of categories. Two monoidal categories are monoidally equivalent if there exists a monoidal equivalence between them.

During most of the paper, we omit all associators and unitors from monoidal categories, implicitly using the *coherence theorem* for monoidal categories (Remark A.4).

Theorem A.3 (Coherence theorem, [55]). *Every monoidal category is monoidally equivalent to a strict monoidal category.*

Remark A.4. Let us comment further on how we use the coherence theorem. Each time we have a morphism $f \colon A \to B$ in a monoidal category, we have a corresponding morphism $A \to B$ in its strictification. This morphism can be lifted to the original category to uniquely produce, say, a morphism $(\lambda_A ; f ; \lambda_B^{-1}) \colon I \otimes A \to I \otimes B$. Each time the source and the target are clearly determined, we simply write f again for this new morphism.

Definition A.5 (Symmetric monoidal category, [55]). A *symmetric monoidal category* $(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho, \sigma)$ is a monoidal category $(\mathsf{C}, \otimes, I, \alpha, \lambda, \rho)$ equipped with a braiding $\sigma_{A,B} \colon A \otimes B \to B \otimes A$, which satisfies the hexagon equation

$$\alpha_{A,B,C} ; \sigma_{A,B \otimes C} ; \alpha_{B,C,A} = (\sigma_{A,B} \otimes \mathrm{id}) ; \alpha_{B,A,C} ; (\mathrm{id} \otimes \sigma_{A,C})$$

and additionally satisifes $\sigma_{A,B} ; \sigma_{B,A} = \mathrm{id}$.

Remark A.6 (Notation). We omit symmetries when this does not cause confusion. We write $\underline{a}$ for the morphism a tensored with some identities when these can be deduced from the context. For instance, let $f\colon A \to B$, let $h\colon B \to D$ and let $g\colon B \otimes D \to E$. We write $\underline{f; h}; g$ for the morphism $(f \otimes \mathrm{id}); \sigma; (\mathrm{id} \otimes h); g$, which could have been also written as $(f \otimes \mathrm{id}); (h \otimes \mathrm{id}); \sigma; g$

Definition A.7 ([55]). A symmetric monoidal functor between two symmetric monoidal categories $(\mathsf{C}, \sigma^{\mathsf{C}})$ and $(\mathsf{D}, \sigma^{\mathsf{D}})$ is a monoidal functor $F\colon \mathsf{C} \to \mathsf{D}$ such that $\sigma^{\mathsf{D}}; \mu = \mu; F(\sigma^{\mathsf{C}})$.

Definition A.8. A *cartesian monoidal category* is a monoidal category whose tensor is the categorical product and whose unit is a terminal object.

Definition A.9. A *feedback functor* between two feedback monoidal categories $(\mathsf{C}, \mathrm{F}^{\mathsf{C}}, \mathrm{fbk}^{\mathsf{C}})$ and $(\mathsf{D}, \mathrm{F}^{\mathsf{D}}, \mathrm{fbk}^{\mathsf{D}})$ is a symmetric monoidal functor $G\colon \mathsf{C} \to \mathsf{D}$ such that $G; \mathrm{F}^{\mathsf{D}} = \mathrm{F}^{\mathsf{C}}; G$ and

$$G(\mathrm{fbk}^{\mathsf{C}}_S(f)) = \mathrm{fbk}^{\mathsf{D}}_{GS}(\mu_{\mathrm{F}S,A}; Gf; \mu^{-1}_{S,B}),$$

for each $f\colon \mathrm{F}S \otimes X \to S \otimes Y$, where $\mu_{A,B}\colon G(A) \otimes G(B) \to G(A \otimes B)$ is the structure morphism of the monoidal functor G.

Theorem A.10 (see [46]). $\mathsf{St}_{\mathrm{F}}(\mathsf{C})$ *is the free category with feedback over* (C, F).

Proof sketch. Let $(\mathsf{D}, \mathrm{F}^{\mathsf{D}}, \mathrm{fbk}^{\mathsf{D}})$ be any other symmetric monoidal category with an endofunctor, and let $H\colon \mathsf{C} \to \mathsf{D}$ be such that $\mathrm{F}; H = H; \mathrm{F}^{\mathsf{D}}$. We will prove that it can be extended uniquely to a feedback functor $\tilde{H}\colon \mathsf{St}_{\mathrm{F}}(\mathsf{C}) \to \mathsf{D}$.

It can be proven that any expression involving feedback can be reduced applying the feedback axioms to an expression of the form $\mathrm{fbk}(f)$ for some $f\colon \mathrm{F}S \otimes X \to S \otimes Y$. After this, the definition of $\tilde{H}$ in this morphism is forced to be $\tilde{H}(\mathrm{fbk} f) = \mathrm{fbk}^{\mathsf{D}}(\mathsf{D})$. This reduction is uniquely up to sliding, and the morphisms of the $\mathsf{St}(\bullet)$ construction are precisely morphisms $f\colon \mathrm{F}S \otimes X \to S \otimes Y$ quotiented by sliding equivalence. This is the core of the proof in [46]. □

A.1 Markov categories

Definition A.11. The finite distribution commutative monad $\mathrm{D}\colon \mathrm{Set} \to \mathrm{Set}$ associates to each set the set of finite-support probability distributions over it.

$$\mathrm{D}(X) = \left\{ p\colon X \to [0,1] \;\middle|\; \sum_{p(x)>0}^{|\{x|p(x)>0\}|<\infty} p(x) = 1 \right\}.$$

We call Stoch to the symmetric monoidal kleisli category of the finite distribution monad, $\mathsf{kl}(\mathrm{D})$.

We write $f(y|x)$ for the probability $f(x)(y) \in [0,1]$. Composition, $f; g$, is defined by

$$(f\,; g)(z|x) = \sum_{y \in Y} g(z|y) f(y|x).$$

The cartesian product $(\times)$ in Set induces a monoidal (non-cartesian) product on $\mathsf{kl}(\mathrm{D})$. That is, $\mathsf{kl}(\mathrm{D})$ has comonoids $(\blacktriangle)_X : X \to X \times X$ on every object, with $(\bullet)_X\colon X \to 1$ as counit. However, contrary to what happens in Set, these comultiplications are not natural: *sampling and copying the result* is different from *taking two independent samples.*

Definition A.12 (Markov category, [30, Definition 2.1]). A *Markov category* C is a symmetric monoidal category in which each object $X \in \mathsf{C}$ has a cocommutative comonoid structure $(X, \varepsilon = \bullet_X\colon X \to I, \delta = \blacktriangle_X\colon X \to X \otimes X)$ with

- *uniform* comultiplications, $\blacktriangle_{X\otimes Y} = (\blacktriangle_X \otimes \blacktriangle_Y)\underline{\sigma_{X,Y}}$;
- *uniform* counits, $\bullet_{X\otimes Y} = \bullet_X \otimes \bullet_Y$; and
- *natural* counits, $f; \bullet_Y = \bullet_X$ for each $f\colon X \to Y$.

Crucially, comultiplications do not need to be natural.

Remark A.13 ([30, Remark 2.4]). Any cartesian category is a Markov category. However, not any Markov category is cartesian, and the most interesting examples are those that fail to be cartesian, such as Stoch. The failure of comultiplication being natural makes it impossible to apply Fox's theorem (Theorem C.5).

The structure of a Markov category is very basic. In most cases, we do need extra structure to reason about probabilities: this is the role of conditionals and ranges.

Remark A.14 (Notation). In a Markov category, given any $f\colon X_0 \to Y_0$ and any $g\colon Y_0 \otimes X_0 \otimes X_1 \to Y_1$, we write $(f \triangleleft g)\colon X_0 \otimes X_1 \to Y_0 \otimes Y_1$ for the morphism defined by

$$(f \triangleleft g) = \underline{(\blacktriangle_A); f; (\blacktriangle_B)}; \underline{g},$$

which is the string diagram in Figure 14.

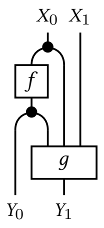


Figure 14. The morphism $(f \triangleleft g)$.

Proposition A.15. *Up to symmetries,*

$$(f \triangleleft g) \triangleleft h = f \triangleleft (g \triangleleft h).$$

We may simply write $(f \triangleleft g \triangleleft h)$ *for any of the two, omitting the symmetry.*

Proof. Using string diagrams (Figure 15). Note that $(\blacktriangle)$ is coassociative and cocommutative. □

The Markov category Stoch also has *conditionals* [30], a property which we will use to prove the main result regarding stochastic processes.

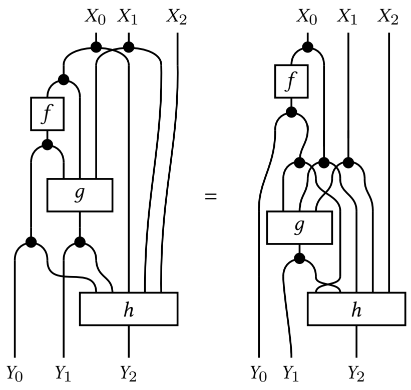


Figure 15. Associativity, up to symmetries, of the triangle operation.

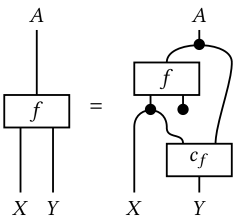


Figure 16. Condititionals in a Markov category.

Definition A.16 (Conditionals, [30, Definition 11.5]). Let C be a Markov category. We say that C has *conditionals* if for every morphism $f\colon A \to X \otimes Y$, writing $f_Y\colon A \to X$ for its first projection, there exists $c_f\colon X \otimes A \to Y$ such that $f = f_Y \triangleleft c_f$ (Figure 16).

Proposition A.17. *The Markov category* Stoch *has conditionals [30, Example 11.6].*

Proof. Let $f\colon A \to X \otimes Y$. If Y is empty, we are automatically done. If not, pick some $y_0 \in Y$, and define

$$c_f(y|x,a) = \begin{cases} f(x,y|a)/\sum_{x\in X} f(x,y|a) \\ \qquad \text{if } f(x,y|a) > 0 \text{ for some } x \in X, \\ (y = y_0) \quad \text{otherwise.} \end{cases}$$

It is straightforward to check that this does indeed define a distribution, and that it factors the original f as expected. □

Definition A.18 (Ranges). In a Markov category, a *range* for a morphism $f\colon A \to B$ is a morphism $r_f\colon A \otimes B \to A \otimes B$ that

1. does not change its output $f \triangleleft \mathrm{id}_{A\otimes B} = f \triangleleft r_f$,
2. is *deterministic*, meaning $r_f ; \curlywedge_{A\otimes B} = \curlywedge_{A\otimes B} ; (r_f \otimes r_f)$,
3. and has the *range* property, $f \triangleleft g = f \triangleleft h$ must imply

   $$(r_f \otimes \mathrm{id}) ; g = (r_f \otimes \mathrm{id}) ; h$$

   for any suitably typed g and h.

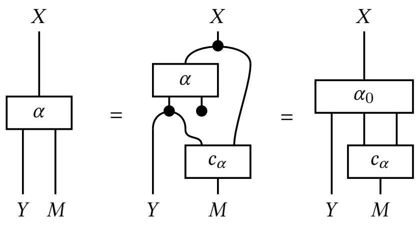


Figure 17. Productivity for Markov categories.

We say that a Markov category *has ranges* if there exists a range for each morphism of the category.

Remark A.19. There already exists a notion of categorical *range* in the literature, due to Cockett, Guo and Hofstra [20]. It arises in parallel to the notion of *support* in *restriction categories* [21]. The definition better suited for our purposes is different, even if it seems inspired by the same idea. The main difference is that we are using a *controlled range*; that is, the range of a morphism depends on the input to the original morphism. We keep the name hoping that it will not cause any confusion, as we do not deal explicitly with restriction categories in this text.

Proposition A.20. *The Markov category* Stoch *has ranges.*

Proof. Given $f\colon A \to B$, we know that for each $a \in A$ there exists some $b_a \in B$ such that $f(b_a|a) > 0$. We fix such $b_a \in B$, and we define $r_f\colon A \otimes B \to A \otimes B$ as

$$r_f(a,b) = \begin{cases} (a,b) & \text{if } f(b|a) > 0, \\ (a,b_a) & \text{if } f(b|a) = 0. \end{cases}$$

It is straightforward to check that it satisfies all the properties of ranges. □

Theorem A.21. *Any Markov category with conditionals and ranges is productive.*

Proof. Given any $\langle\alpha| \in \mathrm{Stage}_1(\mathbb{X},\mathbb{Y})$, we can define

$$\alpha_0 = \curlywedge_A ; \underline{\alpha ; (\curlywedge_Y \otimes \downarrow_M)}.$$

This is indeed well-defined because of naturality of the discarding map $(\downarrow)_M\colon M \to I$ in any Markov category. Let $c_\alpha\colon Y \otimes X \to M$ be a conditional of α. This representative can then be factored as $\alpha = \alpha_0 ; \underline{c_\alpha}$ (Figure 17).

Now assume that for two representatives $\langle\alpha_i| = \langle\alpha_j|$ we have that $\langle\alpha_i ; u| = \langle\alpha_j ; v|$. By naturality of the discarding, $\alpha_i ; \underline{\varepsilon} = \alpha_j ; \underline{\varepsilon}$, and let r be a range of this map. Again by naturality of discarding, we have $\underline{\alpha_i} ; \underline{u} ; \underline{\downarrow} = \underline{\alpha_j} ; \underline{v} ; \underline{\downarrow}$. Let then c_i and c_j be conditionals of α_i and α_j: we have that $(\alpha_0 \triangleleft c_i) ; u ; \underline{\downarrow} = (\alpha_0 \triangleleft c_j) ; v ; \underline{\downarrow}$. By the properties of ranges (Figure 18), $(\alpha_0 \triangleleft r ; c_i) ; u ; \downarrow_{M(u)} = (\alpha_0 \triangleleft r ; c_j) ; v ; \downarrow_{M(v)}$, and thus, $r ; c_i ; u ; \downarrow_{M(u)} = r ; c_j ; v ; \downarrow_{M(v)}$. We pick $s_i = r ; c_i$ and we have proven that $\langle\underline{s_i} ; u| = \langle\underline{s_j} ; v|$. □

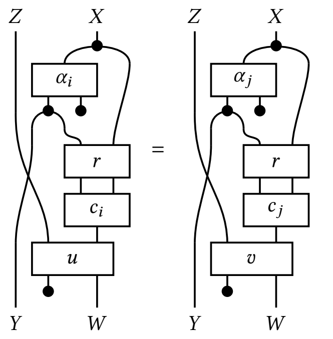


Figure 18. Applying the properties of range.

A.2 Stochastic processes

Definition A.22 (Controlled stochastic process). Let $\mathbb{X} = (X_0, X_1, \dots)$ and $\mathbb{Y} = (Y_0, Y_1, \dots)$ be infinite sequences of sets. A controlled *stochastic process* $\mathbb{f} \colon \mathbb{X} \to \mathbb{Y}$ is an infinite sequence $\mathbb{f} = (f_0, f_1, \dots)$ of functions $f_n \colon X_n \times \dots \times X_1 \to \mathrm{D}(Y_n \times \dots \times Y_1)$ such that f_n coincides with the marginal distribution of f_{n+1} on the first n variables. In other words, $f_{n+1} \mathbin{;} D\pi_{Y_0,\dots,Y_n} = \pi_{X_0,\dots,X_n} \mathbin{;} f_n$.

$$\begin{tikzcd}
X_0 \times \dots \times X_{n+1} \arrow[r, "f_{n+1}"] \arrow[d, "\pi_{0,\dots,n}"'] & D(Y_0 \times \dots \times Y_{n+1}) \arrow[d, "D\pi_{0,\dots,n}"] \\
X_0 \times \dots \times X_n \arrow[r, "f_n"] & D(Y_0 \times \dots \times Y_n)
\end{tikzcd}$$

Let StochProc be the category with objects infinite sequences of sets $\mathbb{X} = (X_0, X_1, \dots)$ and morphisms controlled stochastic processes $\mathbb{f} = (f_0, f_1, \dots)$ with composition and identities defined componentwise in Stoch.

Proposition A.23 (Factoring as conditionals). *A stochastic process $f \colon \mathbb{X} \to \mathbb{Y}$ can be always written as*

$$f_n = c_0 \triangleleft c_1 \triangleleft \dots \triangleleft c_n,$$

for some family of functions

$$c_n \colon Y_0 \times \dots \times Y_{n-1} \times X_0 \times \dots \times X_n \to Y_n,$$

called the conditionals *of the stochastic process.*

Proof. We proceed by induction, noting first that $c_0 = f_0$. In the general case, we apply conditionals to rewrite $f_{n+1} = (f_{n+1} ; (\downarrow)_{Y_{n+1}}) \triangleleft c_{n+1}$. Because of the marginalization property, we know that $f_{n+1} ; (\downarrow)_{Y_{n+1}} = f_n$. So finally, $f_{n+1} = f_n \triangleleft c_{n+1}$, which by the induction hypothesis gives the desired result. □

Proposition A.24. *If two families of conditionals give rise to the same stochastic process,*

$$c_0 \triangleleft c_1 \triangleleft \dots \triangleleft c_n = c_0' \triangleleft c_1' \triangleleft \dots \triangleleft c_n',$$

then, they also give rise to the same n-stage processes in Stoch,

$$\langle c_0 \triangleleft \mathrm{id} | c_1 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \langle c_0' \triangleleft \mathrm{id} | c_1' \triangleleft \mathrm{id} | \dots | c_n' \triangleleft \mathrm{id} | .$$

Proof. We start by defining a family of morphisms r_n by induction. We take $r_0 = \mathrm{id}$ and r_{n+1} to be a *range* of $r_n ; \blacktriangle ; c_n$.

Let us prove now that for any $n \in \mathbb{N}$ and $i \leqslant n$,

$$r_i ; \blacktriangle ; c_i \triangleleft \dots \triangleleft c_n = r_i ; \blacktriangle ; c_i' \triangleleft \dots \triangleleft c_n'.$$

We proceed by induction. Observing that $c_0 = c_0'$, we prove it for $n = 0$ and also for the case $i = 0$ for any $n \in \mathbb{N}$. Assume we have it proven for n, so in particular we know that $r_i ; \blacktriangle ; c_i = r_i ; \blacktriangle ; c_i'$ for any $i \leqslant n$. Now, by induction on i, we can use the properties of ranges to show that

$$\begin{gathered}
r_i ; \blacktriangle ; c_i \triangleleft \dots \triangleleft c_n = r_i ; \blacktriangle ; c_i' \triangleleft \dots \triangleleft c_n' \\
(r_i ; \blacktriangle ; c_i \triangleleft \mathrm{id}) ; c_{i+1} \triangleleft \dots \triangleleft c_n = (r_i ; \blacktriangle ; c_i' \triangleleft \mathrm{id}) ; c_{i+1}' \triangleleft \dots \triangleleft c_n' \\
(r_i ; \blacktriangle ; c_i \triangleleft r_{i+1}) ; c_{i+1} \triangleleft \dots \triangleleft c_n = (r_i ; \blacktriangle ; c_i' \triangleleft r_{i+1}) ; c_{i+1}' \triangleleft \dots \triangleleft c_n' \\
r_{i+1} ; \blacktriangle ; c_{i+1} \triangleleft \dots \triangleleft c_n = r_{i+1} ; \blacktriangle ; c_{i+1}' \triangleleft \dots \triangleleft c_n'.
\end{gathered}$$

In particular, $r_n ; \blacktriangle ; c_n = r_n ; \blacktriangle ; c_n'$.

Now, we claim the following for each $n \in \mathbb{N}$ and each $i \leqslant n$,

$$\begin{gathered}
\langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i (c_i \triangleleft \mathrm{id}) | c_{i+1} | \dots | c_n \triangleleft \mathrm{id} | .
\end{gathered}$$

It is clear for $n = 0$ and for $i = 0$. In the inductive case for i,

$$\begin{gathered}
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i (c_i \triangleleft \mathrm{id}) | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i ; \blacktriangle ; c_i \triangleleft \mathrm{id} | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i ; \blacktriangle ; c_i \triangleleft r_{i+1} | c_{i+1} \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i ; \blacktriangle ; c_i \triangleleft \mathrm{id} | r_{i+1} (c_{i+1} \triangleleft \mathrm{id}) | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_i (c_i \triangleleft \mathrm{id}) | r_{i+1} (c_{i+1} \triangleleft \mathrm{id}) | \dots | c_n \triangleleft \mathrm{id} | .
\end{gathered}$$

A particular case of this claim is then that

$$\begin{gathered}
\langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | = \\
\langle r_0 (c_0 \triangleleft \mathrm{id}) | \dots | r_n (c_n \triangleleft \mathrm{id}) | = \\
\langle r_0 (c_0' \triangleleft \mathrm{id}) | \dots | r_n (c_n' \triangleleft \mathrm{id}) | = \\
\langle c_0' \triangleleft \mathrm{id} | \dots | c_n' \triangleleft \mathrm{id} | .
\end{gathered}$$

This can be then proven for any $n \in \mathbb{N}$. □

Corollary A.25. *Any stochastic process $f \in$ StochProc$(\mathbb{X}, \mathbb{Y})$ with a family of conditionals c_n gives rise to the observational sequence*

$$\begin{aligned}
\mathrm{obs}(f) = [\langle (c_n \triangleleft \mathrm{id}) \colon (X_0 \times Y_0 \times \dots \times X_{n-1} \times Y_{n-1}) \times X_n \to \\
(X_0 \times Y_0 \times \dots \times X_n \times Y_n) \times Y_n \rangle]_{\approx},
\end{aligned}$$

which is independent of the chosen family of conditionals.

Proof. Any two families of conditionals for f give rise to the same n-stage processes in Stoch (by Proposition A.24). Being a productive category, observational sequences are determined by their n-stage processess. □

Proposition A.26. *An observational sequence in* Stoch,

$$[\langle g_n \colon M_{n-1} \otimes X_n \to M_n \otimes Y_n \rangle]_{\approx} \in \mathrm{Obs}(\mathbb{X}, \mathbb{Y})$$

gives rise to a stochastic process $\mathrm{proc}(g) \in$ StochProc$(\mathbb{X}, \mathbb{Y})$ *defined by* $\mathrm{proc}(g)_n = \underline{g_0} ; \underline{g_1} ; \dots ; \underline{g_n} ; \underline{\varepsilon_{M_n}}$.

Proof. The symmetric monoidal category Stoch is productive: by Lemma D.3, observational sequences are determined by their n-stage truncations

$$\langle g_0 | \dots | g_n | \in \mathsf{Stage}_n(\mathbb{X}, \mathbb{Y}).$$

Each n-stage truncation gives rise to the n-th component of the stochastic process, $\mathrm{proc}(g)_n = \underline{g_0}; \underline{g_1}; \dots; \underline{g_n}; \underline{\varepsilon_{M_n}}$, and this is well-defined: composing the morphisms is invariant to *sliding equivalence*, and the last discarding map is natural.

It only remains to show that they satisfy the marginalisation property. Indeed,

$$\begin{aligned}\mathrm{proc}(g)_{n+1}; \underline{\varepsilon_{n+1}} &= \underline{g_0}; \underline{g_1}; \dots; \underline{g_{n+1}}; \underline{\varepsilon_{M_{n+1}}}; \underline{\varepsilon_{Y_{n+1}}} \\ &= \underline{g_0}; \underline{g_1}; \dots; \underline{g_n}; \underline{\varepsilon_{M_n}} \\ &= \mathrm{proc}(g)_n.\end{aligned}$$

Thus, $\mathrm{proc}(g)$ is a stochastic process in $\mathsf{StochProc}(\mathbb{X}, \mathbb{Y})$. □

Proposition A.27. *Let $f \in \mathsf{StochProc}(\mathbb{X}, \mathbb{Y})$, we have that* $\mathrm{proc}(\mathrm{obs}(f)) = f$.

Proof. Indeed, for c_n some family of conditionals,

$$f_n = c_0 \triangleleft \dots \triangleleft c_n = (c_0 \triangleleft \mathrm{id}); \dots; (c_n \triangleleft \mathrm{id}); \varepsilon_{M_n}. \qquad \square$$

Theorem A.28. *Observational sequences in* Stoch *are in bijection with stochastic processes.*

Proof. The function obs is injective by Proposition A.27. We only need to show it is also surjective.

We will prove that any n-stage process $\langle g_0 | \dots | g_n|$ can be equivalently written in the form $\langle (c_0 \triangleleft \mathrm{id}) | \dots | (c_n \triangleleft \mathrm{id})|$. We proceed by induction. Given any $\langle g_0|$ we use conditionals and dinaturality to rewrite it as

$$\langle g_0 | = \langle c_0 \triangleleft c_M | = \langle c_0 \triangleleft \mathrm{id} | \,.$$

Given any $\langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | g_{n+1}|$, we use again conditionals and dinaturality to rewrite it as

$$\begin{gathered}\langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | g_{n+1}| = \\ \langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | c_{n+1} \triangleleft c_M| = \\ \langle c_0 \triangleleft \mathrm{id} | \dots | c_n \triangleleft \mathrm{id} | c_{n+1} \triangleleft \mathrm{id}| \,.\end{gathered}$$

We have shown that obs is both injective and surjective. □

Theorem A.29 (From Theorem 7.2). *The category* Stoch *of stochastic processes is monoidally isomorphic to the category* Stream *over* Stoch.

Proof. We have shown in Theorem A.28 that proc is a bijection. Let us show that it preserves compositions. Indeed,

$$\begin{aligned}\mathrm{proc}(g; h)_n &= \underline{g_0}; \underline{h_0}; \dots; \underline{g_n}; \underline{h_n}; \underline{\varepsilon_{M_n \otimes N_n}} \\ &= \underline{g_0}; \dots; \underline{g_n}; \underline{h_0}; \dots; \underline{h_n}; \underline{(\varepsilon_{M_n} \otimes \varepsilon_{N_n})} \\ &= \underline{g_0} \dots \underline{g_n}; \underline{\varepsilon_{M_n}}; \underline{h_0} \dots \underline{h_n}; \underline{\varepsilon_{N_n}} \\ &= \mathrm{proc}(g)_n; \mathrm{proc}(h)_n.\end{aligned}$$

It also trivially preserves the identity. It induces thus an identity-on-objects functor which is moreover an equivalence of categories. Let us finally show that it preserves tensoring of morphisms.

$$\begin{aligned}\mathrm{proc}(g \otimes h)_n &= \underline{(g_0 \otimes h_0)}; \dots; \underline{(g_n \otimes h_n)}; \underline{\varepsilon_{M_n \otimes N_n}} \\ &= \underline{(g_0 \otimes h_0)}; \dots \underline{((g_n \varepsilon_{M_n}) \otimes (h_n \varepsilon_{N_n}))} \\ &= \underline{(g_0 \dots g_n \varepsilon_{M_n})} \otimes \underline{(h_0 \dots h_n \varepsilon_{N_n})} \\ &= \mathrm{proc}(g)_n \otimes \mathrm{proc}(h)_n.\end{aligned}$$

It is thus also a monoidal equivalence. □

B Coend Calculus and Profunctors

Coend calculus is the name given to the a branch of category theory that describes the behaviour of certain colimits called *coends*. MacLane [55] and Loregian [53] give complete presentations of coend calculus.

Definition B.1. *Coends* are the coequalizers of the action of morphisms on both arguments of a profunctor.

$$\mathrm{coend}(P) := \mathrm{coeq}\left(\textstyle\coprod_{f \colon B \to A} P(A, B) \rightrightarrows \coprod_{X \in \mathsf{C}} P(X, X) \right).$$

Coends are usually denoted with a superscripted integral, drawing on an analogy with classical calculus.

$$\int^{X \in \mathsf{C}} P(X, X) = \mathrm{coend}(P).$$

Proposition B.2 (Yoneda reduction). *Let* C *be any category and let $F \colon \mathsf{C} \to \mathsf{Set}$ be a functor; the following isomorphism holds for any given object $A \in \mathsf{C}$.*

$$\int^{X \in \mathsf{C}} \mathrm{hom}(X, A) \times FX \cong FA.$$

Following the analogy with classical analysis, the hom *profunctor works as a Dirac's delta.*

Proposition B.3 (Fubini rule). *Coends commute between them; that is, there exists a natural isomorphism*

$$\int^{X_1 \in \mathsf{C}} \int^{X_2 \in \mathsf{C}} P(X_1, X_2, X_1, X_2) \cong \int^{X_2 \in \mathsf{C}} \int^{X_1 \in \mathsf{C}} P(X_1, X_2, X_1, X_2).$$

In fact, they are both isomorphic to the coend over the product category,

$$\int^{(X_1, X_2) \in \mathsf{C} \times \mathsf{C}} P(X_1, X_2, X_1, X_2).$$

Following the analogy with classical analysis, coends follow the Fubini rule for integrals.

A profunctor from a category A to a category B is a functor $P \colon \mathsf{A}^{op} \times \mathsf{B} \to \mathsf{Set}$. They can be seen as a categorification of the concept of *relations*, functions $A \times B \to 2$. Under this analogy, existential quantifiers correspond to *coends*. The canonical example of a profunctor is, hom: $\mathsf{A}^{op} \times \mathsf{A} \to \mathsf{Set}$, the profunctor that returns the set of morphisms between two objects. Many operations relating families of processes are more easily defined in terms of profunctors: for instance, sequential composition connects the outputs of a family of processes to the outputs of another family.

Definition B.4 (Sequential composition)**.** Two profunctors $P \colon \mathsf{A}^{op} \times \mathsf{B} \to \mathsf{Set}$ and $Q \colon \mathsf{B}^{op} \times \mathsf{C} \to \mathsf{Set}$ compose sequentially into a profunctor $P \diamond Q \colon \mathsf{A}^{op} \times \mathsf{C} \to \mathsf{Set}$ defined by

$$(P \diamond Q)(A, C) = \int^{B \in \mathsf{B}} P(A, B) \times Q(B, C).$$

The hom-profunctor hom: $\mathsf{A}^{op} \times \mathsf{A} \to \mathsf{Set}$ that returns the set of morphisms between two objects is the unit for sequential composition. Sequential composition is associative up to isomorphism.

Definition B.5 (Parallel composition)**.** Two profunctors $P \colon \mathsf{A}_1^{op} \times \mathsf{B}_1 \to \mathsf{Set}$ and $Q \colon \mathsf{A}_2^{op} \times \mathsf{B}_2 \to \mathsf{Set}$ compose *in parallel* into a profunctor $P \times Q \colon \mathsf{A}_1^{op} \times \mathsf{A}_2^{op} \times \mathsf{B}_1 \times \mathsf{B}_2 \to \mathsf{Set}$ defined by

$$(P \times Q)(A, A', B, B') = P(A, B) \times Q(A', B').$$

Definition B.6 (Intensional communicating composition)**.** Let A, B, C be categories and let B have a monoidal structure. Let $P \colon \mathsf{A}^{op} \times \mathsf{B} \to \mathsf{Set}$ and $Q \colon \mathsf{B}^{op} \times \mathsf{C}^{\mathbb{N}} \to \mathsf{Set}$ be a pair of profunctors. Their *intensional communicating composition* is the profunctor $P \boxdot Q \colon \mathsf{A}^{op} \times \mathsf{B}^{op} \times \mathsf{B} \times \mathsf{C} \to \mathsf{Set}$ defined as

$$(P \boxdot Q)(A, B; B', C) = \sum_{M \in \mathsf{B}} P(A, B \otimes M) \times Q(M \otimes B', C).$$

Remark B.7. Let C be a monoidal category and let $P \colon \mathsf{C}^{op} \times \mathsf{C} \to \mathsf{Set}$ and $Q \colon [\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}] \to \mathsf{Set}$ be a pair of profunctors. Note that $[\mathbb{N}, \mathsf{C}] \cong \mathsf{C} \times [\mathbb{N}, \mathsf{C}]$, and so the second profunctor can be interpreted as having type $Q \colon \mathsf{C}^{op} \times ([\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}]) \to \mathsf{Set}$. In this case, their intensional communicating composition is defined by

$$(P \boxdot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \sum_{M \in \mathsf{C}} P(X_0, M \otimes Y_0) \times Q(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

This is the composition we use when we describe the endofunctor $(\mathrm{hom} \boxdot \bullet) \colon [[\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}] \to [[\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}]$.

$$(\mathrm{hom} \boxdot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \sum_{M \in \mathsf{C}} \mathrm{hom}(X_0, M \otimes Y_0) \times Q(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

Definition B.8 (Communicating profunctor composition)**.** Let A, B, C be categories and let B have a monoidal structure. Two profunctors $P \colon \mathsf{A}^{op} \times \mathsf{B} \to \mathsf{Set}$ and $Q \colon \mathsf{B}^{op} \times \mathsf{C} \to \mathsf{Set}$ compose communicating along B into the profunctor $(P \odot Q) \colon \mathsf{A}^{op} \times \mathsf{B} \times \mathsf{B}^{op} \times \mathsf{C} \to \mathsf{Set}$ defined by

$$(P \odot Q)(A, B; B', C) = \int^{M} P(A, B \otimes M) \times Q(M \otimes B', C).$$

The profunctors $\mathrm{hom}(I, \bullet) \colon \mathsf{B} \to \mathsf{Set}$ and $\mathrm{hom}(\bullet, I) \colon \mathsf{B}^{op} \to \mathsf{Set}$ are left and right units with respect to communicating composition. The communicating composition of three profunctors $P \colon \mathsf{A}^{op} \times \mathsf{B} \to \mathsf{Set}$, $Q \colon \mathsf{B}^{op} \times \mathsf{C} \to \mathsf{Set}$ and $R \colon \mathsf{C}^{op} \times \mathsf{D} \to \mathsf{Set}$ is associative up to isomorphism and a representative can be written simply by $(P \odot Q \odot R) \colon \mathsf{A}^{op} \times \mathsf{B} \times \mathsf{B}^{op} \times \mathsf{C} \times \mathsf{C}^{op} \times \mathsf{D} \to \mathsf{Set}$, where both B and C are assumed to have a monoidal structure.

Remark B.9. This is the composition we use when we describe the endofunctor $(\mathrm{hom} \odot \bullet) \colon [[\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}] \to [[\mathbb{N}, \mathsf{C}]^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}]$.

$$(\mathrm{hom} \odot Q)(\mathbb{X}; \mathbb{Y}) \coloneqq \int^{M \in \mathsf{C}} \mathrm{hom}(X_0, M \otimes Y_0) \times Q(M \cdot \mathbb{X}^+, \mathbb{Y}^+).$$

B.1 Initial algebras, final coalgebras

Definition B.10 (Algebras and coalgebras)**.** Let C be a category and let $F \colon \mathsf{C} \to \mathsf{C}$ be an endofunctor. An *algebra* (X, α) is an object $X \in \mathsf{C}$, together with a morphism $\alpha \colon FX \to X$. A *coalgebra* (Y, β) is an object $Y \in \mathsf{C}$, together with a morphism $\beta \colon Y \to FY$.

An *algebra morphism* $f \colon (X, \alpha) \to (X', \alpha')$ is a morphism $f \colon X \to X'$ such that the diagram on the left commutes. A *coalgebra morphism* $g \colon (Y, \beta) \to (Y', \beta')$ is a morphism $f \colon Y \to Y'$ such that the diagram on the right commutes.

$$\begin{tikzcd} FX \arrow[r, "Ff"] \arrow[d, "\alpha"'] & FX' \arrow[d, "\alpha'"] \\ X \arrow[r, "f"] & X' \end{tikzcd} \qquad \begin{tikzcd} Y \arrow[r, "g"] \arrow[d, "\beta"'] & Y' \arrow[d, "\beta'"] \\ FY \arrow[r, "Fg"] & FY' \end{tikzcd}$$

Algebras for an endofunctor form a category with algebra morphisms between them. The initial algebra is the initial object in this category. Coalgebras for an endofunctor form a category with coalgebra morphisms between them. The final coalgebra is the terminal object in this category.

Definition B.11 (Fixpoints of an endofunctor)**.** Let C be a category and let $F \colon \mathsf{C} \to \mathsf{C}$ be an endofunctor. A *fixpoint* is an algebra (X, α) such that $\alpha \colon FX \to X$ is an isomorphism. Equivalently, a fixpoint is a coalgebra (Y, β) such that $\beta \colon Y \to FY$ is an isomorphism.

Fixpoints form a category with algebra morphisms (or, equivalently, coalgebra morphisms) between them.

Theorem B.12 (Lambek, [50])**.** *The final coalgebra of a functor is a fixpoint. As a consequence, when it exists, it is the final fixpoint.*

Theorem B.13 (Adamek, [4])**.** *Let* D *be a category with a final object* 1 *and* ω*-shaped limits. Let* $F \colon \mathsf{D} \to \mathsf{D}$ *be an endofunctor. We write* $L = \lim_n F^n 1$ *for the limit of the following* ω*-chain, which is called the* terminal sequence.

$$1 \xleftarrow{!} F1 \xleftarrow{F!} FF1 \xleftarrow{FF!} FFF1 \xleftarrow{FFF!} \dots$$

Assume that F preserves this limit, meaning that the canonical morphism $FL \to L$ is an isomorphism. Then, L is the final F-coalgebra.

B.2 Size concerns, limits and colimits

Remark B.14. We call Set to the category of sets and functions below a certain Grothendieck universe. We do take colimits (and coends) over this category without creating size issues: we can be sure of their existence in our metatheoretic category of sets.

Proposition B.15. *Terminal coalgebras exist in* Set. *More generally, the category of sets below a certain regular uncountable cardinal is algebraically complete and cocomplete; meaning that every* Set*-endofunctor has a terminal coalgebra and an initial algebra. See [5, Theorem 13].*

Theorem B.16 (Coproducts commute with connected limits)**.** *Let I be a set, understood as a discrete category, and let A be a connected category with $F\colon I \times \mathsf{A} \to \mathsf{Set}$ a functor. The canonical morphism*

$$\sum_{i\in I} \lim_{a\in A} F(i,a) \to \lim_{a\in A} \sum_{i\in I} F(i,a)$$

is an isomorphism.

In particular, let $F_n\colon I \to \mathsf{Set}$ be a family of functors indexed by the natural numbers with a family of natural transformations $\alpha_n\colon F_{n+1} \to F_n$. The canonical morphism

$$\sum_{i\in I} \lim_{n\in\mathbb{N}} F_n(i) \to \lim_{n\in\mathbb{N}} \sum_{i\in I} F_n(i)$$

is an isomorphism.

Proof. Note that there are no morphisms between any two indices $i, j \in I$. Once some $i \in I$ is chosen in any factor of the connected limit, it forces any other factor to also choose $i \in I$. This makes the local choice of $i \in I$ be equivalent to the global choice of $i \in I$. □

C The List^+ opmonoidal comonad and Fox's theorem

Proposition C.1. *Cartesian monoidal categories are productive.*

Proof. Let $\langle\alpha| \in \mathrm{Stage}_1(\mathbb{X},\mathbb{Y})$. For some given representative $\alpha\colon X_0 \to M \otimes Y_0$, we define the two projections $\alpha_Y = \alpha; \underline{\bullet}_M\colon X_0 \to Y_0$ and $\alpha_M = \alpha; \underline{\bullet}_Y$. The second projection α_M depends on the specific representative α we have chosen; however, the first projection α_Y is defined independently of the specific representative α, as a consequence of naturality of the discarding map (see Fox's theorem for cartesian monoidal categories Theorem C.5). We define $\alpha_0 = \delta_{X_0}; \underline{\alpha_Y}$. Then, we can factor any representative as $\alpha = \alpha_0; \underline{\alpha_M}$ (see Figure 19). Now, assume that we have two representatives $\langle\alpha_i| = \langle\alpha_j|$ for which $\langle\alpha_i; u| = \langle\alpha_j; v|$. By naturality of the discarding map, $\alpha_i; \bullet = \alpha_j; \bullet$, and we call this map α_Y. Again

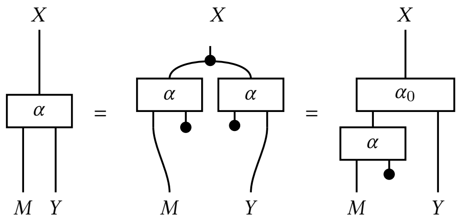


Figure 19. Productivity for cartesian categories.

by naturality of the discarding map, $\alpha_i; u; \underline{\bullet} = \alpha_j; v; \underline{\bullet}$, and discarding the output in Y, we get that $\alpha_{M,i}; u; \underline{\bullet} = \alpha_{M,j}; v; \underline{\bullet}$, which implies $\langle\alpha_{M,i}; u| = \langle\alpha_{M,j}; v|$. □

Definition C.2 (Opmonoidal comonad)**.** In a monoidal category $(\mathsf{C}, \otimes, I)$, a comonad (R, ε, δ) is an *opmonoidal comonad* when the endofunctor $R\colon \mathsf{C} \to \mathsf{C}$ is oplax monoidal with laxators $\psi_{X,Y}\colon R(X \otimes Y) \to RX \otimes RY$ and $\psi^I\colon RI \to I$, and both the counit $\varepsilon_X\colon RX \to X$ and the comultiplication $\delta_X\colon RX \to RRX$ are monoidal natural transformations.

Explicitly, $\varepsilon_I = \psi_0$, $\varepsilon_{X\otimes Y} = \psi_{X,Y}; (\varepsilon_X \otimes \varepsilon_Y)$, $\delta_I; \psi_0; \psi_0 = \psi_0$ and $\delta_{X\otimes Y}; \psi_{X,Y}; \psi_{RX,RY} = \psi_{X,Y}; (\delta_X \otimes \delta_Y)$.

Alternatively, an *opmonoidal comonad* is a comonoid in the bicategory MonOplax of oplax monoidal functors with composition and monoidal natural transformations between them.

Definition C.3. Let $(\mathsf{C}, \otimes, I)$ be a symetric monoidal category. There is a functor $\mathsf{List}^+\colon \mathsf{C} \to \mathsf{C}$ defined on objects by

$$\mathsf{List}^+(X)_n \coloneqq \bigotimes_{i=0}^{n} X_i.$$

This functor is monoidal, with oplaxators $\psi_0^+\colon \mathsf{List}^+(I) \to I$ and $\psi_{X,Y}\colon \mathsf{List}^+(X \otimes Y) \to \mathsf{List}^+(X) \otimes \mathsf{List}^+(Y)$ given by symmetries, associators and unitors.

Theorem C.4 (From Theorem 6.1)**.** *The opmonoidal functor List^+ has an opmonoidal comonad structure if and only if its base monoidal category $(\mathsf{C}, \otimes, I)$ is cartesian monoidal.*

Proof. When C is cartesian, we can construct the comonad structure using projections $\prod_{i=0}^n X_i \to X_n$ and copying together with braidings $\prod_{i=0}^n X_i \to \prod_{i=0}^n \prod_{k=0}^i X_k$. These are monoidal natural transformations making List^+ a monoidal comonad.

Suppose (L, ε, δ) is an opmonoidal comonad structure. This means it has families of natural transformations

$$\delta_n\colon \bigotimes_{i=0}^{n} X_i \to \bigotimes_{i=0}^{n} \bigotimes_{k=0}^{i} X_k \text{ and } \varepsilon_n\colon \bigotimes_{i=0}^{n} X_i \to X_n.$$

We will use these to construct a uniform counital comagma structure on every object of the category. By a refined version of Fox's theorem (Theorem C.7), this will imply that C is cartesian monoidal.

Let $X \in \mathsf{C}$ be any object. Choosing $n = 2$, $X_0 = X$ and $X_1 = I$; and using coherence maps, we get $\delta_2 \colon X \to X \otimes X$ and $\varepsilon_2 \colon X \to I$. These are coassociative, counital, natural and uniform because the corresponding transformations δ and ε are themselves coassociative, counital, natural and monoidal. This induces a uniform comagma structure in every object $(X, \delta_2, \varepsilon_2)$; with this structure, every morphism of the category is a comagma homomorphism because δ_2 and ε_2 are natural. □

Theorem C.5 (Fox's theorem [29]). *A symmetric monoidal category $(\mathsf{C}, \otimes, I)$ is cartesian monoidal if and only if every object $X \in \mathsf{C}$ has a cocommutative comonoid structure $(X, \varepsilon_X, \delta_X)$, every morphism of the category $f \colon X \to Y$ is a comonoid homomorphism, and this structure is uniform across the monoidal category: meaning that $\varepsilon_{X \otimes Y} = \varepsilon_X \otimes \varepsilon_Y$, that $\varepsilon_I = \mathrm{id}$, that $\delta_I = \mathrm{id}$ and that $\delta_{X \otimes Y} = (\delta_X \otimes \delta_Y) ; (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.*

Remark C.6. Most sources ask the comonoid structure in Fox's theorem (Theorem C.5) to be cocommutative [28, 29]. However, cocommutativity and coassociativity of the comonoid structure are implied by the fact that the structure is uniform and natural. We present an original refined version of Fox's theorem.

Theorem C.7 (Refined Fox's theorem). *A symmetric monoidal category $(\mathsf{C}, \otimes, I)$ is cartesian monoidal if and only if every object $X \in \mathsf{C}$ has a counital comagma structure $(X, \varepsilon_X, \delta_X)$, or $(X, \downarrow_X, \curlywedge_X)$, every morphism of the category $f \colon X \to Y$ is a comagma homomorphism, and this structure is uniform across the monoidal category: meaning that $\varepsilon_{X \otimes Y} = \varepsilon_X \otimes \varepsilon_Y$, $\varepsilon_I = \mathrm{id}$, $\delta_I = \mathrm{id}$ and $\delta_{X \otimes Y} = (\delta_X \otimes \delta_Y) ; (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id})$.*

Proof. We prove that such a comagma structure is necessarily coassociative and cocommutative. Note that any comagma homomorphism $f \colon A \to B$ must satisfy $\delta_A ; (f \otimes f) = f ; \delta_B$. In particular, $\delta_X \colon X \to X \otimes X$ must itself be a comagma homomorphism (see Figure 20), meaning that

$$\delta_X ; (\delta_X \otimes \delta_X) = \delta_X ; \delta_{X \otimes X} = \delta_X ; (\delta_X \otimes \delta_X) ; (\mathrm{id} \otimes \sigma_{X,Y} \otimes \mathrm{id}), \tag{3}$$

where the second equality follows by uniformity.

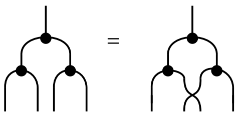


Figure 20. Comultiplication is a comagma homomorphism.

Now, we prove cocommutativity (Figure 21): composing both sides of Equation (3) with $(\epsilon_X \otimes \mathrm{id} \otimes \mathrm{id} \otimes \epsilon_X)$ discards the two external outputs and gives $\delta_X = \delta_X ; \sigma_X$.

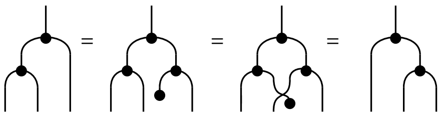


Figure 22. Coassociativity

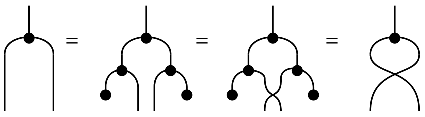


Figure 21. Cocommutativity

Now, we prove coassociativity (Figure 22): composing both sides of Equation (3) with $(\mathrm{id} \otimes \epsilon_X \otimes \mathrm{id} \otimes \mathrm{id})$ discards one of the middle outputs and gives $\delta_X ; (\mathrm{id} \otimes \delta_X) = \delta_X ; (\delta_X \otimes \mathrm{id})$.

A coassociative and cocommutative comagma is a cocommutative comonoid. We can then apply the classical form of Fox's theorem (Theorem C.5). □

Distributive laws. One could hope to add effects such as probability or non-determinism to set-based streams via the bikleisli category arising from a monad-comonad distributive law $\mathsf{List}^+ \circ \mathbb{T} \Rightarrow \mathbb{T} \circ \mathsf{List}^+$ [7, 64], as proposed by Uustalu and Vene [75]. This would correspond to a lifting of the List^+ comonad to the kleisli category of some commutative monad T; the arrows $\mathbb{X} \to \mathbb{Y}$ of such a category would look as follows,

$$f_n \colon X_1 \times \cdots \times X_n \to TY_n.$$

However, we have already shown that this will not result in a monoidal comonad whenever $\mathsf{kl}(T)$ is not cartesian. To see explicitly what fails, we use the string diagrams to show how composition should work for the case $n = 2$ (Figure 23). This composition is not associative or unital whenever the kleisli category does not have natural comultiplications or counits, respectively (Figures 24 and 25).

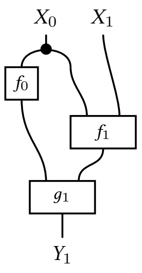


Figure 23. Composition in the case $n = 2$.

D Productive categories

Definition D.1 (Truncating coherently). Let $f_n^k \colon X_n \otimes M_{n-1} \to Y_n \otimes M_n$ be a family of families of morphisms of increasing length, indexed by $k \in \mathbb{N}$ and $n \leqslant k$. We say that this family

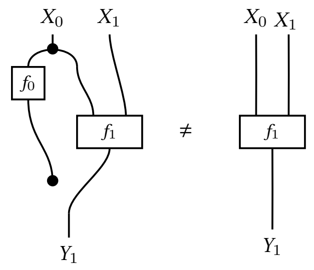


Figure 24. Failure of unitality if discarding is not natural, as it happens, for instance, with partial functions.

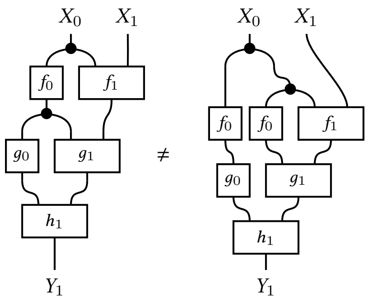


Figure 25. Failure of associativity if copying is not natural, as it happens, for instance, with stochastic functions.

truncates coherently if $\langle f_0^p | \dots | f_n^p | = \langle f_0^q | \dots | f_n^q |$ for each $p, q \in \mathbb{N}$ and each $n \leqslant \min\{p, q\}$.

Lemma D.2 (Factoring a family of processes). *In a productive category, let $(\langle f_0^k | \dots | f_k^k |)_{k \in \mathbb{N}}$ be a sequence of sequences that truncates coherently. Then, there exists a sequence h_i with $s_{i-1}^k f_i^k = h_i s_i^k$ such that, for each $k \in \mathbb{N}$ and each $n \leqslant k$, $\langle f_0^k | \dots | f_n^k | = \langle h_0 | \dots | h_n |$. Moreover, this family h_i is such that $\langle h_0 \dots h_n s_n^p u| = \langle h_0 \dots h_n s_n^q v|$ implies $\langle s_n^p u| = \langle s_n^q v|$.*

Proof. We construct the family by induction. In the case $n = 0$, we use that the family truncates coherently to have that $\langle f_0^p | = \langle f_0^q |$ and thus, by productivity, create an h_0 with $f_0^k = h_0 s_0^k$ such that $\langle h_0 s_0^p u| = \langle h_0 s_0^q v|$ implies $\langle s_0^p u| = \langle s_0^q v|$.

In the general case, assume we already have constructed $h_0, \dots, h_{n-1}$ with $s_{i-1}^k f_i^k = h_i s_i^k$ such that, for each $k \in \mathbb{N}$ and $\langle f_0^k | \dots | f_{n-1}^k | = \langle h_0 | \dots | h_{n-1} |$. Moreover, $\langle h_0 \dots h_{n-1} s_{n-1}^p u| = \langle h_0 \dots h_{n-1} s_{n-1}^q v|$ implies $\langle s_{n-1}^p u| = \langle s_{n-1}^q v|$.

In this case, we use the fact that composition "along a bar" is dinatural: $\langle f_0^p | \dots | f_n^p | = \langle f_0^q | \dots | f_n^q |$ implies that $\langle \underline{f_0^p \dots f_n^p} | = \langle \underline{f_0^q \dots f_n^q} |$. This can be then rewritten as

$$\langle h_0 \dots h_{n-1} s_{n-1}^p f_n^p | = \langle h_0 \dots h_{n-1} s_{n-1}^q f_n^q |,$$

which in turn implies $\langle s_{n-1}^p f_n^p | = \langle s_{n-1}^q f_n^q |$. By productivity, there exists h_n with $s_{n-1}^k f_n^k = h_n s_n^k$ such that $\langle f_0^k | \dots | f_n^k | = \langle h_0 | \dots | h_n |$.

Finally, assume that $\langle h_0 \dots h_{n-1} h_n s_n^p u| = \langle h_0 \dots h_{n-1} h_n s_n^q v|$. Thus, we have $\langle h_0 \dots h_{n-1} s_{n-1}^p f_n^p u| = \langle h_0 \dots h_{n-1} s_{n-1}^q f_n^q v|$ and $\langle s_{n-1}^p f_n^p u| = \langle s_{n-1}^q f_n^q v|$. This can be rewritten as $\langle h_n s_n^p u| = \langle h_n s_n^q v|$, which in turn implies $\langle s_n^p u| = \langle s_n^q v|$. We have shown that the h_n that we constructed satisfies the desired property. □

Lemma D.3. *In a productive category, the set of observational sequencesis isomorphic to the limit of the terminal sequence of the endofunctor* (hom $\odot \bullet$) *via the canonical map between them.*

Proof. We start by noting that observational equivalence of sequences is, by definition, the same thing as being equal under the canonical map to the limit of the terminal sequence

$$\lim_n \int^{M_0, \dots, M_n} \prod_{i=0}^{n} \mathrm{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

We will show that this canonical map is surjective. That means that the domain quotiented by equality under the map is isomorphic to the codomain, q.e.d.

Indeed, given any family f_n^k that truncates coherently, we can apply Lemma D.2 to find a sequence h_i such that $\langle f_0^k | \dots | f_n^k | = \langle h_0 | \dots | h_n |$. This means that it is the image of the stateful sequence h_i. □

Lemma D.4 (Factoring two processes). *In a productive category, let $\langle f_0 | = \langle g_0 |$. Then there exists h_0 with $f_0 = h_0 s_0$ and $g_0 = h_0 t_0$ such that*

$$\langle f_0 | \dots | f_n | = \langle g_0 | \dots | g_n |$$

implies the existence of a family h_i together with s_i and t_i such that $s_{i-1} f_i = h_i s_i$ and $t_{i-1} g_i = h_i t_i$; and moreover, such that

$$\langle h_0 \dots h_n s_n u| = \langle h_0 \dots h_n t_n v| \text{ implies } \langle s_n u| = \langle t_n v|.$$

Proof. By productivity, we can find such a factorization $f_0 = h_0 s_0$ and $g_0 = h_0 t_0$.

Assume now that we have a family of morphisms such that $\langle f_0 | \dots | f_n | = \langle g_0 | \dots | g_n |$. We proceed by induction on n, the size of the family. The case $n = 0$ follows from the definition of productive category.

In the general case, we will construct the relevant h_n. The assumption $\langle f_0 | \dots | f_n | = \langle g_0 | \dots | g_n |$ implies, in particular, that $\langle f_0 | \dots | f_{n-1} | = \langle g_0 | \dots | g_{n-1} |$. Thus, by induction hypothesis, there exist $h_1, \dots, h_{n-1}$ together with $s_{i-1} f_i = h_i s_i$ and $t_{i-1} g_i = h_i t_i$, such that

$$\langle h_0 \dots h_n s_{n-1} u| = \langle h_0 \dots h_n t_{n-1} v| \text{ implies } \langle s_{n-1} u| = \langle t_{n-1} v|.$$

We know that $\langle f_0 \dots f_n | = \langle g_0 \dots g_n |$ and thus,

$$\langle h_0 \dots h_{n-1} s_{n-1} f_n | = \langle h_0 \dots h_{n-1} t_{n-1} g_n |,$$

which, by induction hypothesis, implies $\langle s_{n-1} f_n | = \langle t_{n-1} g_n |$. By productivity, there exists h_n with $s_{n-1} f_n = h_n s_n$ and $t_{n-1} g_n = h_n t_n$ such that $\langle h_n s_n u| = \langle h_n t_n v|$ implies $\langle s_n u| = \langle t_n v|$.

Finally, assume that $\langle h_0 \dots h_{n-1} h_n s_n u| = \langle h_0 \dots h_{n-1} h_n t_n v|$. Thus, we have $\langle h_0 \dots h_{n-1} s_{n-1} f_n u| = \langle h_0 \dots h_{n-1} t_{n-1} g_n v|$ and $\langle s_{n-1} f_n u| = \langle t_{n-1} g_n v|$. This can be rewritten as $\langle h_n s_n u| = \langle h_n t_n v|$, which in turn implies $\langle s_n u| = \langle t_n v|$. We have shown that the h_n that we constructed satisfies the desired property. □

Lemma D.5 (Removing the first step)**.** *In a productive category, let $\langle f_0| = \langle g_0|$. Then there exists h with $f_0 = hs$ and $g_0 = ht$ such that $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$ implies $\langle s f_1| \dots |f_n| = \langle t g_1| \dots |g_n|$.*

Proof. By Lemma D.4, we obtain a factorization $f_0 = hs$ and $g_0 = ht$. Moreover, each time that we have $\langle f_0| \dots |f_n| = \langle g_0| \dots |g_n|$, we can obtain a family h_i together with s_i and t_i such that $s_{i-1} f_i = h_i s_i$ and $t_{i-1} f_i = h_i t_i$. Using the fact that $\langle h_n s_n| = \langle h_n t_n|$, we have that $\langle h_1| \dots |h_n s_n| = \langle h_1| \dots |h_n t_n|$, which can be rewritten using dinaturality as $\langle s_0 f_1| \dots |f_n| = \langle t_0 g_1| \dots |g_n|$. □

Lemma D.6. *In a productive category, the final coalgebra of the endofunctor $(\mathsf{hom} \odot \bullet)$ does exist and it is given by the limit of the terminal sequence*

$$L \coloneqq \lim_n \int^{M_0,\dots,M_n} \prod_{i=0}^{n} \mathsf{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

Proof. We will apply Theorem 1.5. The endofunctor $(\mathsf{hom} \odot \bullet)$ acts on the category $[([\mathbb{N}, \mathsf{C}])^{op} \times [\mathbb{N}, \mathsf{C}], \mathsf{Set}]$, which, being a presheaf category, has all small limits. We will show that there is an isomorphism $\mathsf{hom} \odot L \cong L$ given by the canonical morphism between them.

First, note that the set $L(\mathbb{X}; \mathbb{Y})$ is, explicitly,

$$\lim_n \int^{M_1,\dots,M_n} \prod_{i=1}^{n} \mathsf{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

A generic element from this set is a *sequence of sequences of increasing length*. Moreover, the sequences must *truncate coherently* (Definition D.1).

Secondly, note that the set $(\mathsf{hom} \odot L)(\mathbb{X}; \mathbb{Y})$ is, explicitly,

$$\int^{M_0} \mathsf{hom}(X_0, Y_0 \otimes M_0) \times \lim_n \int^{M_1,\dots,M_n} \prod_{i=1}^{n} \mathsf{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i).$$

A generic element from this set is of the form

$$\langle f | (\langle f_1^k | \dots | f_k^k |)_{k \in \mathbb{N}} |,$$

that is, a pair consisting on a first morphism $f \colon X_0 \to Y_0 \otimes M_0$ and a family of sequences $(\langle f_1^k| \dots |f_k^k|)$, quotiented by dinaturality of M_0 and truncating coherently. The canonical map to $L(\mathbb{X}; \mathbb{Y})$ maps this generic element to the family of sequences $(\langle f_0 | f_1^k | \dots | f_k^k |)_{k \in \mathbb{N}}$, which truncates coherently because the previous family did and we are precomposing with f_0, which is dinatural.

Thirdly, this map is injective. Imagine a pair of elements $\langle f_0 | (\langle f_1^k | \dots | f_k^k |)_{k \in \mathbb{N}} |$ and $\langle g_0 | (\langle g_1^k | \dots | g_k^k |)_{k \in \mathbb{N}} |$ that have the same image, meaning that, for each $k \in \mathbb{N}$,

$$\langle f_0 | f_1^k | \dots | f_k^k | = \langle g_0 | g_1^k | \dots | g_k^k | .$$

By Lemma D.5, we can find h with $f_0 = hs$ and $g_0 = ht$ such that, for each $k \in \mathbb{N}$, $\langle s f_1^k | \dots | f_k^k | = \langle t g_1^k | \dots | g_k^k |$. Thus,

$$\langle f_0 | (\langle f_1^k | \dots | f_k^k |)_{k \in \mathbb{N}} | = \langle h | (\langle s f_1^k | \dots | f_k^k |)_{k \in \mathbb{N}} | =$$
$$\langle h | (\langle t g_1^k | \dots | g_k^k |)_{k \in \mathbb{N}} | = \langle g_0 | (\langle t g_1^k | \dots | g_k^k |)_{k \in \mathbb{N}} | .$$

Finally, this map is also surjective. From Lemma D.2, it follows that any family that truncates coherently can be equivalently written as $\langle h_0 | \dots | h_n |_{n \in \mathbb{N}}$, which is the image of the element $\langle h_0 | \langle h_1 | \dots | h_n |_{n \in \mathbb{N}} |$. □

Theorem D.7 (From Theorem 4.11)**.** *In a productive category, the final coalgebra of the endofunctor $(\mathsf{hom} \odot \bullet)$ exists and it is given by the set of stateful sequences quotiented by observational equivalence.*

$$\left(\int^{M \in [\mathbb{N}, \mathsf{C}]} \prod_{i=0}^{\infty} \mathsf{hom}(X_i \otimes M_{i-1}, Y_i \otimes M_i) \right) \Big/ \approx$$

Proof. By Lemma D.6, we know that the final coalgebra exists and is given by the limit of the terminal sequence. By Lemma D.3, we know that it is isomorphic to the set of stateful sequences quotiented by observational equivalence. □

E Monoidal streams

Lemma E.1. *Sequential composition of streams with memories (Definition 5.3) is associative. Given three streams*

- $f \in \mathsf{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$,
- $g \in \mathsf{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$,
- *and* $h \in \mathsf{Stream}(B \cdot \mathbb{Z}, \mathbb{W})$;

we can compose them in two different ways,

- $(f^A ; g^B) ; h^C \in \mathsf{Stream}((A \otimes B) \otimes C \cdot \mathbb{X}, \mathbb{W})$, *or*
- $f^A ; (g^B ; h^C) \in \mathsf{Stream}(A \otimes (B \otimes C) \cdot \mathbb{X}, \mathbb{W})$.

We claim that

$$((f^A ; g^B) ; h^C) = \alpha_{A,B,C} \cdot (f^A ; (g^B ; h^C)).$$

Proof. First, we note that both sides of the equation represent streams with different memories.

- $M((f^A ; g^B) ; h^C) = (M(f) \otimes M(g)) \otimes M(h)$,
- $M(f^A ; (g^B ; h^C)) = M(f) \otimes (M(g) \otimes M(h))$.

We will prove they are related by dinaturality over the associator α. We know that $\mathsf{now}((f^A ; g^B) ; h^C) = \mathsf{now}(f^A ; (g^B ; h^C))$ by string diagrams (see Figure 26). Then, by coinduction, we know that

$$(\mathsf{later}(f)^{M(f)} ; \mathsf{later}(g)^{M(g)}) ; \mathsf{later}(h)^{M(h)} =$$
$$\alpha \cdot (\mathsf{later}(f)^{M(f)} ; (\mathsf{later}(g)^{M(g)} ; \mathsf{later}(h)^{M(h)})),$$

that is, $\mathsf{later}((f^A ; g^B) ; h^C) = \alpha \cdot \mathsf{later}(f^A ; (g^B ; h^C))$. □

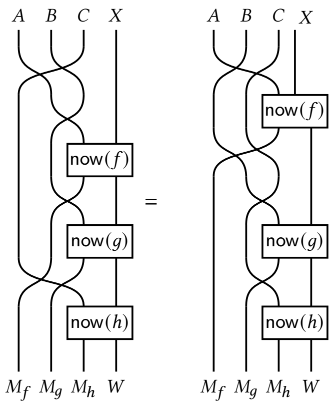


Figure 26. Associativity for sequential composition

Lemma E.2. *Sequential composition of streams (Definition 5.3) is associative. Given three streams*

- $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$,
- $g \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$,
- *and* $h \in \mathsf{Stream}(\mathbb{Z}, \mathbb{W})$;

we claim that $((f;g);h) = (f;(g;h))$.

Proof. Direct consequence of Lemma E.1, after considering the appropriate coherence morphisms. □

Lemma E.3. *Parallel composition of streams with memories is functorial with regards to sequential composition of streams with memories. Given four streams*

- $f \in \mathsf{Stream}(A \cdot \mathbb{X}, \mathbb{Y})$,
- $f' \in \mathsf{Stream}(A' \cdot \mathbb{X}', \mathbb{Y}')$,
- $g \in \mathsf{Stream}(B \cdot \mathbb{Y}, \mathbb{Z})$, *and*
- $g' \in \mathsf{Stream}(B' \cdot \mathbb{Y}', \mathbb{Z}')$,

we can compose them in two different ways,

- $(f^A \otimes f'^{A'})^{(A \otimes A')} \,;\, (g^B \otimes g'^{B'})^{(B \otimes B')}$, *and*
- $(f^A \,;\, g^B)^{(A \otimes B)} \otimes (f'^{A'} \,;\, g'^{B'})^{(A' \otimes B')}$,

having slightly different types, respectively,

- $\mathsf{Stream}((A \otimes A') \otimes (B \otimes B') \cdot \mathbb{X} \otimes \mathbb{X}', \mathbb{Z} \otimes \mathbb{Z}')$, *and*
- $\mathsf{Stream}((A \otimes B) \otimes (A' \otimes B') \cdot \mathbb{X} \otimes \mathbb{X}', \mathbb{Z} \otimes \mathbb{Z}')$.

We claim that

$$(f^A \otimes f'^{A'})^{(A \otimes A')} \,;\, (g^B \otimes g'^{B'})^{(B \otimes B')} =$$
$$\sigma_{A',B} \cdot (f^A \,;\, g^B)^{(A \otimes B)} \otimes (f'^{A'} \,;\, g'^{B'})^{(A' \otimes B')}.$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent strams with different memories.

- $M(LHS) = (M(f) \otimes M(f')) \otimes (M(g) \otimes M(g'))$,
- $M(RHS) = (M(f) \otimes M(g)) \otimes (M(f') \otimes M(g'))$.

We will prove they are related by dinaturality over the symmetry σ. We know that $\mathsf{now}(LHS); \sigma = \mathsf{now}(RHS)$ by string diagrams (see Figure 27). Then, by coinduction, we know that $\mathsf{later}(LHS) = \sigma \cdot \mathsf{later}(RHS)$. □

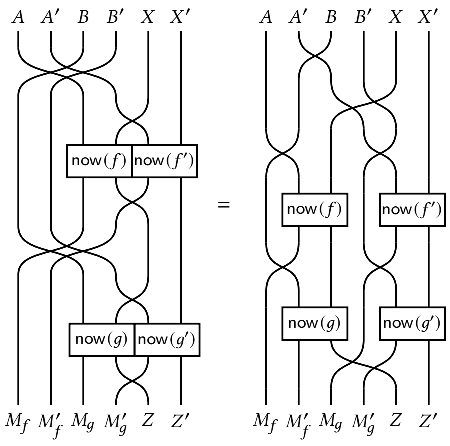


Figure 27. Functoriality of parallel composition.

Lemma E.4. *Parallel composition of streams is functorial with respect to sequential composition of streams. Given four streams*

- $f \in \mathsf{Stream}(\mathbb{X}, \mathbb{Y})$,
- $f' \in \mathsf{Stream}(\mathbb{X}', \mathbb{Y}')$,
- $g \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$, *and*
- $g' \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Z})$;

we claim that $(f \otimes f'); (g \otimes g') = (f;g) \otimes (f';g')$.

Proof. Direct consequence of Lemma E.3, after considering the appropriate coherence morphisms. □

Theorem E.5 (see [65]). *Monoidal streams over a symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a symmetric monoidal category* Stream.

Proof. Sequential composition of streams (Definition 5.3) is associative (Lemma E.2) and unital with respect to identities. Parallel composition is bifunctorial with respect to sequential composition (Lemma E.4); this determines a bifunctor, which is the tensor of the monoidal category. The coherence morphisms and the symmetry can be included from sets, so they still satisfy the pentagon and triangle equations. □

Lemma E.6. *The structure* $(\mathsf{Stream}, \mathsf{fbk})$ *with memories satisfies the tightening axiom (A1). Given three streams*

- $u \in \mathsf{Stream}(A \cdot \mathbb{X}', \mathbb{X})$,
- $f \in \mathsf{Stream}(B \otimes T \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $v \in \mathsf{Stream}(C \cdot \mathbb{Y}, \mathbb{Y}')$;

we claim that

$$\mathsf{fbk}^S(u^A; f^B; v^C) = \sigma \cdot u^A; \mathsf{fbk}^S(f^{B \otimes T}); v^C.$$

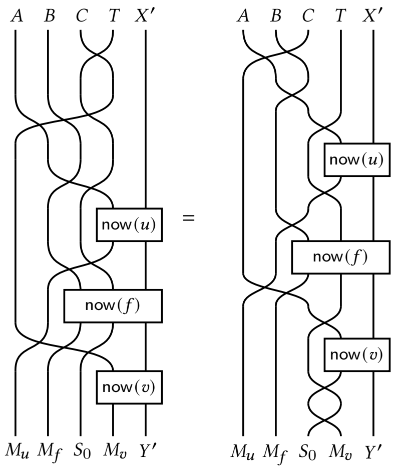


Figure 28. The tightening axiom (A1).

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = A \otimes B \otimes C \otimes T$,
- $M(RHS) = A \otimes B \otimes T \otimes C$.

We will prove that they are related by dinaturality over the symmetry σ. We know that $\mathsf{now}(LHS); \sigma = \mathsf{now}(RHS)$ by string diagrams (see Figure 28). Then, by coinduction, we know that $\mathsf{later}(LHS) = \sigma \cdot \mathsf{later}(RHS)$. □

Lemma E.7. *The structure* (Stream, fbk) *satisfies the tightening axiom (A1). Given streams*

- $u \in \mathsf{Stream}(\mathbb{X}', \mathbb{X})$,
- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $v \in \mathsf{Stream}(\mathbb{Y}, \mathbb{Y}')$*;*

we claim that $\mathrm{fbk}^S(u; f; v) = u; \mathrm{fbk}^S(f); v$.

Proof. Consequence of Lemma E.6, after applying the necessary coherence morphisms. □

Lemma E.8. *The structure* (Stream, fbk) *with memories satisfies the vanishing axiom (A2). Given a stream*

- $f \in \mathsf{Stream}(A \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that $\mathrm{fbk}^I(f^A) = \rho \cdot f$.

Proof. First, we note that both sides of the equation represent streams with different memories, $M(\mathrm{fbk}^I(f^A)) = M(f) \otimes I$. We will prove that they are related by dinaturality over the right unitor ρ. We know that $\mathsf{now}(\mathrm{fbk}^I(f^A)) = \mathsf{now}(f)$ by definition. Then, by coinduction, we konw that $\mathsf{later}(\mathrm{fbk}^I(f^A)) = \rho \cdot \mathsf{later}(f)$. □

Lemma E.9. *The structure* (Stream, fbk) *satisfies the vanishing axiom (A2). Given a stream* $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *we claim that* $\mathrm{fbk}^I(f) = f$.

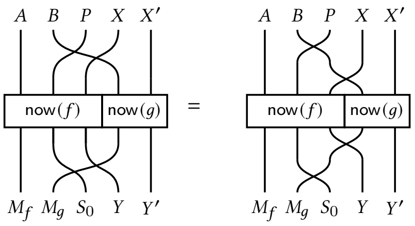


Figure 29. The strength axiom (A4).

Proof. Consequence of Lemma E.8, after applying the necessary coherence morphisms. □

Lemma E.10. *The structure* (Stream, fbk) *with memories satisfies the joining axiom (A3). Given a stream*

- $f \in \mathsf{Stream}((A \otimes P \otimes Q) \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that

$$\mathrm{fbk}^{S \otimes T}(f^{A \otimes (P \otimes Q)}) = \alpha \cdot \mathrm{fbk}^T(\sigma \cdot \mathrm{fbk}^S(\sigma \cdot f^{(A \otimes Q) \otimes P})).$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent strams with different memories.

- $M(LHS) = M(f) \otimes (S_0 \otimes T_0)$,
- $M(RHS) = (M(f) \otimes S_0) \otimes T_0$.

We will prove that they are related by dinaturality over the associator α. We know that $\mathsf{now}(LHS); \alpha = \mathsf{now}(RHS)$ by definition. Then, by coinduction, we know that $\mathsf{later}(LHS) = \alpha \cdot \mathsf{later}(RHS)$. □

Lemma E.11. *The structure* (Stream, fbk) *satisfies the joining axiom (A3). Given a stream*

- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$,

we claim that $\mathrm{fbk}^{S \otimes T}(f) = \mathrm{fbk}^T(\mathrm{fbk}^S(f))$.

Proof. Consequence of Lemma E.10, after applying the necessary coherence morphisms. □

Lemma E.12. *The structure* (Stream, fbk) *with memories satisfies the strength axiom (A4). Given two streams*

- $f \in \mathsf{Stream}((A \otimes P) \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $g \in \mathsf{Stream}(B \cdot \mathbb{X}', \mathbb{Y}')$,

we claim that

$$\alpha \cdot \mathrm{fbk}^S(f^A \otimes g^B) = \mathrm{fbk}^S(f^{A \otimes P})^{A \otimes P} \otimes g^B.$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = (M(f) \otimes S_0) \otimes M(g)$,
- $M(RHS) = M(f) \otimes (S_0 \otimes M(g))$.

We will prove that they are related by dinaturality over the symmetry α. We know that $\mathsf{now}(LHS) = \mathsf{now}(RHS); \alpha$ by string diagrams (see Figure 29). Then, by coinduction, we know that $\alpha \cdot \mathsf{later}(LHS) = \mathsf{later}(RHS)$. □

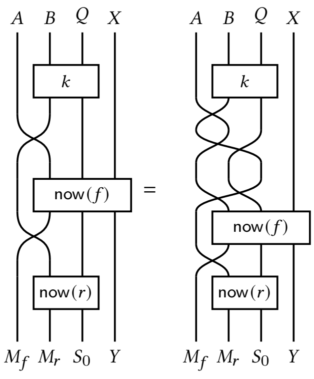


Figure 30. The sliding axiom (A5).

Lemma E.13. *The structure* (Stream, fbk) *with memories satisfies the strength axiom (A4). Given two streams*

- $f \in \mathsf{Stream}(\mathbb{S} \otimes \mathbb{X}, \mathbb{S} \otimes \mathbb{Y})$, *and*
- $g \in \mathsf{Stream}(\mathbb{X}', \mathbb{Y}')$,

we claim that

$$\alpha \cdot \mathsf{fbk}^S(f \otimes g) = \mathsf{fbk}^S(f) \otimes g.$$

Proof. Consequence of Lemma E.12, after applying the necessary coherence morphisms. □

Lemma E.14. *The structure* (Stream, fbk) *with memories satisfies the sliding axiom (A5). Given two streams*

- $f \in \mathsf{Stream}(A \cdot \partial\mathbb{S} \otimes \mathbb{X}, \mathbb{T} \otimes \mathbb{Y})$, *and*
- $r \in \mathsf{Stream}(C \cdot \mathbb{T}, \mathbb{S})$

we claim that, for each $k \colon B \otimes Q \to C \otimes P$,

$$k \cdot \mathsf{fbk}^{\mathbb{S}}(f^{A\otimes P}; (r^C \otimes \mathrm{id})) = \mathsf{fbk}^{\mathbb{T}}(k \cdot (\partial r^C \otimes \mathrm{id}); f^{A\otimes P}).$$

Proof. First, we note that both sides of the equation (which, from now on, we call *LHS* and *RHS*, respectively) represent streams with different memories.

- $M(LHS) = M(f) \otimes M(r) \otimes S_0$,
- $M(RHS) = M(r) \otimes M(f) \otimes T_0$.

We will prove that they are related by dinaturality over the symmetry and the first action of r, that is, $\sigma; (\mathsf{now}(r) \otimes \mathrm{id})$. We know that $\mathsf{now}(LHS) = \mathsf{now}(RHS); (\sigma; (\mathrm{id} \otimes \mathsf{now}(r)))$ by string diagrams (see Figure 30). Using coinduction,

$$\begin{aligned}
&(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathsf{later}(LHS) \\
=&(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathsf{fbk}^{\mathbb{S}}(\mathsf{later}(f)^{M(f)\otimes S_0}; (\mathsf{later}(r)^{M(r)} \otimes \mathrm{id})) \\
=&\mathsf{fbk}^{\mathbb{T}}((\sigma; \underline{\mathsf{now}(r)}) \cdot (\partial\, \mathsf{later}(r)^{M(r)} \otimes \mathrm{id}); \mathsf{later}(f)^{M(f)\otimes S_0}) \\
=&\mathsf{fbk}^{\mathbb{T}}(\sigma \cdot (\mathsf{later}(\partial r)^{M(r)} \otimes \mathrm{id}); \mathsf{later}(f)^{M(f)\otimes S_0}) \\
=&\,\mathsf{later}(RHS),
\end{aligned}$$

we show that $(\sigma; \underline{\mathsf{now}(r)}) \cdot \mathsf{later}(LHS) = \mathsf{later}(RHS)$. □

Lemma E.15. *The structure* (Stream, fbk) *satisfies the sliding axiom (A5). Given two streams*

- $f \in \mathsf{Stream}(\partial\mathbb{S} \otimes \mathbb{X}, \mathbb{T} \otimes \mathbb{Y})$, *and*
- $r \in \mathsf{Stream}(\mathbb{T}, \mathbb{S})$

we claim that, $\mathsf{fbk}^{\mathbb{S}}(f; (r \otimes \mathrm{id})) = \mathsf{fbk}^{\mathbb{T}}((\partial r \otimes \mathrm{id}); f)$.

Proof. Consequence of Lemma E.14, after applying the necessary coherence morphisms. □

Theorem E.16 (From Theorem 5.10)**.** *Monoidal streams over a symmetric monoidal category* $(\mathsf{C}, \otimes, I)$ *form a ∂-feedback monoidal category* (Stream, fbk).

Proof. We have proven that it satisfies all the feedback axioms:

- the tightening axiom by Lemma E.7,
- the vanishing axiom by Lemma E.9,
- the joining axiom by Lemma E.11,
- the strength axiom by Lemma E.13,
- and the sliding axiom by Lemma E.15.

Thus, it is a feedback structure □

F Type theory

As our base type theory, we consider the that of symmetric monoidal categories over some generators forming a multigraph $\mathcal{G}$, as described by Shulman [70]. Our only change will be to consider an *unbiased presentation*, meaning that we consider n-ary tensor products instead of only binary and 0-ary (the monoidal unit). This reduces the number of rules: the binary case and the 0-ary case of the usual presentation are taken care of by a single n-ary case.

Definition F.1 (Shuffling contexts [70])**.** A *shuffle of contexts* $\mathrm{Shuf}(\Gamma_1, \dots, \Gamma_n)$ is the result of a permutation of $\bigsqcup_{i=0}^{n} \Gamma_n$ that leaves invariant the internal order of each Γ_i. Shuffles allow us to derive morphisms that make use of the symmetry without introducing redundancy in our type theory.

Substitution is admissible in the type theory for symmetric monoidal categories [70]. It can be inductively defined: we write $P[m/x]$ for the substitution of the variable x by a term m inside the term p. Using substitution, we can state a pair of β/η-reduction equalities for the terms of our type theory.

Gen
$$\frac{f \in \mathcal{G}(A_1, \dots, A_n; B) \qquad \Gamma_1 \vdash x_1 : A_1 \dots \Gamma_n \vdash x_n : A_n}{\mathsf{Shuf}(\Gamma_1, \dots, \Gamma_n; \Gamma) \vdash f(x_1, \dots, x_n) : B}$$

Pair
$$\frac{\Gamma_1 \vdash x_1 : A_1 \qquad \dots \qquad \Gamma_n \vdash x_n : A_n}{\mathsf{Shuf}(\Gamma, \Delta) \vdash [x_1, \dots, x_n] : A_1 \otimes \dots \otimes A_n}$$

Var
$$\frac{}{x : A \vdash x : A}$$

Split
$$\frac{\Delta \vdash m : A_1 \otimes \dots \otimes A_n \qquad \Gamma, x_1 : A_1, \dots, x_n : A_n \vdash z : C}{\mathsf{Shuf}(\Gamma_1, \dots, \Gamma_n) \vdash \textsc{Split}\ m \longrightarrow [x_1, \dots, x_n]\ \textsc{in}\ z : C}$$

Figure 31. Type theory of symm. monoidal categories [70].

- $\textsc{Split}\ [E_1, \dots, E_n] \to [x_1, \dots, x_n]\ \textsc{in}\ z \equiv z[E_1/x_1, \dots]$
- $\textsc{Split}\ [E_1, \dots, E_n] \to [x_1, \dots, x_n]\ \textsc{in}\ N[[x_1, \dots, x_n]/u]$ $\equiv N[[E_1, \dots, E_n]/u]$

F.1 Type theory for a strong monoidal endofunctor

Definition F.2 (Signature)**.** Let T be a set of basic types. We write

$$T^{\star}_{\partial} := \{\partial^{n_1} t_1 \partial^{n_2} t_2 \dots \partial^{n_k} t_k \mid n_1, \dots, n_k \in \mathbb{N}, t_i \in T\}$$

for the free monoid-with-an-endomorphism $\partial : T^{\star}_{\partial} \to T^{\star}_{\partial}$ over T. The **signature** for a type theory of monoidal categories with a monoidal endofunctor is given by a pair of functions $s, t \colon O \rightrightarrows T^{\star}_{\partial}$, assigning source and target to every generator from a set O.

Example F.3. We will usually include two families of generators in our theory. For each type $t_0 \in T$ that can be copied, we have a $\mathsf{copy} \in O$ generator with $s(\mathsf{copy}) = t_0 \in T^{\star}_{\partial}$ and $t(\mathsf{copy}) = t_0 \cdot t_0 \in T^{\star}_{\partial}$.

Definition F.4. The type theory of symmetric monoidal categories with a symmetric monoidal endofunctor over a signature $O \rightrightarrows T^{\star}_{\partial}$ is the type theory of symmetric monoidal categories extended with an operator ∂ on types such that

$$\partial[] = [] \quad \text{and} \quad \partial(\Gamma, x{:}A) = \partial\Gamma, x{:}\partial A,$$

and the following $\textsc{Delay}$ introduction rule.

$$\begin{array}{l}\textsc{Delay}\\ \dfrac{\Gamma \vdash x : A}{\partial\Gamma \vdash \textsc{Delay}(x) : \partial A}\end{array}$$

The delay operator is a monoid homomorphism on types, satisfying $\partial I \equiv I$ and $\partial(A \otimes B) \equiv \partial A \otimes \partial B$. The $\textsc{Delay}$ introduction rule satisfies the following conversion equalities, that state that it acts as a functor preserving the monoidal structure.

- $\textsc{Delay}(x) \equiv x$ for x a variable,
- $[\textsc{Delay}(e_1), \dots, \textsc{Delay}(e_n)] \equiv \textsc{Delay}([e_1, \dots, e_n])$,
- $\textsc{Split}\ \textsc{Delay}(m) \to [e_1, \dots, e_n]\ \textsc{in}\ \textsc{Delay}(z)$

  $\equiv \textsc{Delay}(\textsc{Split}\ m \to [e_1, \dots, e_n]\ \textsc{in}\ z)$,

We choose not to explicitly state the delay rule when writing terms of the type theory (as we do in Section 8.2), as it does not cause ambiguity with the typing. However, we do write it when typechecking.

F.2 Type Theory for Delayed Feedback

We finally augment the type theory of symmetric monoidal categories with a monoidal endofunctor by adding the following derivation rule.

$$\begin{array}{l}\textsc{Fbk}\\ \dfrac{\Gamma, s : \partial S \vdash x(s) : S \otimes A}{\Gamma \vdash \textsc{Fbk}\ s.\ x(s) : A}\end{array}$$

Following the axioms of categories with delayed feedback and their normalization theorem, we introduce rules that follow from the feedback axioms. Note that these rules simplify multiple applications of feedback into a single application at the head of the term.

1. $g(\textsc{Fbk}\ s\ \textsc{in}\ x(s)) \equiv$
   $\textsc{Fbk}\ s\ \textsc{in}\ \textsc{Let}\ [t, b] \leftarrow x(s)\ \textsc{in}\ [t, g(b)]$
2. $[\textsc{Fbk}\ s\ \textsc{in}\ x(s), \textsc{Fbk}\ t\ \textsc{in}\ y(t)] \equiv \textsc{Fbk}\ m\ \textsc{in}$
   $\textsc{Split}\ m \to [s, t]\ \textsc{in}$
   $\textsc{Split}\ x(s) \to [u, v]\ \textsc{in}$
   $\textsc{Split}\ y(t) \to [u', v']\ \textsc{in}$
   $[[u, u'], [v, v']]$
3. $\textsc{Split}\ m \to [x_1, \dots, x_n]\ \textsc{in}\ \textsc{Fbk}\ s\ \textsc{in}\ z \equiv$
   $\textsc{Fbk}\ s\ \textsc{in}\ \textsc{Split}\ m \to [x_1, \dots, x_n]\ \textsc{in}\ z$
4. $\textsc{Split}\ (\textsc{Fbk}\ u\ \textsc{in}\ z(u)) \to [x_1, \dots, x_n]\ \textsc{in}\ m \equiv$
   $\textsc{Fbk}\ u\ \textsc{in}\ \textsc{Split}\ z(u) \to [v, n]\ \textsc{in}$
   $\textsc{Split}\ n \to [x_1, \dots, x_n]\ \textsc{in}\ [v, m]$
5. $x \equiv \textsc{Fbk}\ i\ \textsc{in}\ \textsc{Split}\ i \to []\ \textsc{in}\ [t, x]$
6. $\textsc{Fbk}\ x\ \textsc{in}\ \textsc{Fbk}\ y\ \textsc{in}\ m(x, y) \equiv$
   $\textsc{Fbk}\ n\ \textsc{in}\ \textsc{Split}\ n \to [x, y]\ \textsc{in}\ m(x, y)$

Finally, we introduce an equality representing the *sliding axiom* (Figure 7). Having all of these rules together means that we can always rewrite a term in the type theory with feedback as a term of the type theory of symmetric monoidal categories *up to sliding*. These are precisely the morphisms of the $\mathsf{St}(\bullet)$-construction, the free category with feedback.

1. $\textsc{Fbk}\ s\ \textsc{in}\ f(\textsc{Delay}(h(s)), x) \equiv$
   $\textsc{Fbk}\ t\ \textsc{in}\ \textsc{Split}\ f(x, t) \to [y, s]\ \textsc{in}\ [y, h(s)]$

F.3 Categories with copy and syntax sugar

Definition F.5. A *category with copying* is a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ in which every object $A \in \mathsf{C}$ has a (non-necessarily natural) coassociative and cocommutative comultiplication $\delta_A = (\blacktriangle)_A \colon A \to A \otimes A$, called the "copy".

Every cartesian category and every kleisli category of a Set-based commutative monad is a category with copying. In our type theory, this is translated into a $\textsc{Copy}$ generator acting as follows.

$$\begin{array}{l}\textsc{Copy}\\ \dfrac{\Gamma \vdash x : A}{\Gamma \vdash \textsc{Copy}(x) : A \otimes A}\end{array}$$

Definition F.6. A *category with ∂-merging* is a symmetric monoidal category $(\mathsf{C}, \otimes, I)$ with a symmetric monoidal endofunctor $\partial \colon \mathsf{C} \to \mathsf{C}$ in which every object $A \in \mathsf{C}$ has an associated morphism $\phi_A \colon A \otimes \partial A \to A$.

In our type theory, this is translated by a $\textsc{Fby}$ generator acting as follows.

$$\begin{array}{l}\textsc{Fby}\\ \dfrac{\Gamma \vdash x : A \qquad \Delta \vdash y : \partial(A)}{\mathsf{Shuf}(\Gamma, \Delta) \vdash x\ \textsc{Fby}\ y : A}\end{array}$$

We allow three pieces of syntax sugar in our language, suited only for the case of categories with copying. These make the language more LUCID-like without changing its formal description.

1. We allow multiple occurences of a variable, implicily copying it.
2. We apply DELAY rules where needed for type-checking, without explicitly writing the rule.
3. Recursive definitions are syntax for the FBK rule and the COPY rule. That is,

$$M = x(M) \quad \text{means} \quad M = \textsc{Fbk}\ m \text{ IN } \textsc{Copy}(x(m)).$$

4. We use WAIT to declare an implicit feedback loop.

$$\textsc{Wait}(x) \quad \text{means} \quad \textsc{Fbk}\ y \text{ IN } [x, y].$$

G Implementation

We use the **Haskell** [39] programming language for computations. We use `Arrows` [40] to represent monoidal categories with an identity-on-objects monoidal functor from our base category of Haskell types and functions. Notations for arrows [61] have been explained in terms of Freyd categories [63]. In particular, the `loop` notation is closely related to feedback, as it is usually employed to capture *traces*.

Our definition of monoidal streams follows Definition 5.1.

```
type Stream c = StreamWithMemory c ()

data StreamWithMemory c n x y where
  StreamWithMemory :: (Arrow c) =>
     c (n , x) (m , y)
    -> StreamWithMemory c m x y
    -> StreamWithMemory c n x y
```

Their sequential and parallel composition (`comp` and `tensor`) follow from Definitions 5.3 and 5.5. In Appendix G.2 we describe both first on the now part; and then trivially extended by coinduction.

$$\mathsf{now}(f ;_N g) = (\sigma \otimes \mathrm{id}_A);(\mathrm{id} \otimes \mathsf{now}(f));(\sigma \otimes \mathrm{id}_B);\,(\mathrm{id} \otimes \mathsf{now}(g)).$$

$$\mathsf{now}(f \otimes_N g) = (\mathrm{id} \otimes \sigma \otimes \mathrm{id});\,(\mathsf{now}(f) \otimes \mathsf{now}(g));\,(\mathrm{id} \otimes \sigma \otimes \mathrm{id}).$$

Example G.1 (Fibonacci example). The code for `fibonacci` in Appendix G.2 follows the definition in Section 6.3. We can execute it to obtain the first 10 numbers from the Fibonacci sequence.

```
> take 10 <$> run fibonacci
Identity [0,1,1,2,3,5,8,13,21,34]
```

Example G.2 (Random walk example). The code for `walk` in Appendix G.2 follows the definition in Example 7.4. We can execute it multiple times to obtain different random walks starting from 0.

```
> take 10 <$> run walk
[0,1,0,-1,-2,-1,-2,-3,-2,-3]
> take 10 <$> run walk
[0,1,2,1,2,1,2,3,4,5]
> take 10 <$> run walk
[0,-1,-2,-1,-2,-1,0,-1,0,-1]
```

Example G.3 (Ehrenfest model). The code for `ehrenfest` in Appendix G.2 follows the definition in Figure 12. We can execute it to simulate the Ehrenfest model.

```
> take 10 <$> run ehrenfest
[([2,3,4],[1]),([2,3],[1,4]),
([2,3,4],[1]),([2,4],[1,3]),
([2],[1,3,4]),([2,4],[1,3]),
([2],[1,3,4]),([2,3],[1,4]),
([3],[1,2,4]),([2,3],[1,4])]
```

G.1 Term derivations

Example G.4 (Fibonacci).

$$\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathbb{N}_0} \quad \dfrac{\dfrac{\dfrac{\dfrac{}{f : \mathbb{N} \vdash f : \mathbb{N}}}{f : \mathbb{N} \vdash \textsc{Copy}(f) : \mathbb{N} \otimes \mathbb{N}} \quad \dfrac{\dfrac{}{f_1 : \mathbb{N} \vdash f_1 : \mathbb{N}} \quad \dfrac{\dfrac{}{\vdash 1 : \mathbb{N}_0} \quad \dfrac{\dfrac{}{f_2 : \mathbb{N} \vdash f_2 : \mathbb{N}}}{f_2 : \mathbb{N} \vdash \textsc{Wait}(f_2) : \partial\mathbb{N}}}{f_2 : \mathbb{N} \vdash 1\ \textsc{Fby}\ \textsc{Wait}(f_2) : \mathbb{N}}}{f_1 : \mathbb{N}, f_2 : \mathbb{N} \vdash f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2) : \mathbb{N}}}{f : \mathbb{N} \vdash \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2)) : \mathbb{N}}}{f : \partial\mathbb{N} \vdash \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2)) : \partial\mathbb{N}}}{f : \partial\mathbb{N} \vdash 0\ \textsc{Fby}\ \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2)) : \mathbb{N}}}{\dfrac{f : \partial\mathbb{N} \vdash \textsc{Copy}(0\ \textsc{Fby}\ \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2))\ ) : \mathbb{N} \otimes \mathbb{N}}{\vdash : \textsc{Fbk}\ f\ \textsc{in}\ \textsc{Copy}(0\ \textsc{Fby}\ \textsc{Split}\ \textsc{Copy}(f) \to [f_1, f_2]\ \textsc{in}\ (f_1 + 1\ \textsc{Fby}\ \textsc{Wait}(f_2))\ ) : \mathbb{N}}}$$

Example G.5 (Random walk).

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash 0 : \mathbb{N}_0} \quad \dfrac{\dfrac{}{\vdash \textsc{Uniform}(-1,1) : \mathbb{N}} \quad \dfrac{}{w : \partial\mathbb{N} \vdash w : \partial\mathbb{N}}}{w : \partial\mathbb{N} \vdash \textsc{Uniform}(-1,1) + w : \partial\mathbb{N}}}{w : \partial\mathbb{N} \vdash 0\ \textsc{Fby}\ (\textsc{Uniform}(-1,1) + w) : \mathbb{N}}}{w : \partial\mathbb{N} \vdash \textsc{Copy}(0\ \textsc{Fby}\ (\textsc{Uniform}(-1,1) + w)) : \mathbb{N} \otimes \mathbb{N}}}{\vdash \textsc{Fbk}\ w\ \textsc{in}\ \textsc{Copy}(0\ \textsc{Fby}\ (\textsc{Uniform}(-1,1) + w)) : \mathbb{N}}$$

Example G.6 (Ehrenfest model).

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash (1,2,3,4) : \mathsf{Urn}_0 \quad \vdash () : \mathsf{Urn}_0}{\vdash [(1,2,3,4),()] : \mathsf{Urn}_0 \otimes \mathsf{Urn}_0} \quad \dfrac{\dfrac{\dfrac{}{u : \mathsf{Urn} \otimes \mathsf{Urn} \vdash u : \mathsf{Urn} \otimes \mathsf{Urn}} \quad \dfrac{\dfrac{\dfrac{}{\vdash \textsc{Uniform} : \mathbb{N}}}{\vdash \textsc{Copy}(\textsc{Uniform}) : \mathbb{N} \otimes \mathbb{N}} \quad \dfrac{\dfrac{\dfrac{}{n_1 : \mathbb{N} \vdash n_1 : \mathbb{N}} \quad \dfrac{}{u_1 : \mathsf{Urn} \vdash u_1 : \mathsf{Urn}}}{n_1 : \mathbb{N}, u_1 : \mathsf{Urn} \vdash \textsc{Move}(n_1, u_1) : \mathsf{Urn}} \quad \dfrac{\dfrac{}{n_2 : \mathbb{N} \vdash n_2 : \mathbb{N}} \quad \dfrac{}{u_2 : \mathsf{Urn} \vdash u_2 : \mathsf{Urn}}}{n_2 : \mathbb{N}, u_2 : \mathsf{Urn} \vdash \textsc{Move}(n_2, u_2) : \mathsf{Urn}}}{n_1 : \mathbb{N}, n_2 : \mathbb{N}, u_1 : \mathsf{Urn}, u_2 : \mathsf{Urn} \vdash [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathsf{Urn} \otimes \mathsf{Urn}}}{u_1 : \mathsf{Urn}, u_2 : \mathsf{Urn} \vdash \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathsf{Urn} \otimes \mathsf{Urn}}}{u : \mathsf{Urn} \otimes \mathsf{Urn} \vdash \textsc{Split}\ u \to [u_1, u_2]\ \textsc{in}\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathsf{Urn} \otimes \mathsf{Urn}}}{u : \partial(\mathsf{Urn} \otimes \mathsf{Urn}) \vdash \textsc{Split}\ u \to [u_1, u_2]\ \textsc{in}\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \partial(\mathsf{Urn} \otimes \mathsf{Urn})}}{u : \partial(\mathsf{Urn} \otimes \mathsf{Urn}) \vdash [(1,2,3,4),()]\ \textsc{Fby}\ \textsc{Split}\ u \to [u_1, u_2]\ \textsc{in}\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)] : \mathsf{Urn} \otimes \mathsf{Urn}}}{\dfrac{u : \partial(\mathsf{Urn} \otimes \mathsf{Urn}) \vdash \textsc{Copy}([(1,2,3,4),()]\ \textsc{Fby}\ \textsc{Split}\ u \to [u_1, u_2]\ \textsc{in}\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]) : (\mathsf{Urn} \otimes \mathsf{Urn}) \otimes (\mathsf{Urn} \otimes \mathsf{Urn})}{\vdash \textsc{Fbk}\ u\ \textsc{in}\ \textsc{Copy}([(1,2,3,4),()]\ \textsc{Fby}\ \textsc{Split}\ u \to [u_1, u_2]\ \textsc{in}\ \textsc{Split}\ \textsc{Copy}(\textsc{Uniform}) \to [n_1, n_2]\ \textsc{in}\ [\textsc{Move}(n_1, u_1), \textsc{Move}(n_2, u_2)]) : \mathsf{Urn} \otimes \mathsf{Urn}}}$$

G.2 Code

The following code has been compiled under GHCi, version 8.6.5. The "random" library may need to be installed.

```haskell
-- | Monoidal Streams for Dataflow Programming.

-- Anonymous.

-- The following code implements the category of monoidal streams over a
-- monoidal category with an identity-on-objects functor from the
-- (pseudo)category of Haskell types.

-- During the manuscript, we have needed to perform some computations: for
-- instance, to see that according to our definitions the Fibonacci morphism we
-- describe really computes the Fibonacci sequence. Computations of this kind
-- are difficult and tedious to write and to justify, and the reader may find
-- difficult to reproduce them. Instead of explicitly writing these
-- computations, we implement them and we provide the necessary code so that the
-- reader can verify the result from the computation.

-- Morphisms in a monoidal category are written in "Arrow" notation, using (>>>)
-- for sequential composition and (***) for parallel composition. Coherence
-- morphisms need to be written explicitly, we usually write them at the side of
-- the diagram.

{-# LANGUAGE GADTs                     #-}
{-# LANGUAGE TypeSynonymInstances      #-}
{-# LANGUAGE FlexibleInstances        #-}

module Main where

import Prelude hiding (id)
import Data.Functor.Identity
import Data.List
import Control.Category
import Control.Arrow
import System.Random
import System.IO.Unsafe


-- Fixpoint equation for monoidal streams. Figure 5.
type Stream c = StreamWithMemory c ()

data StreamWithMemory c n x y where
  StreamWithMemory :: (Arrow c) =>
     c (n , x) (m , y)
    -> StreamWithMemory c m x y
    -> StreamWithMemory c n x y


--------------
-- EXAMPLES --
--------------
fibonacci :: Stream (Kleisli Identity) () Int
fibonacci = fbk                             $ runitS
```

```
  >>> copy                               >>> lunitinvS *** id
  >>> delay (k1 *** wait) *** id
  >>> delay fby *** id
  >>> plus                               >>> lunitinvS
  >>> k0 *** id
  >>> fby
  >>> copy

walk :: Stream (Kleisli IO) (()) (Int)
walk = fbk
    $ (id *** unif)
  >>> plus                               >>> lunitinvS
  >>> k0 *** id
  >>> fby
  >>> copy
 where
   unif :: Stream (Kleisli IO) () Int
   unif = lift $ Kleisli (\() -> do
      boolean <- randomIO
      return $ if boolean then 1 else -1)

type Urn = [Int]

ehrenfest :: Stream (Kleisli IO) (()) (Urn,Urn)
ehrenfest = fbk                         $ runitS>>> lunitinv *** lunitinv
  >>> (full *** idS) *** (empty *** idS)    >>> runitinv
  >>> (fby *** fby) *** unif
  >>> idS *** copy                        >>> associnv >>> idS *** assoc
  >>> idS *** (sigma *** idS)             >>> idS *** associnv >>> assoc
  >>> move *** move
  >>> copy *** copy                        >>> associnv >>> idS *** assoc
  >>> idS *** (sigma *** idS)             >>> idS *** associnv >>> assoc
  where
    unif :: Stream (Kleisli IO) () Int
    unif = lift $ Kleisli (\() -> randomRIO (1,4))

    empty :: Stream (Kleisli IO) () Urn
    empty = lift $ arr (\() -> [])

    full :: Stream (Kleisli IO) () Urn
    full = lift $ arr (\() -> [1,2,3,4])

    move :: Stream (Kleisli IO) (Urn, Int) Urn
    move = lift $ arr (\(u,i) ->
      if elem i u
        then (delete i u)
        else (insert i u))

--- take 10 <$> run fibonacci
--- take 10 <$> run walk
--- take 10 <$> run ehrenfest
```

```
---------------------------
-- THE FEEDBACK CATEGORY --
---------------------------

compS :: (Arrow c) =>
  StreamWithMemory c m x y ->
  StreamWithMemory c n y z ->
  StreamWithMemory c (m , n) x z
compS
  (StreamWithMemory fnow flater)
  (StreamWithMemory gnow glater) =
  StreamWithMemory (sequentialComposition fnow gnow) (compS flater glater)
 where

   -- Definition 5.2.
   -- Sequential composition, "now".
   sequentialComposition :: Arrow c
     => c (m , x) (p , y)
     -> c (n , y) (q , z)
     -> c ((m,n),x) ((p,q),z)
   sequentialComposition f g =
        sigma  *** id    >>> associnv
    >>> id *** f       >>> assoc
    >>> sigma  *** id  >>> associnv
    >>> id *** g       >>> assoc

comp :: (Arrow c) => Stream c x y -> Stream c y z -> Stream c x z
comp f g = lact lunitinv (compS f g)



tensorS :: (Arrow c) =>
  StreamWithMemory c p x y ->
  StreamWithMemory c p' x' y' ->
  StreamWithMemory c (p , p') (x,x') (y,y')
tensorS
  (StreamWithMemory fnow flater)
  (StreamWithMemory gnow glater) =
  StreamWithMemory (parallelCompTosition fnow gnow) (tensorS flater glater)
 where

   -- Definition 5.3. Parallel compTosition.
   parallelCompTosition :: Arrow c
     => c (m,x) (p,z)
     -> c (n,y) (q,w)
     -> c ((m,n),(x,y)) ((p,q),(z,w))
   parallelCompTosition f g =
         associnv                   >>> id *** assoc
     >>> (id *** (sigma *** id))  >>> id *** associnv >>> assoc
     >>> (f *** g)                >>> associnv             >>> id *** assoc
     >>> (id *** (sigma *** id))  >>> id *** associnv >>> assoc

tensor :: Arrow c => Stream c x y -> Stream c x' y' -> Stream c (x,x') (y,y')
tensor f g = lact lunitinv (tensorS f g)
```

```
lact :: (Arrow c) => c n m -> StreamWithMemory c m x y -> StreamWithMemory c n x y
lact f (StreamWithMemory now later) = StreamWithMemory ((f *** id) >>> now) later


fbkS :: (Arrow c) =>
  StreamWithMemory c m (s,x) (s,y) ->
  StreamWithMemory c (m, s) x y
fbkS (StreamWithMemory now later) =
  StreamWithMemory (nowFeedback now) (fbkS later)
 where

   -- Definition 5.7. Feedback operation.
   nowFeedback :: (Arrow c) => c (m,(s,x)) (n,(t,y)) -> c ((m,s),x) ((n,t),y)
   nowFeedback f = associnv >>> f >>> assoc

fbk :: (Arrow c) => Stream c (s,x) (s,y) -> Stream c x y
fbk t = lact (arr (\() -> ((),undefined))) (fbkS t)

idS :: (Arrow c) => Stream c x x
idS = StreamWithMemory (id) idS


lift :: (Arrow c) => c x y -> Stream c x y
lift f = StreamWithMemory (id *** f) (lift f)

liftarr :: (Arrow c) => (x -> y) -> Stream c x y
liftarr s = lift $ arr s

instance (Arrow c) => Category (Stream c) where
  id = idS
  (.) f g = comp g f

instance (Arrow c) => Arrow (Stream c) where
  arr = liftarr
  (***) = tensor

instance (Arrow c) => ArrowLoop (Stream c) where
  loop f = fbk $ sigma >>> f >>> sigma


delay :: (Arrow c) => Stream c x y -> Stream c x y
delay f = StreamWithMemory (id *** undefined)  f




------------
-- ARROWS --
------------
assoc :: Arrow c => c (x,(y,z)) ((x,y),z)
assoc = arr $ \(x,(y,z)) -> ((x,y),z)
assocS :: Arrow c => Stream c (x,(y,z)) ((x,y),z)
```

```haskell
assocS = lift assoc

associnv :: Arrow c => c ((x,y),z) (x,(y,z))
associnv = arr $ \((x,y),z) -> (x,(y,z))
associnvS :: Arrow c => Stream c ((x,y),z) (x,(y,z))
associnvS = lift $ associnv

lunit :: Arrow c => c ((),a) a
lunit = arr $ \((),a) -> a
lunitS :: Arrow c => Stream c ((),a) a
lunitS = lift $ lunit

lunitinv :: Arrow c => c a ((),a)
lunitinv = arr $ \a -> ((),a)
lunitinvS :: Arrow c => Stream c a ((),a)
lunitinvS = lift $ lunitinv

runit :: Arrow c => c (a,()) a
runit = arr $ \(a,()) -> a
runitS :: Arrow c => Stream c (a,()) a
runitS = lift $ runit

runitinv :: Arrow c => c a (a,())
runitinv = arr $ \a -> (a,())
runitinvS :: Arrow c => Stream c a (a,())
runitinvS = lift $ runitinv


sigma :: Arrow c => c (x,y) (y,x)
sigma = arr $ \(x,y) -> (y,x)

sigmaS :: Arrow c => Stream c (x,y) (y,x)
sigmaS = lift $ sigma


----------------
-- GENERATORS --
----------------
fby :: (Monad t) => Stream (Kleisli t) (a , a) a
fby = StreamWithMemory (Kleisli $ \((),(x,y)) -> pure ((),x)) (lift (arr snd))

copy :: (Monad t) => Stream (Kleisli t) a (a,a)
copy = lift (Kleisli $ \a -> pure (a,a))


k0,k1,k2 :: (Arrow c) => Stream c () Int
k0 = lift $ arr (\() -> 0)
k1 = lift $ arr (\() -> 1)
k2 = lift $ arr (\() -> 2)

plus :: (Arrow c) => Stream c (Int,Int) Int
plus = lift $ arr (\(a,b) -> a + b)

wait :: (Arrow c) => Stream c a a
wait = fbk sigmaS

------------
-- SYSTEM --
------------

class (Monad m) => IOMonad m where unsafeRun :: m a -> m a
instance IOMonad IO where unsafeRun = unsafeInterleaveIO
instance IOMonad Identity where unsafeRun = id


runUnsafeWithMemory :: (IOMonad t) => m -> StreamWithMemory (Kleisli t) m a b -> [a] -> t [b]
runUnsafeWithMemory m (StreamWithMemory (Kleisli now) later) (a:as) = do
  (n , b)<- now (m , a)
  l <- unsafeRun $ runUnsafeWithMemory n later as
  pure (b : l)

runUnsafe :: (IOMonad t) => Stream (Kleisli t) a b -> [a] -> t [b]
runUnsafe = runUnsafeWithMemory ()

run :: (IOMonad t) => Stream (Kleisli t) () a -> t [a]
run s = runUnsafe s (repeat ())

------------------------------------------

main :: IO ()
main = return ()
```

3. Promonads and String Diagrams for Effectful Categories

Mario Román
Applied Category Theory (ACT, 2022)

Abstract: Premonoidal and Freyd categories are both generalized by non-cartesian Freyd categories: effectful categories. We construct string diagrams for effectful categories in terms of the string diagrams for a monoidal category with a freely added object. We show that effectful categories are pseudomonoids in a monoidal bicategory of promonads with a suitable tensor product.

Declaration: *Hereby I declare that my contribution to this manuscript was to: propose the research area, find the main theorem and proof, and write the whole article. My supervisor, Pawel Sobocinski, provided valuable feedback.*

Promonads and String Diagrams for Effectful Categories

Mario Román
Tallinn University of Technology
mroman@ttu.ee

Premonoidal and Freyd categories are both generalized by non-cartesian Freyd categories: effectful categories. We construct string diagrams for effectful categories in terms of the string diagrams for a monoidal category with a freely added object. We show that effectful categories are pseudomonoids in a monoidal bicategory of promonads with a suitable tensor product.

1 Introduction

Category theory has two sucessful applications that are rarely combined: monoidal string diagrams [23] and functional programming semantics [28]. We use string diagrams to talk about quantum transformations [1], relational queries [6], and even computability [31]; at the same time, proof nets and the geometry of interaction [13, 5] have been widely applied in computer science [2, 18]. On the other hand, we traditionally use monads and comonads, Kleisli categories and premonoidal categories to explain effectful functional programming [19, 20, 28, 34, 42]. Even if we traditionally employ Freyd categories with a cartesian base [32], we can also consider non-cartesian Freyd categories [40], which we call *effectful categories*.

Contributions. These applications are well-known. However, some foundational results in the intersection between string diagrams, premonoidal categories and effectful categories are missing in the literature. This manuscript contributes two such results.

- We introduce string diagrams for effectful categories. Jeffrey [22] was the first to preformally employ string diagrams of premonoidal categories. His technique consists in introducing an extra wire – which we call the *runtime* – that prevents some morphisms from interchanging. We promote this preformal technique into a result about the construction of free premonoidal, Freyd and effectful categories: the free premonoidal category can be constructed in terms of the free monoidal category with an extra wire.

  Our slogan, which constitutes the statement of Theorem 2.14, is

  "Premonoidal categories are Monoidal categories with a Runtime."

- We prove that effectful categories are promonad pseudomonoids. Promonads are the profunctorial counterpart of monads; they are used to encode effects in functional programming (where they are given extra properties and called *arrows* [19]). We claim that, in the same way that monoidal categories are pseudomonoids in the bicategory of categories [9], premonoidal effectful categories are pseudomonoids in a monoidal bicategory of promonads. This result justifies the role of effectful categories as a foundational object.

1.1 Synopsis

Sections 2.1 and 2.2 contain mostly preliminary material on premonoidal, Freyd and effectful categories. Our first original contribution is in Section 2.3; we prove that premonoidal categories are monoidal categories with runtime (Theorem 2.14). Section 3 makes explicit the well-known theory of profunctors, promonads and identity-on-objects functors. In Section 4, we introduce the pure tensor of promonads. We use it in Section 5 to prove our second main contribution (Theorem 5.3).

2 Premonoidal and Effectful Categories

2.1 Premonoidal categories

Premonoidal categories are monoidal categories without the *interchange law*, $(f \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes g) \neq (\mathrm{id} \otimes g) ⨟ (f \otimes \mathrm{id})$. This means that we cannot tensor any two arbitrary morphisms, $(f \otimes g)$, without explicitly stating which one is to be composed first, $(f \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes g)$ or $(\mathrm{id} \otimes g) ⨟ (f \otimes \mathrm{id})$, and the two compositions are not equivalent (Figure 1).

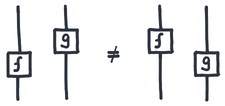


Figure 1: The interchange law does not hold in a premonoidal category.

In technical terms, the tensor of a premonoidal category $(\otimes) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$ is not a functor, but only what is called a *sesquifunctor*: independently functorial on each variable. Tensoring with any identity is itself a functor $(\bullet \otimes \mathrm{id}) \colon \mathbb{C} \to \mathbb{C}$, but there is no functor $(\bullet \otimes \bullet) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$.

A good motivation for dropping the interchange law can be found when describing transformations that affect some global state. These effectful processes should not interchange in general, because the order in which we modify the global state is meaningful. For instance, in the Kleisli category of the *writer monad*, $(\Sigma^* \times \bullet) \colon \mathsf{Set} \to \mathsf{Set}$ for some alphabet $\Sigma \in \mathsf{Set}$, we can consider the function $\mathsf{print} \colon \Sigma^* \to \Sigma^* \times 1$. The order in which we "print" does matter (Figure 2).

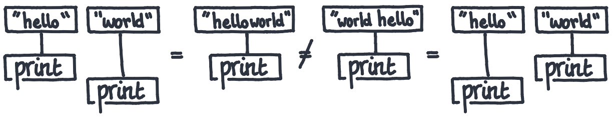


Figure 2: Writing does not interchange.

Not surprisingly, the paradigmatic examples of premonoidal categories are the Kleisli categories of Set-based monads $T \colon \mathsf{Set} \to \mathsf{Set}$ (more generally, of strong monads), which fail to be monoidal unless the monad itself is commutative [15, 33, 34, 16]. Intuitively, the morphisms are "effectful", and these effects do not always commute.

However, we may still want to allow some morphisms to interchange. For instance, apart from asking the same associators and unitors of monoidal categories to exist, we ask them to be *central*: that means that they interchange with any other morphism. This notion of centrality forces us to write the definition of premonoidal category in two different steps: first, we introduce the minimal setting in which centrality can be considered (*binoidal* categories [34]) and then we use that setting to bootstrap the full definition of premonoidal category with central coherence morphisms.

Definition 2.1 (Binoidal category). A *binoidal category* is a category $\mathbb{C}$ endowed with an object $I \in \mathbb{C}$ and an object $A \otimes B$ for each $A \in \mathbb{C}$ and $B \in \mathbb{C}$. There are functors $(A \otimes \bullet) \colon \mathbb{C} \to \mathbb{C}$, and $(\bullet \otimes B) \colon \mathbb{C} \to \mathbb{C}$ that coincide on $(A \otimes B)$, even if $(\bullet \otimes \bullet)$ is not itself a functor.

Again, this means that we can tensor with identities (whiskering), functorially; but we cannot tensor two arbitrary morphisms: the interchange law stops being true in general. The *centre*, $\mathscr{Z}(\mathbb{C})$, is the wide subcategory of morphisms that do satisfy the interchange law with any other morphism. That is, $f \colon A \to B$ is *central* if, for each $g \colon A' \to B'$,

$$(f \otimes \mathrm{id}_{A'}) ⨾ (\mathrm{id}_B \otimes g) = (\mathrm{id}_A \otimes g) ⨾ (f \otimes \mathrm{id}_{B'}), \text{ and } (\mathrm{id}_{A'} \otimes f) ⨾ (g \otimes \mathrm{id}_B) = (g \otimes \mathrm{id}_A) ⨾ (\mathrm{id}_{B'} \otimes f).$$

Definition 2.2. A *premonoidal category* is a binoidal category $(\mathbb{C}, \otimes, I)$ together with the following coherence isomorphisms $\alpha_{A,B,C} \colon A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, $\rho_A \colon A \otimes I \to A$ and $\lambda_A \colon I \otimes A \to A$ which are central, natural *separately at each given component*, and satisfy the pentagon and triangle equations.

A premonoidal category is *strict* when these coherence morphisms are identities. A premonoidal category is moreover *symmetric* when it is endowed with a coherence isomorphism $\sigma_{A,B} \colon A \otimes B \to B \otimes A$ that is central and natural at each given component, and satisfies the symmetry condition and hexagon equations.

Remark 2.3. The coherence theorem of monoidal categories still holds for premonoidal categories: every premonoidal is equivalent to a strict one. We will construct the free strict premonoidal category using string diagrams. However, the usual string diagrams for monoidal categories need to be restricted: in premonoidal categories, we cannot consider two morphisms in parallel unless any of the two is *central*.

2.2 Effectful and Freyd categories

Premonoidal categories immediately present a problem: what are the strong premonoidal functors? If we want them to compose, they should preserve centrality of the coherence morphisms (so that the central coherence morphisms of $F ⨾ G$ are these of F after applying G), but naively asking them to preserve all central morphisms rules out important examples [40]. The solution is to explicitly choose some central morphisms that represent "pure" computations. These do not need to form the whole centre: it could be that some morphisms considered *effectful* just "happen" to fall in the centre of the category, while we do not ask our functors to preserve them. This is the well-studied notion of a *non-cartesian Freyd category*, which we shorten to *effectful monoidal category* or *effectful category*.[1]

Effectful categories are premonoidal categories endowed with a chosen family of central morphisms. These central morphisms are called pure morphisms, constrasting with the general, non-central, morphisms that fall outside this family, which we call effectful.

Definition 2.4. An *effectful category* is an identity-on-objects functor $\mathbb{V} \to \mathbb{C}$ from a monoidal category $\mathbb{V}$ (the pure morphisms, or "values") to a premonoidal category $\mathbb{C}$ (the effectful morphisms, or "computations"), that strictly preserves all of the premonoidal structure and whose image is central. It is *strict* when both are. A *Freyd category* [24] is an effectful category where the pure morphisms form a cartesian monoidal category.

Effectful categories solve the problem of defining premonoidal functors: a functor between effectful categories needs to preserve only the pure morphisms. We are not losing expressivity: premonoidal categories are effectful with their centre, $\mathscr{Z}(\mathbb{C}) \to \mathbb{C}$. From now on, we study effectful categories.

[1]The name "Freyd category" sometimes assumes cartesianity of the pure morphisms, but it is also used for the general case. Choosing to call "effectful categories" to the general case and reserving the name "Freyd categories" for the cartesian ones avoids this clash of nomenclature. There exists also the more fine-grained notion of "Cartesian effect category" [12], which generalizes Freyd categories and may further justify calling "effectful category" to the general case.

Definition 2.5 (Effectful functor)**.** Let $\mathbb{V} \to \mathbb{C}$ and $\mathbb{W} \to \mathbb{D}$ be effectful categories. An *effectful functor* is a quadruple $(F, F_0, \varepsilon, \mu)$ consisting of a functor $F \colon \mathbb{C} \to \mathbb{D}$ and a functor $F_0 \colon \mathbb{V} \to \mathbb{W}$ making the square commute, and two natural and pure isomorphisms $\varepsilon \colon J \cong F(I)$ and $\mu \colon F(A \otimes B) \cong F(A) \otimes F(B)$ such that they make F_0 a monoidal functor. It is *strict* if these are identities.

When drawing string diagrams in an effectful category, we shall use two different colours to declare if we are depicting either a value or a computation (Figure 3).

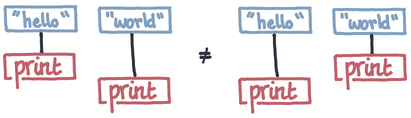


Figure 3: "Hello world" is not "world hello".

Here, the values "hello" and "world" satisfy the interchange law as in an ordinary monoidal category. However, the effectful computation "print" does not need to satisfy the interchange law. String diagrams like these can be found in the work of Alan Jeffrey [22]. Jeffrey presents a clever mechanism to graphically depict the failure of interchange: all effectful morphisms need to have a control wire as an input and output. This control wire needs to be passed around to all the computations in order, and it prevents them from interchanging.

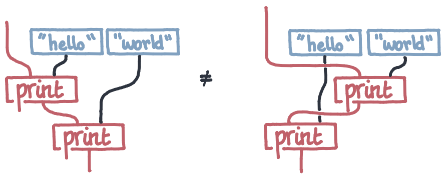


Figure 4: An extra wire prevents interchange.

A common interpretation of monoidal categories is as theories of resources. We can interpret premonoidal categories as monoidal categories with an extra resource – the "runtime" – that needs to be passed to all computations. The next section promotes Jeffrey's observation into a theorem.

2.3 Premonoidals are monoidals with runtime

String diagrams rely on the fact that the morphisms of the monoidal category freely generated over a polygraph of generators are string diagrams on these generators, quotiented by topological deformations [23]. We justify string diagrams for premonoidal categories by proving that the freely generated effectful category over a pair of polygraphs (for pure and effectful generators, respectively) can be constructed as the freely generated monoidal category over a particular polygraph that includes an extra wire.

Definition 2.6. A *polygraph* $\mathcal{G}$ (analogue of a *multigraph* [38]) is given by a set of objects, $\mathcal{G}_{\text{obj}}$, and a set of arrows $\mathcal{G}(A_0,\dots,A_n;B_0,\dots,B_m)$ for any two sequences of objects $A_0,\dots,A_n$ and $B_0,\dots,B_m$. A morphism of polygraphs $f \colon \mathcal{G} \to \mathcal{H}$ is a function between their object sets, $f_{\text{obj}} \colon \mathcal{G}_{\text{obj}} \to \mathcal{H}_{\text{obj}}$, and a function between their corresponding morphism sets,

$$f_{A_0,\dots,A_n;B_0,\dots,B_n} \colon \mathcal{G}(A_0,\dots,A_n;B_0,\dots,B_m) \to \mathcal{H}(f_{\text{obj}}(A_0),\dots,f_{\text{obj}}(A_n);f_{\text{obj}}(B_0),\dots,f_{\text{obj}}(B_m)).$$

A *polygraph couple* is a pair of polygraphs $(\mathcal{V},\mathcal{G})$ sharing the same objects, $\mathcal{V}_{\text{obj}} = \mathcal{G}_{\text{obj}}$. A morphism of polygraph couples $(u,f)\colon (\mathcal{V},\mathcal{G}) \to (\mathcal{W},\mathcal{H})$ is a pair of morphisms of polygraphs, $u\colon \mathcal{V} \to \mathcal{W}$ and $f\colon \mathcal{G} \to \mathcal{H}$, such that they coincide on objects, $f_{\text{obj}} = u_{\text{obj}}$.

Remark 2.7. There exists an adjunction between polygraphs and strict monoidal categories. Any monoidal category $\mathbb{C}$ can be seen as a polygraph $\mathcal{U}_{\mathbb{C}}$ where the edges $\mathcal{U}_{\mathbb{C}}(A_0,\dots,A_n;B_0,\dots,B_m)$ are the morphisms $\mathbb{C}(A_0 \otimes \dots \otimes A_n, B_0 \otimes \dots \otimes B_m)$, and we forget about composition and tensoring. Given a polygraph $\mathcal{G}$, the free strict monoidal category $\text{Mon}(\mathcal{G})$ is the strict monoidal category that has as morphisms the string diagrams over the generators of the polygraph.

We will construct a similar adjunction between polygraph couples and effectful categories. Let us start by formally adding the runtime to a free monoidal category.

Definition 2.8 (Runtime monoidal category)**.** Let $(\mathcal{V},\mathcal{G})$ be a polygraph couple. Its *runtime monoidal category*, $\text{Mon}_{\text{Run}}(\mathcal{V},\mathcal{G})$, is the monoidal category freely generated from adding an extra object – the runtime, R – to the input and output of every effectful generator in $\mathcal{G}$ (but not to those in $\mathcal{V}$), and letting that extra object be braided with respect to every other object of the category.

In other words, it is the monoidal category freely generated by the following polygraph, $\text{Run}(\mathcal{V},\mathcal{G})$, (Figure 5), assuming $A_0,\dots,A_n$ and $B_0,\dots,B_m$ are distinct from R

- $\text{Run}(\mathcal{V},\mathcal{G})_{\text{obj}} = \mathcal{G}_{\text{obj}} + \{R\} = \mathcal{V}_{\text{obj}} + \{R\}$,
- $\text{Run}(\mathcal{V},\mathcal{G})(R,A_0,\dots,A_n;R,B_0,\dots,B_n) = \mathcal{G}(A_0,\dots,A_n;B_0,\dots,B_n)$,
- $\text{Run}(\mathcal{V},\mathcal{G})(A_0,\dots,A_n;B_0,\dots,B_n) = \mathcal{V}(A_0,\dots,A_n;B_0,\dots,B_n)$,
- $\text{Run}(\mathcal{V},\mathcal{G})(R,A_0;A_0,R) = \text{Run}(\mathcal{V},\mathcal{G})(A_0,R;R,A_0) = \{\sigma\}$,

with $\text{Run}(\mathcal{V},\mathcal{G})$ empty in any other case, and quotiented by the braiding axioms for R (Figure 6).

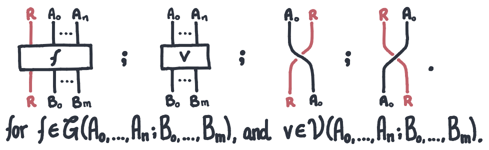


Figure 5: Generators for the runtime monoidal category.

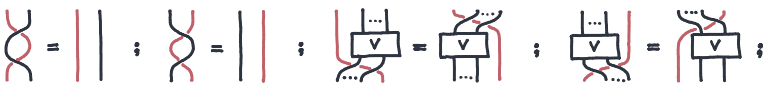


Figure 6: Axioms for the runtime monoidal category.

Somehow, we are asking the runtime R to be in the Drinfeld centre [11] of the monoidal category. The extra wire that R provides is only used to prevent interchange, and so it does not really matter where it is placed in the input and the output. We can choose to always place it on the left, for instance – and indeed we will be able to do so – but a better solution is to just consider objects "up to some runtime braidings". This is formalized by the notion of *braid clique*.

Definition 2.9 (Braid clique)**.** Given any list of objects $A_0,\dots,A_n$ in $\mathcal{V}_{\text{obj}} = \mathcal{G}_{\text{obj}}$, we construct a *clique* [41, 39] in the category $\text{Mon}_{\text{Run}}(\mathcal{V},\mathcal{G})$: we consider the objects, $A_0 \otimes \dots \otimes R_{(i)} \otimes \dots \otimes A_n$, created by inserting the runtime R in all of the possible $0 \leqslant i \leqslant n+1$ positions; and we consider the family of commuting isomorphisms constructed by braiding the runtime,

$$\sigma_{i,j} \colon A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n \to A_0 \otimes \ldots \otimes R_{(j)} \otimes \ldots \otimes A_n.$$

We call this the *braid clique*, $\mathrm{Braid}_R(A_0,\ldots,A_n)$, on that list.

Definition 2.10. A *braid clique morphism*, $f \colon \mathrm{Braid}_R(A_0,\ldots,A_n) \to \mathrm{Braid}_R(B_0,\ldots,B_m)$ is a family of morphisms in the runtime monoidal category, $\mathrm{Mon}_{\mathrm{Run}}(\mathscr{V},\mathscr{G})$, from each of the objects of first clique to each of the objects of the second clique,

$$f_{ik} \colon A_0 \otimes \ldots \otimes R_{(i)} \otimes \ldots \otimes A_n \to B_0 \otimes \ldots \otimes R_{(k)} \otimes \ldots \otimes B_m,$$

that moreover commutes with all braiding isomorphisms, $f_{ij} \fatsemi \sigma_{jk} = \sigma_{il} \fatsemi f$.

A braid clique morphism $f \colon \mathrm{Braid}_R(A_0,\ldots,A_n) \to \mathrm{Braid}_R(B_0,\ldots,B_m)$ is fully determined by *any* of its components, by pre/post-composing it with braidings. In particular, a braid clique morphism is always fully determined by its leftmost component $f_{00} \colon R \otimes A_0 \otimes \ldots \otimes A_n \to R \otimes B_0 \otimes \ldots \otimes B_m$.

Lemma 2.11. *Let $(\mathscr{V},\mathscr{G})$ be a polygraph couple. There exists a premonoidal category, $\mathrm{Eff}(\mathscr{V},\mathscr{G})$, that has objects the braid cliques, $\mathrm{Braid}_R(A_0,\ldots,A_n)$, in $\mathrm{Mon}_{\mathrm{Run}}(\mathscr{V},\mathscr{G})$, and as morphisms the braid clique morphisms between them. See Appendix.*

Lemma 2.12. *Let $(\mathscr{V},\mathscr{G})$ be a polygraph couple. There exists an identity-on-objects functor $\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G})$ that strictly preserves the premonoidal structure and whose image is central. See Appendix.*

Lemma 2.13. *Let $(\mathscr{V},\mathscr{G})$ be a polygraph couple and consider the effectful category determined by $\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G})$. Let $\mathbb{V} \to \mathbb{C}$ be a strict effectful category endowed with a polygraph couple morphism $F \colon (\mathscr{V},\mathscr{G}) \to \mathscr{U}(\mathbb{V},\mathbb{C})$. There exists a unique strict effectful functor from $(\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G}))$ to $(\mathbb{V} \to \mathbb{C})$ commuting with F as a polygraph couple morphism. See Appendix.*

Theorem 2.14 (Runtime as a resource)**.** *The free strict effectful category over a polygraph couple $(\mathscr{V},\mathscr{G})$ is $\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G})$. Its morphisms $A \to B$ are in bijection with the morphisms $R \otimes A \to R \otimes B$ of the runtime monoidal category,*

$$\mathrm{Eff}(\mathscr{V},\mathscr{G})(A,B) \cong \mathrm{Mon}_{\mathrm{Run}}(\mathscr{V},\mathscr{G})(R \otimes A, R \otimes B).$$

Proof. We must first show that $\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G})$ is an effectful category. The first step is to see that $\mathrm{Eff}(\mathscr{V},\mathscr{G})$ forms a premonoidal category (Lemma 2.11). We also know that $\mathrm{Mon}(\mathscr{V})$ is a monoidal category: in fact, a strict, freely generated one. There exists an identity on objects functor, $\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G})$, that strictly preserves the premonoidal structure and centrality (Lemma 2.12).

Let us now show that it is the free one over the polygraph couple $(\mathscr{V},\mathscr{G})$. Let $\mathbb{V} \to \mathbb{C}$ be an effectful category, with an polygraph couple map $F \colon (\mathscr{V},\mathscr{G}) \to \mathscr{U}(\mathbb{V},\mathbb{C})$. We can construct a unique effectful functor from $(\mathrm{Mon}(\mathscr{V}) \to \mathrm{Eff}(\mathscr{V},\mathscr{G}))$ to $(\mathbb{V} \to \mathbb{C})$ giving its universal property (Lemma 2.13). □

Corollary 2.15 (String diagrams for effectful categories)**.** *We can use string diagrams for effectful categories, quotiented under the same isotopy as for monoidal categories, provided that we do represent the runtime as an extra wire that needs to be the input and output of every effectful morphism.*

3 Profunctors and Promonads

We have elaborated on string diagrams for effectful categories. Let us now show that effectful categories are fundamental objects. The profunctorial counterpart of a monad is a promonad. Promonads have been widely used for functional programming semantics, although usually with an extra assumption of strength and under the name of "arrows" [17, 19, 20]. Promonads over a category endow it with some new, "effectful", morphisms; while the base morphisms of the category are called the "pure" morphisms. This terminology will coincide when regarding effectful categories as promonads.

In this section, we introduce profunctors and promonads. In the following sections, we show that effectful categories are to promonads what monoidal categories are to categories: they are the pseudomonoids of a suitably constructed monoidal bicategory of promonads. In order to obtain this result, we introduce the pure tensor of promonads in Section 4. The pure tensor of promonads combines the effects of two promonads over different categories into a single one. In some sense, it does so in the universal way that turns "purity" into "centrality" (Theorem 4.2).

3.1 Profunctors: an algebra of processes

Profunctors $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathsf{Set}$ [3, 7, 4] can be thought as indexing *families of processes* $P(A,B)$ by the types of an input channel A and an output channel B [10].

The category $\mathbb{A}$ has as morphisms the pure transformations $f\colon A' \to A$ that we can apply to the input of a process $p \in P(A,B)$ to obtain a new process, which we call $(f > p) \in P(A',B)$. Analogously, the category $\mathbb{B}$ has as morphisms the pure transformations $g\colon B \to B'$ that we can apply to the output of a process $p \in P(A,B)$ to obtain a new process, which we call $(p < g) \in P(A,B')$. The profunctor axioms encode the compositionality of these transformations.

Definition 3.1. A *profunctor* $(P,>,<)$ between two categories $\mathbb{A}$ and $\mathbb{B}$ is a family of sets $P(A,B)$ indexed by objects of $\mathbb{A}$ and $\mathbb{B}$, and endowed with jointly functorial left and right actions of the morphisms of $\mathbb{A}$ and $\mathbb{B}$, respectively. Explicitly, types of these actions are $(>)\colon \hom(A',A) \times P(A',B) \to P(A,B)$, and $(<)\colon \hom(B,B') \times P(A,B) \to P(A,B')$. They must satisfy

- compatibility, $(f > p) < g = f > (p < g)$,
- preserve identities, $\mathrm{id} > p = p$, and $p < \mathrm{id} = p$,
- and composition, $(p < f) < g = p < (f \mathbin{⨟} g)$ and $f > (g > p) = (f \mathbin{⨟} g) > p$.

More succintly, a *profunctor* $P\colon \mathbb{A} \nrightarrow \mathbb{B}$ is a functor $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathsf{Set}$. When presented as a family of sets with a pair of actions, profunctors are sometimes called *bimodules*.

A profunctor homomorphism $\alpha\colon P \to Q$ transforms processes of type $P(A,B)$ into processes of type $Q(A,B)$. The homomorphism affects only the effectful processes, and not the pure transformations we could apply in $\mathbb{A}$ and $\mathbb{B}$. This means that $\alpha(f > p) = f > \alpha(p)$ and that $\alpha(p < g) = \alpha(p) < g$.

Definition 3.2 (Profunctor homomorphism)**.** A *profunctor homomorphism* from the profunctor $P\colon \mathbb{A} \nrightarrow \mathbb{B}$ to the profunctor $Q\colon \mathbb{A} \nrightarrow \mathbb{B}$ is a family of functions $\alpha_{A,B}\colon P(A,B) \to Q(A,B)$ preserving the left and right actions, $\alpha(f > p < g) = f > \alpha(p) < g$. Equivalently, it is a natural transformation $\alpha\colon P \to Q$ between the two functors $\mathbb{A}^{op} \times \mathbb{B} \to \mathsf{Set}$.

How to compose two families of processes? Assume we have a process $p \in P(A,B_1)$ and a process $q \in Q(B_2,C)$. Moreover, assume we have a transformation $f\colon B_1 \to B_2$ translating from the output of the second to the input of the first. In this situation, we can plug together the processes: $p \in P(A,B_1)$ writes to an output of type B_1, which is translated by f to an input of type B_2, then used by $q \in Q(B_2,C)$.

There are two slightly different ways of describing this process, depending on whether we consider the translation to be part of the first or the second process. We could translate just after finishing the first process, $(p < f, q)$; or translate just before starting the second process, $(p, f > q)$.

These are two different pairs of processes, with different types. However, if we take the process interpretation seriously, it does not really matter when to apply the translation. These two descriptions represent the same process. They are *dinaturally equivalent* [10, 25].

Definition 3.3 (Dinatural equivalence)**.** Let $P\colon \mathbb{A} \nrightarrow \mathbb{B}$ and $Q\colon \mathbb{B} \nrightarrow \mathbb{C}$ be two profunctors. Consider the set of matching pairs of processes, with a given input A and output C,

$$R_{P,Q}(A,C) = \sum_{B \in \mathbb{B}} P(A,B) \times Q(B,C).$$

Dinatural equivalence $(\sim)$, on the set $R_{P,Q}(A,C)$ is the smallest equivalence relation satisfying $(p < g, q) \sim (p, g > q)$. The set of matching processes $R_{P,Q}(A,C)$ quotiented by dinaturality $(\sim)$ is written as $(P \diamond Q)(A,C)$. It is a particular form of colimit over the category $\mathbb{B}$, called a *coend*, usually denoted by an integral sign.

$$(P \diamond Q)(A,C) = R_{P,Q}(A,C)/(\sim) = \int^{B \in \mathbb{B}} P(A,B) \times Q(B,C).$$

Definition 3.4 (Profunctor composition)**.** The composition of two profunctors $P\colon \mathbb{A} \nrightarrow \mathbb{B}$ and $Q\colon \mathbb{B} \nrightarrow \mathbb{C}$ is the profunctor $(P \diamond Q)\colon \mathbb{A} \nrightarrow \mathbb{C}$ has as processes the matching pairs of processes in P and Q quotiented by dinaturality on $\mathbb{B}$,

$$(p, g < q) \sim (p > g, q).$$

Its actions are the left and right actions of p and q, respectively, $f > (p,q) < g = (f > p, q < g)$.

The identity profunctor $\mathbb{A}\colon \mathbb{A} \nrightarrow \mathbb{A}$ has as processes the morphisms of the category $\mathbb{A}$, it is given by the hom-sets. Its actions are pre and post-composition, $f > h < g = f \mathbin{⨾} h \mathbin{⨾} g$.

Profunctors are better understood as providing a double categorical structure to the category of categories. A double category $\mathbb{D}$ contains 0-cells (or "objects"), two different types of 1-cells (the "arrows" and the "proarrows"), and cells [37]. Arrows compose in an strictly associative and unital way, while proarrows come equipped with natural isomorphisms representing associativity and unitality. We employ the graphical calculus of double categories [29], with arrows going left to right and proarrows going top to bottom.

Definition 3.5. The double category of categories, **CAT**, has as objects the small categories $\mathbb{A}, \mathbb{B}, \dots$, as arrows the functors between them, $F\colon \mathbb{A} \to \mathbb{A}'$, as proarrows the profunctors between them, $P\colon \mathbb{A} \nrightarrow \mathbb{B}$, and as cells, the natural transformations, $\alpha_{A,B}\colon P(A,B) \to Q(FA, GB)$.

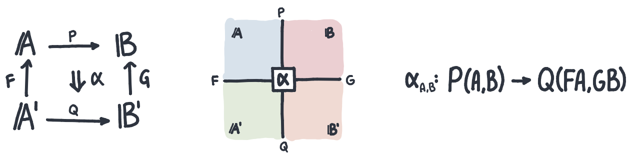

$$\begin{tikzcd} \mathbb{A} \arrow[r, "P"] & \mathbb{B} \\ \mathbb{A}' \arrow[u, "F"] \arrow[r, "Q"'] & \mathbb{B}' \arrow[u, "G"'] \end{tikzcd}$$
$\Downarrow \alpha$
P
A
B
F
$\boxed{\alpha}$
G
A'
B'
Q
$\alpha_{A,B}$: P(A,B) → Q(FA,GB)

Figure 7: Cell in the double category of categories.

Every functor has a companion and a conjoint profunctors: their representable and corepresentable profunctors [14]. This structure makes **CAT** into the paradigmatic example of a proarrow equipment (or *framed bicategory* [37]).

3.2 Promonads: new morphisms for an old category

Promonads are to profunctors what monads are to functors.[2] It may be then surprising to see that so little attention has been devoted to them, relative to their functorial counterparts. The main source of examples and focus of attention has been the semantics of programming languages [19, 30, 20]. Strong monads are commonly used to give categorical semantics of effectful programs [28], and the so-called *arrows* (or *strong promonads*) strictly generalize them.

Part of the reason behind the relative unimportance given to promonads elsewhere may stem from the fact that promonads over a category can be shown in an elementary way to be equivalent to identity-on-objects functors from that category [25]. The explicit proof is, however, difficult to find in the literature, and so we include it here (Theorem 3.9).

Under this interpretation, promonads are new morphisms for an old category. We can reinterpret the old morphisms into the new ones in a functorial way. The paradigmatic example is again that of Kleisli or cokleisli categories of strong monads and comonads. This structure is richer than it may sound, and we will explore it further during the rest of this text.

Definition 3.6 (Monoids and promonoids)**.** A *monoid* in a double category is an arrow $T\colon \mathbb{A} \to \mathbb{A}$ together with cells $m \in \mathrm{hom}(M \otimes M;1,1;M)$ and $e \in \mathrm{cell}(1;1,1;M)$, called multiplication and unit, satisfying unitality and associativity. A *promonoid* in a double category is a proarrow $M\colon \mathbb{A} \nrightarrow \mathbb{A}$ together with cells $m \in \mathrm{cell}(1;M \otimes M,M,1)$ and $e \in \mathrm{cell}(1;1,M;1)$, called promultiplication and prounit, satisfying unitality and associativity.

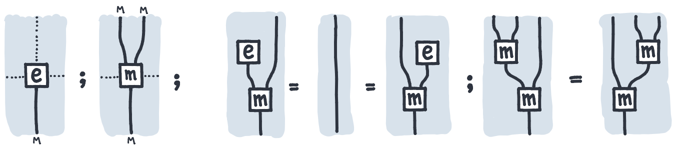


Figure 8: Data and axioms of a promonoid in a double category.

Dually, we can define *comonoids* and *procomonoids*.

A monad is a monoid in the category of categories, functors and profunctors **Cat**. In the same way, a promonad is a promonoid in **Cat**.

Definition 3.7. A *promonad* $(P,\star,^{\circ})$ over a category $\mathbb{C}$ is a profunctor $P\colon \mathbb{C} \nrightarrow \mathbb{C}$ together with natural transformations representing inclusion $(^{\circ})_{X,Y}\colon \mathbb{C}(X,Y) \to P(X,Y)$ and multiplication $(\star)_{X,Y}\colon P(X,Y) \times P(Y,Z) \to P(X,Z)$, and such that

i. the right action is premultiplication, $f^{\circ} \star p = f > p$;

ii. the left action is posmultiplication, $p \star f^{\circ} = p < f$;

iii. multiplication is dinatural, $p \star (f > q) = (p < f) \star q$;

iv. and multiplication is associative, $(p_1 \star p_2) \star p_3 = p_1 \star (p_2 \star p_3)$.

Equivalently, promonads are promonoids in the double category of categories, where the dinatural multiplication represents a transformation from the composition of the profunctor P with itself.

Lemma 3.8 (Kleisli category of a promonad)**.** *Every promonad* $(P,\star,^{\circ})$ *induces a category with the same objects as its base category, but with hom-sets given by* $P(\bullet,\bullet)$*, composition given by* $(\star)$ *and identities given by* (id°)*. This is called its* Kleisli category, $\mathrm{kleisli}(P)$*. Moreover, there exists an identity-on-objects functor* $\mathbb{C} \to \mathrm{kleisli}(P)$*, defined on morphisms by the unit of the promonad. See Appendix.*

[2]To quip, a promonad is just a monoid on the category of endoprofunctors.

The converse is also true: every category $\mathbb{C}$ with an identity-on-objects functor from some base category $\mathbb{V}$ arises as the Kleisli category of a promonad.

Theorem 3.9. *Promonads over a category $\mathbb{C}$ correspond to identity-on-objects functors from the category $\mathbb{C}$. Given any identity-on-objects functor $i\colon \mathbb{C} \to \mathbb{D}$ there exists a unique promonad over $\mathbb{C}$ having $\mathbb{D}$ as its Kleisli category: the promonad given by the profunctor* $\hom_{\mathbb{D}}(i(\bullet), i(\bullet))$. *See Appendix.*

3.3 Homomorphisms and transformations of promonads

We have characterized promonads as identity-on-objects functors. We now characterize the homomorphisms and transformations of promonads as suitable pairs of functors and natural transformations.

Definition 3.10 (Promonoid homomorphism)**.** Let $(\mathbb{A}, M, m, e)$ and $(\mathbb{B}, N, n, u)$ be promonoids in a double category. A promonoid homomorphism is an arrow $T\colon \mathbb{A} \to \mathbb{B}$ together with a cell $t \in \mathrm{cell}(F; M, N; F)$ that preserves the promonoid promultiplication and prounit.

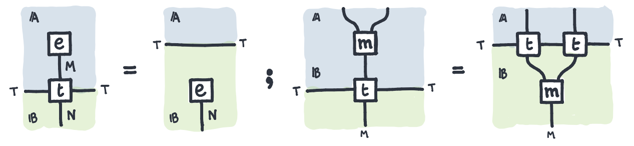


Figure 9: Axioms for a promonoid homomorphism.

Definition 3.11 (Promonad homomorphism)**.** Let $(\mathbb{A}, P, \star, ^\circ)$ and $(\mathbb{B}, Q, \star, ^\circ)$ be two promonads, possibly over two different categories. A promonad homomorphism (F_0, F) is a functor between the underlying categories $F_0\colon \mathbb{A} \to \mathbb{B}$ and a natural transformation $F_{X,Y}\colon P(X,Y) \to Q(FX, FY)$ preserving composition and inclusions. That is, $F(p_1 \star p_2) = F(p_1) \star F(p_2)$, and $F(f^\circ) = F_0(f)^\circ$.

Proposition 3.12. *A promonad homomorphism between two promonads understood as identity-on-objects functors, $\mathbb{V} \to \mathbb{C}$ and $\mathbb{W} \to \mathbb{D}$, is equivalently a pair of functors (F_0, F) that commute strictly with the two identity-on-objects functors on objects $F_0(X) = F(X)$ and morphisms $F_0(f)^\circ = F(f^\circ)$. See Appendix.*

Definition 3.13 (Promonoid modification)**.** Let $(\mathbb{A}, M, m, e)$ and $(\mathbb{B}, N, n, u)$ be promonoids in a double category, and let $t \in \mathrm{cell}(F; M, N; F)$ and $r \in \mathrm{cell}(G; M, N; G)$ be promonoid homomorphisms. A *promonoid modification* is a cell $\alpha \in \mathrm{cell}(F; 1, 1; G)$ such that its precomposition with t is its postcomposition with r.

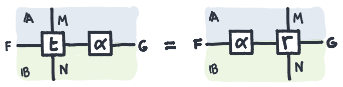


Figure 10: Axiom for a promonoid transformation.

Definition 3.14. A *promonad modification* between two promonad homomorphisms (F_0, F) and (G_0, G) between the same promonads $(\mathbb{A}, P, \star, ^\circ)$ and $(\mathbb{B}, Q, \star, ^\circ)$ is a natural transformation $\alpha_X\colon F_0(X) \to G_0(X)$ such that $\alpha_X > G(p) = F(p) < \alpha_Y$ for each $p \in P(X,Y)$.

Proposition 3.15. *A promonad modification between two promonad homomorphisms understood as commutative squares of identity-on-objects functors $F_0(f)^\circ = F(f^\circ)$ and $G_0(f)^\circ = G(f^\circ)$ is a natural transformation $\alpha\colon F_0 \Rightarrow G_0$ that can be lifted via the identity-on-objects functor to a natural transformation $\alpha^\circ\colon F \Rightarrow G$. In other words, a pure natural transformation.*

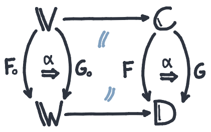


Figure 11: Promonad modifications are cylinder transformations.

Summarizing this section, we have shown a correspondence between promonads, their homomorphisms and modifications, and identity-on-objects functors, squares and cylinder transformations of squares. The double category structure allows us to talk about homomorphisms and modifications, which would be more difficult to address in a bicategory structure.

Promonad	Identity-on-objects functor	Theorem 3.9
Promonad homomorphism	Commuting square	Proposition 3.12
Promonad modification	Cylinder transformation	Proposition 3.15

4 Pure Tensor of Promonads

This section introduces the *pure tensor of promonads*. The pure tensor of promonads combines the effects of two promonads, possibly over different categories, into the effects of a single promonad over the product category. Effects do not generally interchange. However, this does not mean that no morphisms should interchange in the pure tensor of promonads: in our interpretation of a promonad $\mathbb{V} \to \mathbb{C}$, the morphisms coming from the inclusion are *pure*, they produce no effects; pure morphisms with no effects should always interchange with effectful morphisms, even if effectful morphisms do not interchange among themselves.

A practical way to encode and to remember all of the these restrictions is to use monoidal string diagrams. This is another application of the idea of runtime: we introduce an extra wire so that all the rules of interchange become ordinary interchange laws in a monoidal category. That is, we insist again that effectful morphisms are just pure morphisms using a shared resource – the runtime. When we compute the pure tensor of two promonads, the runtime needs to be shared between the impure morphisms of both promonads.

4.1 Pure tensor, via runtime

Definition 4.1 (Pure tensor)**.** Let $\mathbb{C} \colon \mathbb{V} \nrightarrow \mathbb{V}$ and $\mathbb{D} \colon \mathbb{W} \nrightarrow \mathbb{W}$ be two promonads. Their *pure tensor*, $\mathbb{C} * \mathbb{D} \colon \mathbb{V} \times \mathbb{W} \to \mathbb{V} \times \mathbb{W}$, is a promonad over $\mathbb{V} \times \mathbb{W}$ where elements of $\mathbb{C} * \mathbb{D}(X,Y;X',Y')$, the morphisms $X \otimes R \otimes Y \to X' \otimes R \otimes Y'$ in the freely presented monoidal category generated by the elements of Figure 12 and quotiented by the axioms of Figure 13.

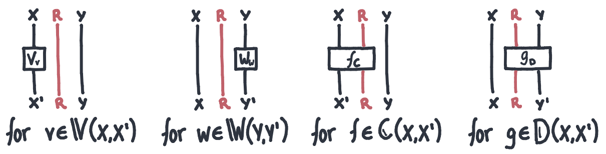

X R Y
X' R Y
for $v \in \mathbb{V}(X,X')$
X R Y
X R Y'
for $w \in \mathbb{W}(Y,Y')$
X R Y
X' R Y
for $f \in \mathbb{C}(X,X')$
X R Y
X R Y'
for $g \in \mathbb{D}(X,X')$

Figure 12: Generators for the elements of the pure tensor of promonads.

Multiplication is defined by composition in the monoidal category, and the unit is defined by the inclusion of pairs, as depicted in Figure 14.

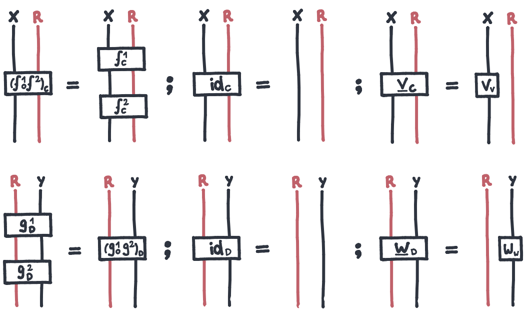


Figure 13: Axioms for the elements of the pure tensor of promonads.

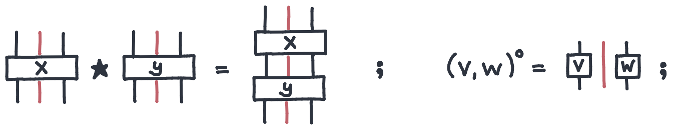


Figure 14: The pure tensor promonad.

In other words, the elements of the pure tensor are the morphisms the category presented by the graph that has as objects the pairs of objects (X,Y) with $X \in \mathbb{V}_{\text{obj}}$ and $Y \in \mathbb{W}_{\text{obj}}$, formally written as $X \otimes R \otimes Y$; and the morphisms generated by

- an edge $f_{\mathbb{C}} \colon X \otimes R \otimes Y \to X' \otimes R \otimes Y$ for each arrow $f \in \mathbb{C}(X,X')$ and each object $Y \in \mathbb{W}$;
- an edge $g_{\mathbb{D}} \colon X \otimes R \otimes Y \to X \otimes R \otimes Y'$ for each arrow $g \in \mathbb{D}(Y,Y')$ and each object $X \in \mathbb{V}$;
- an edge $v_{\mathbb{V}} \colon X \otimes R \otimes Y \to X' \otimes R \otimes Y$ for each arrow $v \in \mathbb{V}(X,X')$ and each object $Y \in \mathbb{W}$;
- and an edge $w_{\mathbb{W}} \colon X \otimes R \otimes Y \to X \otimes R \otimes Y'$ for each arrow $w \in \mathbb{W}(Y,Y')$ and each object $X \in \mathbb{V}$;

quotiented by centrality of pure morphisms: $f_{\mathbb{C}} \fatsemi w_{\mathbb{W}} = w_{\mathbb{W}} \fatsemi f_{\mathbb{C}}$ and $g_{\mathbb{D}} \fatsemi v_{\mathbb{V}} = v_{\mathbb{V}} \fatsemi g_{\mathbb{D}}$; by compositions and identities of one promonad: $f_{\mathbb{C}} \fatsemi f'_{\mathbb{C}} = (f \star f')_{\mathbb{C}}$ and $\mathrm{id}_{\mathbb{C}} = \mathrm{id}$; by compositions and identities of the other promonad: $g_{\mathbb{D}} \fatsemi g'_{\mathbb{D}} = (g \star g')_{\mathbb{D}}$ and $\mathrm{id}_{\mathbb{D}} = \mathrm{id}$; and by the coincidence of pure morphisms and their effectful representatives: $v_{\mathbb{V}} = v^{\circ}_{\mathbb{C}}$ and $w_{\mathbb{W}} = w^{\circ}_{\mathbb{D}}$.

Crucially in this definition, $f_{\mathbb{C}}$ and $g_{\mathbb{D}}$ do not interchange: they are sharing the runtime, and that prevents the application of the interchange law. The pure tensor of promonads, $\mathbb{C} * \mathbb{D}$, takes its name from the fact that, if we interpret the promonads $\mathbb{V} \to \mathbb{C}$ and $\mathbb{W} \to \mathbb{D}$ as declaring the morphisms in $\mathbb{V}$ and $\mathbb{W}$ as pure, then the pure morphisms of the composition interchange with all effectful morphisms. The spirit is similar to the *free product of groups with commuting subgroups* [26].

4.2 Universal property of the pure tensor

There are multiple canonical ways in which one could combine the effects of two promonads, $\mathbb{C} \colon \mathbb{V} \nrightarrow \mathbb{V}$ and $\mathbb{D} \colon \mathbb{W} \nrightarrow \mathbb{W}$, into a single promonad, such as taking the product of both, $\mathbb{C} \times \mathbb{D} \colon \mathbb{V} \times \mathbb{W} \nrightarrow \mathbb{V} \times \mathbb{W}$. Let us show that the pure tensor has a universal property: it is the universal one in which we can include impure morphisms from each promonads, interchanging with pure morphisms from the other promonad, so that purity is preserved.

Theorem 4.2. *Let $\mathbb{C}\colon \mathbb{V} \nrightarrow \mathbb{V}$ and $\mathbb{D}\colon \mathbb{W} \nrightarrow \mathbb{W}$ be two promonads and let $\mathbb{C} * \mathbb{D}\colon \mathbb{V}\times\mathbb{W} \to \mathbb{V}\times\mathbb{W}$ be their pure tensor. There exist a pair of promonad homomorphisms $L\colon \mathbb{C}\times\mathbb{W} \to \mathbb{C}*\mathbb{D}$ and $R\colon \mathbb{V}\times\mathbb{D} \to \mathbb{C}*\mathbb{D}$. These are universal in the sense that, for every pair of promonad homomorphisms, $A\colon \mathbb{C}\times\mathbb{W} \to \mathbb{E}$ and $B\colon \mathbb{V}\times\mathbb{D} \to \mathbb{E}$, there exists a unique promonad homomorphism $(A \vee B)\colon \mathbb{C}*\mathbb{D} \to \mathbb{E}$ that commutes strictly with them, $(A\vee B) \mathbin{⨟} L = A$ and $(A \vee B) \mathbin{⨟} R = B$. See Appendix.*

5 Effectful Categories are Pseudomonoids

We will now use the pure tensor of promonads to justify effectful categories as the promonadic counterpart of monoidal categories: effectful categories are pseudomonoids in the monoidal bicategory of promonads with the pure tensor. Pseudomonoids [9, 43] are the categorification of monoids. They are still formed by a 0-cell representing the carrier of the monoid and a pair of 1-cells representing multiplication and units. However, we weaken the requirement for associativity and unitality to the existence of invertible 2-cells, called the *associator* and *unitor*.

In the same way that monoids live in monoidal categories, pseudomonoids live in monoidal bicategories. A monoidal bicategory $\mathbb{A}$ is a bicategory in which we can tensor objects with a pseudofunctor $(\boxtimes)\colon \mathbb{A}\times\mathbb{A} \to \mathbb{A}$ and we have a tensor unit $I\colon 1 \to \mathbb{A}$, these are associative and unital up to equivalence, and satisfy certain coherence equations up to invertible modification [36].

5.1 Pseudomonoids

Definition 5.1. In a monoidal bicategory, a *pseudomonoid* over a 0-cell M is a pair of 1-cells, $M \boxtimes M \to M$ and $I \to M$, together with the following triple of invertible 2-cells representing associativity and unitality (Figure 15), and satisfying the pentagon and triangle equations (see Appendix).A homomorphism of pseudomonoids is given by a 1-cell between their underlying 0-cells and the following invertible 2-cells, representing preservation of the multiplication and the unit (Figure 15), and satisfying compatibility with associativity and unitality (see Appendix).

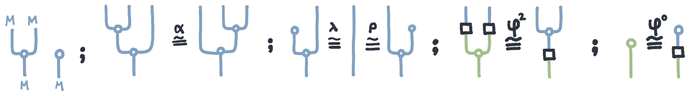


Figure 15: Data for a pseudomonoid and pseudomonoid homomorphism.

A pseudomonoid is *strict* when the associators and unitors are identity cells. Note that, in strict 2-categories (sometimes called 2-categories, in contrast to bicategories), this is the same as a monoid in the monoidal category that we obtain by ignoring the 2-cells.

Remark 5.2. A pseudomonoid in the monoidal bicategory of categories with the cartesian product of categories, $(\mathbf{Cat}, \times)$ is a monoidal category. A strict pseudomonoid in the same monoidal bicategory is a strict monoidal category.

A strict pseudomonoid in the monoidal bicategory of categories with the funny tensor product of categories $(\mathbf{Cat}, \square)$ is a strict premonoidal category. However, it is not immediately clear how to recover premonoidal categories as pseudomonoids. A naive attempt will fail: $(\mathbf{Cat}, \square)$ is usually made into a monoidal bicategory with non-necessarily-natural transformations, but we do want our coherence morphisms to be natural, so we must ask at least naturality. This will not be enough: taking natural transformations as 2-cells will give us premonoidal categories where the associators and unitors do not need to be *central*. Centrality is what requires a more careful approach.

5.2 Effectful categories are promonad pseudomonoids

Promonads form a monoidal category with the pure tensor product and moreover a strict monoidal bicategory with promonad modifications. Effectful categories are the pseudomonoids in this category.

Theorem 5.3. *An effectful category (or monoidal Freyd category) is a pseudomonoid on the monoidal 2-category of promonads with promonad homomorphism, promonad transformations and the pure tensor of promonads. A pseudomonoid homomorphism between effectful categories is an effectful functor.*

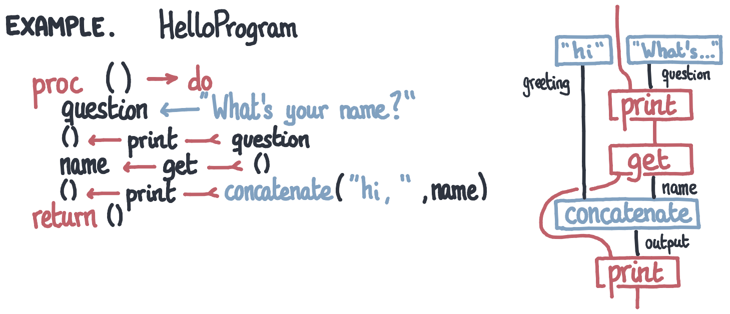
As a consequence, preomonoidal categories with their centre *are pseudomonoids. See Appendix.*

6 Conclusions

Premonoidal categories are monoidal categories with runtime, and we can stil use monoidal string diagrams and unrestricted topological deformations to reason about them. Instead of dealing directly with premonoidal categories, we employ the better behaved notion of non-cartesian Freyd categories, effectful categories. There exists a more fine-grained notion of "Cartesian effect category" [12], which generalizes Freyd categories and justifies calling "effectful category" to the general case.

Promonads have been arguably under-appreciated, possibly because of their characterization as "just" identity-on-objects functors. However, speaking of promonads as the proarrow counterpart of monads makes many aspects of the theory of monads clearer: every monad and every comonad induce a promonad (their Kleisli category) via the proarrow equipment, monad morphisms lift to promonad morphisms, distributive laws of monads induce a way of composing morphisms from different kleisli categories [8]. Justifying effectful categories in terms of promonads highlights their importance as the monadic counterpart of monoidal categories.

Ultimately, this is a first step towards our more ambitious project of presenting the categorical structure of programming languages in a purely diagrammatic way, revisiting Alan Jeffrey's work [22, 21, 35]. The internal language of premonoidal categories and effectful categories is given by the *arrow do-notation* [30]; at the same time, we have shown that it is given by suitable string diagrams. This correspondence allows us to translate between programs and string diagrams (Figure 16).



```
EXAMPLE.   HelloProgram

proc () → do
  question ← "What's your name?"
  () ← print ≺ question
  name ← get ≺ ()
  () ← print ≺ concatenate("hi, ", name)
return ()
```

Figure 16: Premonoidal program in arrow do-notation and string diagrams.

Related work. Staton and Møgelberg [27] propose a formalization of Jeffrey's graphical calculus for effectful categories that arise as the Kleisli category of a strong monad. They prove that *'every strong monad is a linear-use state monad'*, that is, a state monad of the form $R \multimap !(\bullet) \otimes R$, where the state R, is an object that cannot be copied nor discarded.

7 Acknowledgements

The author wants to thank Tarmo Uustalu, Sam Staton, Niels Voorneveld and Paweł Sobociński for many helpful comments and discussion. The author gratefully thanks the anonymous reviewers at ACT'22 for many constructive suggestions that improved this manuscript. Mario Román was 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 & Bob Coecke (2009): *Categorical Quantum Mechanics*. In Kurt Engesser, Dov M. Gabbay & Daniel Lehmann, editors: *Handbook of Quantum Logic and Quantum Structures*, Elsevier, Amsterdam, pp. 261–323, doi:10.1016/B978-0-444-52869-8.50010-4.

[2] Samson Abramsky, Esfandiar Haghverdi & Philip J. Scott (2002): *Geometry of Interaction and Linear Combinatory Algebras*. *Math. Struct. Comput. Sci.* 12(5), pp. 625–665, doi:10.1017/S0960129502003730.

[3] Jean Bénabou (1967): *Introduction to bicategories*. In: *Reports of the Midwest Category Seminar*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–77, doi:10.1007/BFb0074299.

[4] Jean Bénabou (2000): *Distributors at work*. *Lecture notes written by Thomas Streicher* 11.

[5] R.F. Blute, J.R.B. Cockett, R.A.G. Seely & T.H. Trimble (1996): *Natural deduction and coherence for weakly distributive categories*. *Journal of Pure and Applied Algebra* 113(3), pp. 229–296, doi:10.1016/0022-4049(95)00159-X.

[6] Filippo Bonchi, Jens Seeber & Pawel Sobocinski (2018): *Graphical Conjunctive Queries*. In Dan R. Ghica & Achim Jung, editors: *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, *LIPIcs* 119, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 13:1–13:23, doi:10.4230/LIPIcs.CSL.2018.13.

[7] Francis Borceux (1994): *Handbook of Categorical Algebra*. *Encyclopedia of Mathematics and its Applications* 1, Cambridge University Press, doi:10.1017/CBO9780511525858.

[8] Eugenia Cheng (2021): *Distributive Laws for Lawvere Theories (Invited Talk)*. In Fabio Gadducci & Alexandra Silva, editors: *9th Conference on Algebra and Coalgebra in Computer Science, CALCO 2021, August 31 to September 3, 2021, Salzburg, Austria*, *LIPIcs* 211, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 1:1–1:1, doi:10.4230/LIPIcs.CALCO.2021.1.

[9] Brian Day & Ross Street (1997): *Monoidal Bicategories and Hopf Algebroids*. *Advances in Mathematics* 129(1), pp. 99–157, doi:10.1006/aima.1997.1649.

[10] Elena Di Lavore, Giovanni de Felice & Mario Román (2022): *Monoidal Streams for Dataflow Programming*. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3531130.3533365.

[11] Vladimir Drinfeld, Shlomo Gelaki, Dmitri Nikshych & Victor Ostrik (2010): *On braided fusion categories I*. *Selecta Mathematica* 16(1), pp. 1–119, doi:10.1007/s00029-010-0017-z.

[12] Jean-Guillaume Dumas, Dominique Duval & Jean-Claude Reynaud (2011): *Cartesian effect categories are Freyd-categories*. *Journal of Symbolic Computation* 46(3), pp. 272–293, doi:10.1016/j.jsc.2010.09.008.

[13] Jean-Yves Girard (1989): *Geometry of Interaction 1: Interpretation of System F*. In R. Ferro, C. Bonotto, S. Valentini & A. Zanardo, editors: *Logic Colloquium '88*, *Studies in Logic and the Foundations of Mathematics* 127, Elsevier, pp. 221–260, doi:10.1016/S0049-237X(08)70271-4.

[14] Marco Grandis & Robert Paré (1999): *Limits in double categories*. *Cahiers de topologie et géométrie différentielle catégoriques* 40(3), pp. 162–220. Available at `http://www.numdam.org/item/CTGDC_1999__40_3_162_0/`.

[15] René Guitart (1980): *Tenseurs et machines*. *Cahiers de topologie et géométrie différentielle catégoriques* 21(1), pp. 5–62. Available at `http://www.numdam.org/item/CTGDC_1980__21_1_5_0/`.

[16] Jules Hedges (2019): *Folklore: Monoidal kleisli categories*. Available at `https://julesh.com/2019/04/18/folklore-monoidal-kleisli-categories/`.

[17] Chris Heunen & Bart Jacobs (2006): *Arrows, like Monads, are Monoids*. In Stephen D. Brookes & Michael W. Mislove, editors: *Proceedings of the 22nd Annual Conference on Mathematical Foundations of Programming Semantics, MFPS 2006, Genova, Italy, May 23-27, 2006*, *Electronic Notes in Theoretical Computer Science* 158, Elsevier, pp. 219–236, doi:10.1016/j.entcs.2006.04.012.

[18] Naohiko Hoshino, Koko Muroya & Ichiro Hasuo (2014): *Memoryful geometry of interaction: from coalgebraic components to algebraic effects*. In Thomas A. Henzinger & Dale Miller, editors: *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, ACM, pp. 52:1–52:10, doi:10.1145/2603088.2603124.

[19] John Hughes (2000): *Generalising monads to arrows*. *Science of Computer Programming* 37(1-3), pp. 67–111, doi:10.1016/S0167-6423(99)00023-4.

[20] Bart Jacobs, Chris Heunen & Ichiro Hasuo (2009): *Categorical semantics for arrows*. *J. Funct. Program.* 19(3-4), pp. 403–438, doi:10.1017/S0956796809007308.

[21] Alan Jeffrey (1997): *Premonoidal categories and a graphical view of programs*. *Preprint at ResearchGate*. Available at `https://www.researchgate.net/profile/Alan-Jeffrey/publication/228639836_Premonoidal_categories_and_a_graphical_view_of_programs/links/00b495182cd648a874000000/Premonoidal-categories-and-a-graphical-view-of-programs.pdf`.

[22] Alan Jeffrey (1997): *Premonoidal categories and flow graphs*. *Electron. Notes Theor. Comput. Sci.* 10, p. 51, doi:10.1016/S1571-0661(05)80688-7.

[23] André Joyal & Ross Street (1991): *The geometry of tensor calculus, I*. *Advances in Mathematics* 88(1), pp. 55–112, doi:10.1016/0001-8708(91)90003-P.

[24] Paul Blain Levy (2022): *Call-by-Push-Value*. *ACM SIGLOG News* 9(2), p. 7–29, doi:10.1145/3537668.3537670.

[25] Fosco Loregian (2021): *(Co)end Calculus*. London Mathematical Society Lecture Note Series, Cambridge University Press, doi:10.1017/9781108778657.

[26] Wilhelm Magnus, Abraham Karrass & Donald Solitar (2004): *Combinatorial group theory: Presentations of groups in terms of generators and relations*. Courier Corporation.

[27] Rasmus Ejlers Møgelberg & Sam Staton (2014): *Linear usage of state*. *Log. Methods Comput. Sci.* 10(1), doi:10.2168/LMCS-10(1:17)2014.

[28] Eugenio Moggi (1991): *Notions of Computation and Monads*. *Inf. Comput.* 93(1), pp. 55–92, doi:10.1016/0890-5401(91)90052-4.

[29] David Jaz Myers (2016): *String Diagrams For Double Categories and Equipments*, doi:10.48550/ARXIV.1612.02762.

[30] Ross Paterson (2001): *A New Notation for Arrows*. In Benjamin C. Pierce, editor: *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, ACM, pp. 229–240, doi:10.1145/507635.507664.

[31] Dusko Pavlovic (2013): *Monoidal computer I: Basic computability by string diagrams*. *Inf. Comput.* 226, pp. 94–116, doi:10.1016/j.ic.2013.03.007.

[32] John Power (2002): *Premonoidal categories as categories with algebraic structure*. *Theor. Comput. Sci.* 278(1-2), pp. 303–321, doi:10.1016/S0304-3975(00)00340-6.

[33] John Power & Edmund Robinson (1997): *Premonoidal Categories and Notions of Computation*. *Math. Struct. Comput. Sci.* 7(5), pp. 453–468, doi:10.1017/S0960129597002375.

[34] John Power & Hayo Thielecke (1999): *Closed Freyd- and kappa-categories*. In Jirí Wiedermann, Peter van Emde Boas & Mogens Nielsen, editors: *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, *Lecture Notes in Computer Science* 1644, Springer, pp. 625–634, doi:10.1007/3-540-48523-6_59.

[35] Mario Román (2022): *Notes on Jeffrey's A Graphical View of Programs*. Available at `https://www.ioc.ee/~mroman/data/talks/premonoidalgraphicalview.pdf`.

[36] Christopher J. Schommer-Pries (2011): *The Classification of Two-Dimensional Extended Topological Field Theories*. arXiv:1112.1000.

[37] Michael Shulman (2008): *Framed Bicategories and Monoidal Fibrations*. *Theory and Applications of Categories* 20(18), pp. 650–738, doi:10.48550/arxiv.0706.1286.

[38] Michael Shulman (2016): *Categorical logic from a categorical point of view*. *Personal webpage*. Available at `https://mikeshulman.github.io/catlog/catlog.pdf`.

[39] Michael Shulman (2018): *The 2-Chu-Dialectica construction and the polycategory of multivariable adjunctions*. *arXiv preprint arXiv:1806.06082*, doi:10.48550/arxiv.1806.06082.

[40] Sam Staton & Paul Blain Levy (2013): *Universal properties of impure programming languages*. In Roberto Giacobazzi & Radhia Cousot, editors: *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, ACM, pp. 179–192, doi:10.1145/2429069.2429091.

[41] Todd Trimble (2010): *Coherence Theorem for Monoidal Categories (nLab entry), Section 3. Discussion*. `https://ncatlab.org/nlab/show/coherence+theorem+for+monoidal+categories`, Last accessed on 2022-05-10.

[42] Tarmo Uustalu & Varmo Vene (2008): *Comonadic Notions of Computation*. In Jiří Adámek & Clemens Kupke, editors: *Proceedings of the Ninth Workshop on Coalgebraic Methods in Computer Science, CMCS 2008, Budapest, Hungary, April 4-6, 2008*, *Electronic Notes in Theoretical Computer Science* 203, Elsevier, pp. 263–284, doi:10.1016/j.entcs.2008.05.029.

[43] Dominic Verdon (2017): *Coherence for braided and symmetric pseudomonoids*. *CoRR* abs/1705.09354, doi:10.48550/arxiv.1705.09354. arXiv:1705.09354.

4. The Produoidal Algebra of Process Decomposition

Matt Earnshaw, James Hefford, Mario Román
Computer Science Logic (CSL, 2023)

Abstract: We introduce the normal produoidal category of monoidal contexts over an arbitrary monoidal category. In the same sense that a monoidal morphism represents a process, a monoidal context represents an incomplete process: a piece of a decomposition, possibly containing missing parts. We characterize monoidal contexts in terms of universal properties. In particular, symmetric monoidal contexts coincide with monoidal lenses, endowing them with a novel universal property. We apply this algebraic structure to the analysis of multiparty interaction protocols in arbitrary theories of processes.

Declaration: *Hereby I declare that my contribution to this manuscript was to: provide the main idea and theorem, and write most of the main text, with help and revisions from Matt Earnshaw and feedback from James Hefford. The work started from discussions with James Hefford and Matt Earnshaw.*

The Produoidal Algebra of Process Decomposition

Matt Earnshaw, James Hefford and Mario Román

***Abstract*—We introduce the normal produoidal category of monoidal contexts over an arbitrary monoidal category. In the same sense that a monoidal morphism represents a process, a monoidal context represents an incomplete process: a piece of a decomposition, possibly containing missing parts. We characterize monoidal contexts in terms of universal properties. In particular, symmetric monoidal contexts coincide with monoidal lenses, endowing them with a novel universal property. We apply this algebraic structure to the analysis of multi-party interaction protocols in arbitrary theories of processes.**

1 Introduction

Theories of processes, such as *stochastic*, *partial* or *linear* functions, are a foundational tool in computer science. They help us model how systems interact in terms of a solid mathematical foundation. Any theory of processes involving operations for *sequential composition* and *parallel composition*, satisfying reasonable axioms, forms a *monoidal category*.

Monoidal categories are versatile: they can be used in the description of quantum circuits [AC09], stochastic processes [CJ19], [Fri20], relational queries [BSS18] and non-terminating processes [CL02], among many other applications [CFS16].

At the same time, monoidal categories have two intuitive, sound and complete calculi: the first in terms of *string diagrams* [JS91], and the second in terms of their *linear type theory* [Shu16]. String diagrams are a 2-dimensional syntax in which processes are represented by boxes, and their inputs and outputs are connected by wires. The type theory of symmetric monoidal categories is the basis of the more specialized *arrow do-notation* used in functional programming languages [Hug00], [Pat01], which becomes *do-notation* for Kleisli categories of commutative monads [Mog91], [Gui80]. Let us showcase monoidal categories, their string diagrams and the use of do-notation in the description of a protocol.

1.1 Protocol Description

The Transmission Control Protocol (TCP) is a connection-based communication protocol. Every connection begins with a *three-way handshake*: an exchange of messages that synchronizes the state of both parties. This handshake is defined in RFC793 to have three steps: SYN, SYN-ACK and ACK [Pos81].

The client initiates the communication by sending a synchronization packet (SYN) to the server. The synchronization packet contains a pseudorandom number associated to the session, the Initial Sequence Number of the client (CLI).

The server acknowledges this packet and sends a message (ACK) containing its own sequence number (SRV) together with the client's sequence number plus one (CLI+1). These two form the SYN-ACK message. Finally, the client sends a final ACK message with the server's sequence number plus one, SRV + 1. When the protocol works correctly, both client and server end up with the pair (CLI + 1, SRV + 1).

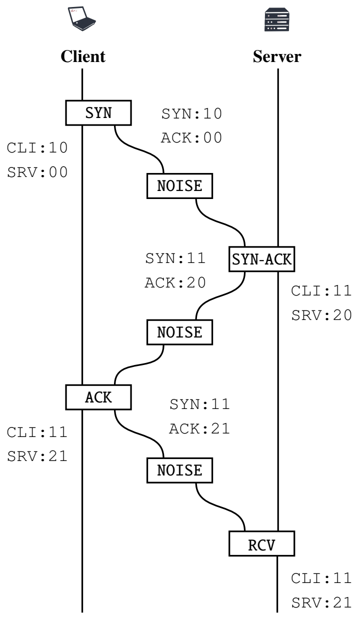


Fig. 1: TCP Three way handshake.

This protocol is traditionally described in terms of a communication diagram (Figure 1). This diagram can be taken seriously as a formal mathematical object: it is a string diagram describing a *morphism* in a monoidal category.

```
syn :: Client ~> (Client, Syn, Ack)
syn(client) = do
  client <- random
  return (client, client, 0)
```

Fig. 2: Implementation of the SYN component.

The implementation of each component of the protocol is traditionally written as pseudocode. This pseudocode can also be taken seriously as the expression of a morphism in the same monoidal category, possibly with extra structure: in this case, a commutative *Freyd category* (Figure 2, see Appendix Section A.1 [Mog91]). That is, symmetric monoidal categories admit two different internal languages, and we can use both to interpret formally the traditional description of a protocol in terms of string diagrams and pseudocode.

1.2 Types for Message Passing

The last part in formalizing a multi-party protocol in terms of monoidal categories is to actually separate its component parties. For instance, the three-way handshake can be split into the client, the server and a channel. Here is where the existing literature in monoidal categories seems to fall short: the parts resulting from the decomposition of a monoidal morphism are not necessarily monoidal morphisms themselves (see Figure 3 for the diagrammatic representation). We say that these are only *monoidal contexts*.

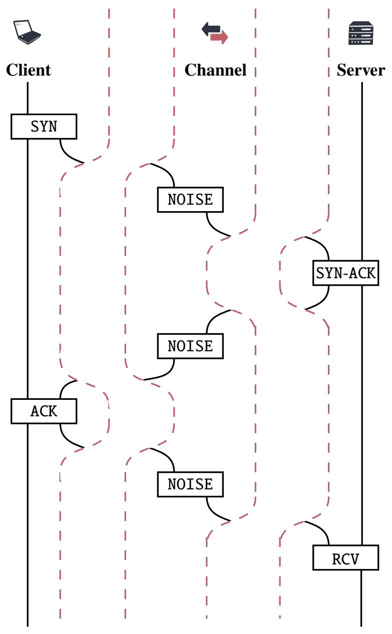


Fig. 3: Parties in the TCP Three-way handshake.

Contrary to monoidal morphisms, which only need to declare their input and output types, monoidal contexts need *behavioural types* [PS93], [HLV+16] that specify the order and type of the exchange of information along their boundary.

A monoidal context may declare intermediate *send* ($!A$) and *receive* ($?A$) types, separated by a sequencing operator ($\triangleleft$). For instance, the channel is a monoidal morphism just declaring that it takes an input message (**Msg**) and produces another output message; but the client is a monoidal context that transforms its memory type **Client** $\to$ **Client** at the same time it *sends*, *receives* and then *sends* a message; and the server transforms its type **Server** $\to$ **Server** while, dually to the client, it *receives*, *sends* and then *receives* a message.

$$💻 \in \mathcal{LC}\left(\substack{\mathbf{Client}\\ \mathbf{Client}}; !\mathbf{Msg} \triangleleft ?\mathbf{Msg} \triangleleft !\mathbf{Msg}\right);$$
$$🗄 \in \mathcal{LC}\left(\substack{\mathbf{Server}\\ \mathbf{Server}}; ?\mathbf{Msg} \triangleleft !\mathbf{Msg} \triangleleft ?\mathbf{Msg}\right);$$
$$\mathtt{NOISE} \in \mathbb{C}\left(\mathbf{Msg}; \mathbf{Msg}\right);$$

Session types [HYC08], including the send ($!A$) and receive ($?A$) polarized types, have been commonplace in logics of message passing. Cockett and Pastro [CP09] already proposed a categorical semantics for message-passing which, however, needs to go beyond monoidal categories, into *linear actegories* and *polyactegories*.

Our claim is that, perhaps surprisingly, monoidal categories already have the necessary algebraic structure to define *monoidal contexts* and their send-receive polarized types. Latent to any monoidal category, there exists a universal category of contexts with polarized types ($!/?$) and parallel/sequence operators ($\otimes/\triangleleft$).

1.3 Reasoning with Contexts

This manuscript introduces the notion of monoidal context and symmetric monoidal context; and it explains how dinaturality allows us to reason with them. In the same way that we reason with monoidal morphisms using string diagrams, we can reason about monoidal contexts using *incomplete string diagrams* [BDSPV15], [Rom21].

For instance, consider the following fact about the TCP three-way handshake: the client does not need to store a starting SRV number for the server, as it will be overwritten as soon as the real one arrives. This fact only concerns the actions of the client, and it is independent of the server and the channel. We would like to reason about it preserving this modularity, and this is what the incomplete diagrams in Figure 4 achieve.

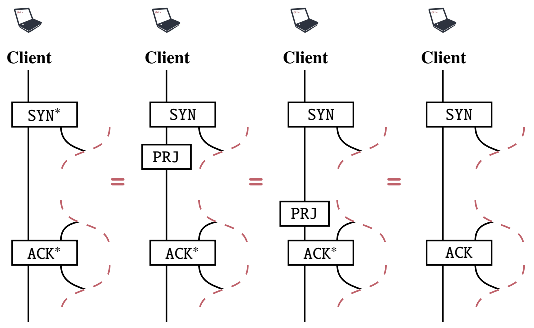


Fig. 4: Reasoning only with the Client.

Here, we define $\mathtt{SYN}^* = \mathtt{SYN} \fatsemi \mathtt{PRJ}$ to be the same as the SYN process but projecting out only the client CLI number. We also define a new $\mathtt{ACK}^*$ that ignores the server SRV number, so that $\mathtt{ACK} = \mathtt{PRJ} \fatsemi \mathtt{ACK}^*$. These two equations are enough to complete our reasoning.

Monoidal contexts and their incomplete diagrams are defined to be convenient tuples of morphisms, e.g. $(\mathtt{SYN}|\mathtt{ACK})$ in our example; what makes them interesting is the equivalence relation we impose on them: this equivalence relation makes the pair $(\mathtt{SYN} \fatsemi \mathtt{PRJ}|\mathtt{ACK}^*)$ equal to $(\mathtt{SYN}|\mathtt{PRJ} \fatsemi \mathtt{ACK}^*)$. *Dinaturality* is the name we give to this relation, and we will see how it arises canonically from the algebra of profunctors.

1.4 The Produoidal Algebra of Monoidal Context

Despite the relative popularity of string diagrams and other forms of formal 2-dimensional syntax, the algebra of incomplete monoidal morphisms has remained obscure. This manuscript elucidates this algebra: we show that, as monoidal morphisms together with their string diagrams form *monoidal categories*, monoidal contexts together with their incomplete string diagrams form *normal produoidal categories*. Normal produoidal categories were a poorly understood categorical structure, for which we provide examples. Let us motivate "normal produoidal categories" by parts.

First, the *"duoidal"* part. Monoidal contexts can be composed sequentially and in parallel, but also nested together to fill the missing parts. Nesting is captured by categorical composition, so we need specific tensors for both sequential ($\triangleleft$) and parallel ($\otimes$) composition. This is what duoidal categories provide. Duoidal categories are categories with two monoidal structures, e.g. $(\triangleleft, N)$ and $(\otimes, I)$. These two monoidal structures are in principle independent but, whenever they share the same unit ($I \cong N$), they become well-suited to express process dependence [SS22]: they become *"normal"*.

Finally, the *"pro-"* prefix. It is not that we want to impose this structure on top of the monoidal one, but we want to capture the structure morphisms already form. The two tensors $(\triangleleft, \otimes)$ do not necessarily exist in the original category; in technical terms, they are not *representable* or *functorial*, but *virtual* or *profunctorial*. This makes us turn to the produoidal categories of Booker and Street [BS13].

Not only is all of this algebra present in monoidal contexts. Monoidal contexts are the *canonical* such algebra; in a precise sense given by universal properties. The slogan for the main result of this manuscript (Theorem 6.6) is that

> Monoidal contexts are the *free* normalization of the *cofree* produoidal category over a monoidal category.

1.5 Related Work

Far from being the proposal of yet another paradigm, monoidal contexts form a novel algebraic formalization of a widespread paradigm. We argue that the idea of monoidal contexts has been recurrent in the literature, just never appearing explicitly and formally. Our main contribution is to formalize an algebra of monoidal contexts, in the form of a *normal produoidal* category.

In fact, the Symposium on Logic in Computer Science has recently seen multiple implicit applications of monoidal contexts. Kissinger and Uijlen [KU17] describe higher order quantum processes using contexts with holes in compact closed monoidal categories. Ghani, Hedges, Winschel and Zahn [GHWZ18] describe economic game theory in terms of *lenses* and incomplete processes in cartesian monoidal categories. Bonchi, Piedeleu, Sobociński and Zanasi [BPSZ19] study contextual equivalence in their monoidal category of affine signal flow graphs. Di Lavore, de Felice and Román [DLdFR22] define *monoidal streams* by iterating monoidal context coalgebraically.

Language theory. Motivated by language theory and the Chomsky-Schützenberger theorem, Melliès and Zeilberger [MZ22] were the first to present the multicategorical *splice-contour* adjunction. We are indebted to their exposition, which we extend to the promonoidal and produoidal cases. Earnshaw and Sobociński [ES22] described a congruence on formal languages of string diagrams using monoidal contexts. We prove how monoidal contexts arise from an extended produoidal splice-contour adjunction; unifying these two threads.

Session types. Session types [Hon93], [HYC08] are the mainstay type formalism for communication protocols, and they have been extensively applied to the π-calculus [SW01]. Our approach is not set up to capture all of the features of a fully fledged session type theory [KPT96]. Arguably, this makes it more general in what it does: it always provides a universal way of implementing send ($!A$) and receive ($?A$) operations in an arbitrary theory of processes represented by a monoidal category. For instance, recursion and the internal/external choice duality [GH99], [PS93] are not discussed, although they could be considered as extensions in the same way they are to monoidal categories: via trace [Has97] and linear distributivity [CS97].

Lenses and incomplete diagrams. Lenses are a notion of bidirectional transformation [FGM$^+$07] that can be cast in arbitrary monoidal categories. The first mention of monoidal lenses separate from their classical database counterparts [JRW12] is due to Pastro and Street [PS07], who identify them as an example of a promonoidal category. However, it was with a different monoidal structure [Ril18] that they became popular in recent years, spawning applications not only in bidirectional transformations [FGM$^+$07] but also in functional programming [PGW17], [CEG$^+$20], open games [GHWZ18], polynomial functors [NS22] and quantum combs [HC22]. Relating this monoidal category of lenses with the previous promonoidal category of lenses was an open problem; and the promonoidal structure was mostly ignored in applications.

We solve this problem, proving that lenses are a universal normal symmetric produoidal category (the symmetric monoidal contexts), which endows them with a novel algebra and a novel universal property. This also extends work on the relation between *incomplete diagrams*, *comb-shaped diagrams*, and *lenses* [Rom20], [Rom21].

Finally, Nester et al. have recently proposed a syntax for lenses and message-passing [Nes23], [BNR22] and lenses themselves have been applied to protocol specification [VC22]. Spivak [Spi13] also discusses the multicategory of *wiring diagrams*, later used for incomplete diagrams [PSV21] and related to lenses [SSV20]. The promonoidal categories we use can be seen as multicategories with an extra coherence property. In this sense, we contribute the missing algebraic structure of the universal multicategory of *wiring diagrams relative to a monoidal category*.

1.6 Contributions

Our main contribution is the original definition of a produoidal category of *monoidal contexts* over a monoidal category (Definition 6.1) and its characterization in terms of universal properties (Theorem 6.6).

Section 2 presents expository material on profunctors, dinaturality and promonoidal categories; the rest are novel contributions. Section 3 constructs spliced arrows as the cofree promonoidal over a category (Theorem 3.7). Section 4, on top of this, constructs spliced monoidal arrows as the cofree produoidal over a monoidal category (Theorem 4.9). Section 6 explicitly constructs a produoidal algebra of monoidal contexts (Proposition 6.5) as a free normalization. Section 7 constructs a symmetric produoidal algebra of monoidal lenses (Proposition 7.2), universally characterizing them (Theorem 7.3), and an interpretation of send/receive types ($\mathbf{!}/\mathbf{?}$) (Proposition 7.6). Section 5 introduces a novel free normalization procedure (Theorems 5.3 and 5.4) as an idempotent monad on produoidal categories, employed in Sections 6 and 7.

2 Profunctors and Virtual Structures

Profunctors describe families of processes indexed by the input and output types of a category. Profunctors provide canonical notions for *composition*, *dinaturality* and *virtual structure*. These notions are not only canonical, but also easy to reason with thanks to *coend calculus* [Lor21].

Definition 2.1. A *profunctor* $P \colon \mathbb{B}_0 \times ... \times \mathbb{B}_m \bullet\!\!\!-\!\!\circ\ \mathbb{A}_0 \times ... \times \mathbb{A}_n$ is a functor $P \colon \mathbb{A}_0^{op} ... \times \mathbb{A}_n^{op} \times \mathbb{B}_0 \times ... \times \mathbb{B}_m \to \mathbf{Set}$.

For our purposes, a profunctor $P(A_0, ..., A_n; B_0, ..., B_m)$ is a family of processes indexed by contravariant inputs $A_0, ..., A_n$ and covariant outputs $B_0, ..., B_m$. The profunctor is endowed with jointly functorial left $(\succ_0, ..., \succ_m)$ and right $(\prec_0, ..., \prec_n)$ actions of the morphisms of $\mathbb{A}_0, ..., \mathbb{A}_n$ and $\mathbb{B}_0, ..., \mathbb{B}_m$, respectively [Bén00], [Lor21].[1]

2.1 Dinaturality

Composing profunctors is subtle: the same processes could arise as the composite of different pairs of processes and so, we need to impose a careful equivalence relation. Fortunately, profunctors come with a canonical notion of dinatural equivalence which achieves precisely this.

Imagine we try to connect two different processes: $p \in P(A_0, ..., A_n; B_0, ..., B_m)$, and $q \in Q(C_0, ..., C_k; D_0, ..., D_h)$; and we have some morphism $f \colon B_i \to C_j$ that translates the i-th output port of p to the j-th input port of q. Let us write $({}_i|_j)$ for this connection operation. Note that we could connect them in two different ways:

- we could use f to change *the output of the first process* $p \prec_i f$ before connecting both, $(p \prec_i f)\ {}_i|_j\ q$;
- and we could use f to change *the input of the second process* $f \succ_j q$ before connecting both, $p\ {}_i|_j\ (f \succ_j q)$.

These are different descriptions, made up of two different components. However, they essentially describe the same process: they are *dinaturally equal* [DLdFR22]. Indeed, profunctors are canonically endowed with a notion of *dinatural equivalence* [Bén00], [Lor21], which precisely equates these two descriptions. Profunctors, and their elements, are thus composed *up to dinatural equivalence*.

Definition 2.2 (Dinatural equivalence)**.** Consider two profunctors $P \colon \mathbb{B}_0 \times ... \times \mathbb{B}_m \bullet\!\!\!-\!\!\circ\ \mathbb{A}_0 \times ... \times \mathbb{A}_n$ and $Q \colon \mathbb{C}_0 \times ... \times \mathbb{C}_k \bullet\!\!\!-\!\!\circ\ \mathbb{D}_0 \times ... \times \mathbb{D}_h$ such that $\mathbb{B}_i = \mathbb{C}_j$; and let $\mathbf{S}^{i,j}_{P,Q}(A; C)$ be the set

$$\sum_{X \in \mathbb{B}_i} P(A_0 ... A_n; B_0 ... X ... B_m) \times Q(C_0 ... X ... C_k; D_0 ... D_h).$$

Dinatural equivalence, $(\sim)$, on the set $\mathbf{S}^{i,j}_{P,Q}(A; C)$ is the smallest equivalence relation satisfying $(p \prec_i f\ {}_i|_j\ q) \sim (p\ {}_i|_j\ f \succ_j q)$. The *coend* is defined as this coproduct quotiented by dinaturality, $\mathbf{S}^{i,j}_{P,Q}(A; C)/(\sim)$, and written as an integral.

$$\int^{X \in \mathbb{C}} P(A_0 ... A_n; B_0 ... X ... B_m) \times Q(C_0 ... X ... C_k; D_0 ... D_h).$$

Definition 2.3 (Profunctor composition)**.** Consider two profunctors $P \colon \mathbb{B}_0 \times ... \times \mathbb{B}_m \bullet\!\!\!-\!\!\circ\ \mathbb{A}_0 \times ... \times \mathbb{A}_n$ and $Q \colon \mathbb{C}_0 \times ... \times \mathbb{C}_k \bullet\!\!\!-\!\!\circ\ \mathbb{D}_0 \times ... \times \mathbb{D}_h$ such that $\mathbb{B}_i = \mathbb{C}_j$; their *composition* along ports i and j is a profunctor; we write it marking this connection

$$P(A_0 ... A_n; B_0 ... \bullet_x ... B_n) \diamond Q(C_0 ... \bullet_x ... C_k; D_0 ... D_h),$$

and it is defined as the coproduct of the product of both profunctors, indexed by the common variable, and quotiented by dinatural equivalence,

$$\int^{X \in \mathbb{C}} P(A_0 ... A_n; B_0 ... X ... B_m) \times Q(C_0 ... X ... C_k; D_0 ... D_h).$$

Remark 2.4 (Representability)*.* Every functor $F \colon \mathbb{A} \to B$ gives rise to two different profunctors: its representable profunctor $\mathbb{A}(F\bullet, \bullet) \colon \mathbb{A} \circ\!\!-\!\!\!\bullet\ \mathbb{B}$, and its corepresentable profunctor $\mathbb{A}(\bullet, F\bullet) \colon \mathbb{A} \bullet\!\!\!-\!\!\circ\ \mathbb{B}$. We say that a profunctor is *representable* or *corepresentable* if it arises in this way. Under this interpretation, functors are profunctors that happen to be representable. This suggests that we can repeat structures based on functors, such as monoidal categories, now in terms of profunctors.

We justified in the introduction the importance of monoidal categories: they are the algebra of processes composing sequentially and in parallel, joining and splitting resources. However, there exist some theories that can deal only with splitting without being necessarily full theories of processes: that is, we may be able to talk about splitting without being able to talk about joining. Such "monoidal categories on one side" are *promonoidal categories*.

The difference between monoidal categories and promonoidal categories is that the tensor is no longer a functor but is instead a profunctor.[2] In other words, the tensor is no longer representable – such a structure is called *virtual*, as in *virtual double* and *virtual duoidal* categories [CS10], [Shu17].

[1]We simply use $(\prec/\succ)$ without any subscript whenever the input/output is unique. See Appendix, Section B for more details on profunctors.

[2]In more technical terms, monoidal categories are pseudomonoids in the monoidal bicategory of categories *and functors*; while promonoidal categories are pseudomonoids in the monoidal bicategory of categories *and profunctors*.

2.2 Promonoidal Categories

Promonoidal categories are the algebra of *coherent decomposition*. A category $\mathbb{C}$ contains sets of *morphisms*, $\mathbb{C}(X;Y)$. In the same way, a promonoidal category $\mathbb{V}$ contains sets of *splits*, $\mathbb{V}(X;Y_0 \triangleleft Y_1)$, *morphisms*, $\mathbb{V}(X;Y)$, and *units*, $\mathbb{V}(X;N)$, where N is the virtual tensor unit. Splits, $\mathbb{V}(X;Y_0 \triangleleft Y_1)$, represent a way of decomposing objects of type X into objects of type Y_0 and Y_1. Morphisms, $\mathbb{V}(X;Y)$, as in any category, are transformations of X into Y. Units, $\mathbb{V}(X;N)$, are the atomic pieces of type X.

These decompositions must be coherent. For instance, imagine we want to split X into Y_0, Y_1 and Y_2. Splitting X into Y_0 and something $(\bullet)$, and then splitting that something into Y_1 and Y_2 *should be doable in essentially the same ways* as splitting X into something $(\bullet)$ and Y_2, and then splitting that something into Y_0 and Y_1. Formally, we are saying that,

$$\mathbb{V}(X;Y_0 \triangleleft \bullet) \diamond \mathbb{V}(\bullet;Y_1 \triangleleft Y_2) \cong \mathbb{V}(X;\bullet \triangleleft Y_2) \diamond \mathbb{V}(\bullet;Y_0 \triangleleft Y_1),$$

and, in fact, we just write $\mathbb{V}(X;Y_0 \triangleleft Y_1 \triangleleft Y_2)$ for the set of such decompositions.

Definition 2.5 (Promonoidal category)**.** A *promonoidal category* is a category $\mathbb{V}(\bullet;\bullet)$ endowed with profunctors

$$\mathbb{V}(\bullet;\bullet \triangleleft \bullet) \colon \mathbb{V} \times \mathbb{V} \multimapdotboth \mathbb{V}, \text{ and } \mathbb{V}(\bullet;N) \colon 1 \multimapdotboth \mathbb{V}.$$

Equivalently, these are functors

$$\mathbb{V}(\bullet;\bullet \triangleleft \bullet) \colon \mathbb{V}^{\mathrm{op}} \times \mathbb{V} \times \mathbb{V} \to \mathbf{Set}, \text{ and } \mathbb{V}(\bullet;N) \colon \mathbb{V}^{\mathrm{op}} \to \mathbf{Set}.$$

Moreover, promonoidal categories must be endowed with the following natural isomorphisms,

$$\begin{aligned}\mathbb{V}(X;\bullet \triangleleft Y_2) \diamond \mathbb{V}(\bullet;Y_0 \triangleleft Y_1) &\cong \mathbb{V}(X;\bullet \triangleleft Y_2) \diamond \mathbb{V}(\bullet;Y_0 \triangleleft Y_1),\\ \mathbb{V}(X;\bullet \triangleleft Y) \diamond \mathbb{V}(\bullet;N) &\cong \mathbb{V}(X;Y),\\ \mathbb{V}(X;Y \triangleleft \bullet) \diamond \mathbb{V}(\bullet;N) &\cong \mathbb{V}(X;Y),\end{aligned}$$

called α, λ, ρ, respectively, and asked to satisfy the pentagon and triangle coherence equations, $\alpha \fatsemi \alpha = (\alpha \diamond 1) \fatsemi \alpha \fatsemi (1 \diamond \alpha)$, and $(\rho \diamond 1) = \alpha \fatsemi (\lambda \diamond 1)$.

Definition 2.6 (Promonoidal functor)**.** Let $\mathbb{V}$ and $\mathbb{W}$ be promonoidal categories. A *promonoidal functor* $F \colon \mathbb{V}(\bullet,\bullet) \to \mathbb{W}(\bullet,\bullet)$ is a functor between the two categories, together with natural transformations:

$$\begin{aligned}&F_{\triangleleft} \colon \mathbb{V}(A;B \triangleleft C) \to \mathbb{W}(FA;FB \triangleleft FC), \text{ and}\\ &F_N \colon \mathbb{V}(A;N) \to \mathbb{W}(FA;N),\end{aligned}$$

that satisfy $\lambda \fatsemi F_{\mathrm{map}} = (F_{\triangleleft} \times F_N) \fatsemi \lambda$, $\rho \fatsemi F_{\mathrm{map}} = (F_{\triangleleft} \times F_N) \fatsemi \rho$, and $\alpha \fatsemi (F_{\triangleleft} \times F_{\triangleleft}) \fatsemi i = (F_{\triangleleft} \times F_{\triangleleft}) \fatsemi i \fatsemi \alpha$. We denote by **Promon** the category of promonoidal categories and promonoidal functors.

Remark 2.7 (Promonoidal coherence). As with monoidal categories, the pentagon and triangle equations imply that every formal equation written out of coherence isomorphisms holds. This means we can write $\mathbb{V}(\bullet;\bullet \triangleleft \bullet \triangleleft \bullet)$ without specifying which one of the two sides of the associator we are describing.

Remark 2.8 (Multicategories). The reader may be more familiar with the algebra of not-necessarily-coherent decomposition: *multicategories*. Every promonoidal category $\mathbb{V}$ induces a co-multicategory with morphisms given by elements of the following sets $\mathbb{V}(\bullet;\bullet \triangleleft \overset{n}{\ldots} \triangleleft \bullet)$. Similarly, $\mathbb{V}^{\mathrm{op}}$ is a co-promonoidal category and thus induces a multicategory. These are special kinds of (co-)multicategories, they are *coherent* so that every n-to-1 morphism splits, in any possible shape, as 2-to-1 and 0-to-1 morphisms; moreover, they do so *uniquely up to dinaturality*. Appendix B.2 spells out this relation.

The next section studies how to coherently decompose morphisms of a category. Categories are an algebraic structure for sequential composition: they contain a "sequencing" operator $(\fatsemi)$ and a neutral element, id. We present an algebra for decomposing sequential compositions in terms of promonoidal categories.

3 Sequential context

Assume a morphism factors as follows,

$$f_0 \fatsemi g_0 \fatsemi h \fatsemi g_1 \fatsemi f_1 \fatsemi k \fatsemi f_2.$$

We can say that this morphism came from the context $f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2$, filled on its left side with the context $g_0 \fatsemi \square \fatsemi g_1$, then filled with h, and finally completed on its right side with the morphism k. Figure 5 expresses this decomposition.

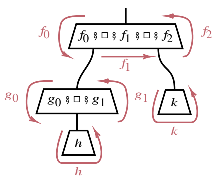

f_0
$f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2$
f_2
f_1
g_0
$g_0 \fatsemi \square \fatsemi g_1$
g_1
k
k
h
h

Fig. 5: Decomposition of $f_0 \fatsemi g_0 \fatsemi h \fatsemi g_1 \fatsemi f_1 \fatsemi k \fatsemi f_2$.

Contexts compose in a tree-like structure, and their resulting morphism is extracted by *contouring* that tree. This section presents the algebra of *context* and *decomposition*. We then prove that they are two sides of the same coin: the two sides of an adjunction of categories.

3.1 Contour of a Promonoidal Category

Any promonoidal category freely generates another category, its *contour*. This can be interpreted as the category that tracks the processes of decomposition that the promonoidal category describes. The construction is particularly pleasant from the geometric point of view: it takes its name from the fact that it can be constructed by following the contour of the shape of the decomposition.

Definition 3.1 (Contour)**.** The *contour* of a promonoidal category $\mathbb{V}$ is a category $\mathcal{C}\mathbb{V}$ that has two objects, X^L (left-handed) and X^R (right-handed), for each object $X \in \mathbb{V}_{\mathrm{obj}}$; and has as arrows those that arise from *contouring* the decompositions of the promonoidal category.

Specifically, it is freely presented by *(i)* a morphism $a_0 \in \mathcal{C}\mathbb{V}(A^L;A^R)$, for each unit $a \in \mathbb{V}(A;N)$; *(ii)* a pair of morphisms $b_0 \in \mathcal{C}\mathbb{V}(B^L;X^L)$, $b_1 \in \mathcal{C}\mathbb{V}(X^R;B^R)$, for each

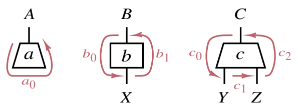


Fig. 6: Contour of a promonoidal.

morphism $b \in \mathbb{V}(B; X)$; and *(iii)* a triple of morphisms $c_0 \in C\mathbb{V}(C^L; Y^L)$, $c_1 \in C\mathbb{V}(Y^R; Z^L)$, $c_2 \in C\mathbb{V}(Z^R; C^R)$ for each split $c \in \mathbb{V}(C; Y \triangleleft Z)$, see Figure 6.

For each equality $\alpha(a \,|\, b) = (c \,|\, d)$, we impose the equations $a_0 = c_0 \fatsemi d_0$; $a_1 \fatsemi b_0 = d_1$ and $b_1 = d_2 \fatsemi c_1$; $a_2 \fatsemi b_2 = c_2$. For each equality $\rho(a \,|\, b) = c = \lambda(d \,|\, e)$, we impose $a_0 = c_0 = d_0 \fatsemi e_0 \fatsemi d_1$ and $a_1 \fatsemi b_0 \fatsemi a_2 = c_1 = d_2$. Graphically, these follow Figure 7.

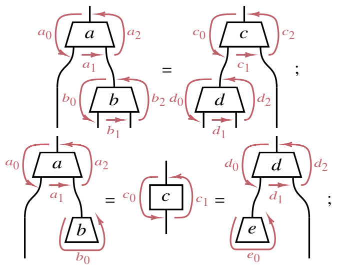


Fig. 7: Equations between contours from α, ρ, and λ in $\mathbb{V}$.

Proposition 3.2. *Contour gives a functor C :* **Promon** $\to$ **Cat***.*

Proof. See Appendix, Proposition C.1. □

Remark 3.3. The contour of a multicategory was first introduced by Melliès and Zeilberger [MZ22]. Definition 3.1 and the following Theorem 3.7 closely follow their work; although the promonoidal version we introduce does involve fewer equations due to the extra coherence (Remark 2.8).

3.2 The Promonoidal Category of Spliced Arrows

We described a category tracking the process of decomposing in a given promonoidal category. However, we want to go the other way around: given a category, what is the promonoidal category describing decomposition in that category? This subsection finds a right adjoint to the contour construction: the spliced arrows promonoidal category. Spliced arrows have already been used to describe context in parsing [MZ22].

Definition 3.4 (Spliced arrows)**.** Let $\mathbb{C}$ be a category. The promonoidal category of *spliced arrows*, $\mathcal{S}\mathbb{C}$, has as objects pairs of objects of $\mathbb{C}$. It uses the following profunctors to define morphisms, splits and units.

$$\mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y, B);$$
$$\mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; B);$$
$$\mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; N\right) = \mathbb{C}(A; B).$$

In other words, morphisms are *pairs of arrows* $f : A \to X$ and $g : Y \to B$. Splits are *triples of arrows* $f : A \to X$, $g : Y \to X'$ and $h : Y' \to B$. Units are simply *arrows* $f : A \to B$. We use the following notation for

morphisms,	$f \fatsemi \square \fatsemi g$	$\in \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right)$;
splits,	$f \fatsemi \square \fatsemi g \fatsemi \square \fatsemi h$	$\in \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right)$;
and units,	f	$\in \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; N\right)$.

The profunctor actions, associativity and unitality of the promonoidal category are defined in a straightforward way by *filling the holes*. For instance,

$$(f \fatsemi \square \fatsemi g \fatsemi \square \fatsemi h) \prec_1 (u \fatsemi \square \fatsemi v) = (f \fatsemi u \fatsemi \square \fatsemi v \fatsemi g \fatsemi \square \fatsemi h),$$
$$(f \fatsemi \square \fatsemi g \fatsemi \square \fatsemi h) \prec_2 (u \fatsemi \square \fatsemi v) = (f \fatsemi \square \fatsemi g \fatsemi u \fatsemi \square \fatsemi v \fatsemi h).$$

See the Appendix, Section C for details.

Proposition 3.5. *Spliced arrows form a promonoidal category with their splits, units, and suitable coherence morphisms.*

Proof. See Appendix, Proposition C.2. □

As a consequence, we can talk about spliced arrows with an arbitrary number of holes: for instance, a three-way split arises as a split filled by another split, in either position. For instance,

$$\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2 \fatsemi \square \fatsemi f_3 \rangle$$

can be written in two different ways,

$$\langle f_0 \fatsemi \square \fatsemi f_2 \fatsemi \square \fatsemi f_3 \rangle \prec_1 \langle id \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi id \rangle \quad \text{or}$$
$$\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_3 \rangle \prec_2 \langle id \fatsemi \square \fatsemi f_2 \fatsemi \square \fatsemi id \rangle.$$

Proposition 3.6. *Splice gives a functor $\mathcal{S}$:* **Cat** $\to$ **Promon***.*

Proof. See Appendix, Proposition C.6. □

Theorem 3.7. *There exists an adjunction between categories and promonoidal categories, where the contour of a promonoidal is the left adjoint, and the splice category is the right adjoint.*

Proof. See Appendix, Theorem C.7. □

Spliced arrows can be computed for *any* category, including monoidal categories. However, we expect the spliced arrows of a monoidal category to have a richer algebraic structure. This extra structure is the subject of the next section.

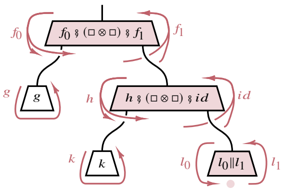
4 Parallel-Sequential Context

Monoidal categories are an algebraic structure for sequential and parallel composition: they contain a "tensoring" operator on morphisms, $(\otimes)$, apart from the usual sequencing, $(\fatsemi)$, and identities (id).

Assume a monoidal morphism factors as follows,

$$f_0 \fatsemi (g \otimes (h \fatsemi (k \otimes (l_0 \fatsemi l_1)))) \fatsemi f_1.$$

We can say that this morphism came from dividing everything between f_0 and f_1 by a tensor. That is, from a context $f_0 \fatsemi (\square \otimes \square) \fatsemi f_1$. We filled the first hole of this context with a g, and then proceeded to split the second part as $h \fatsemi (\square \otimes \square) \fatsemi \mathrm{id}$. Finally, we filled the first part with k and the second one we left disconnected by filling it with l_0, id_I, and l_1.



$f_0 ⨟ (\square \otimes \square) ⨟ f_1$
f_0
f_1
g
h
$h ⨟ (\square \otimes \square) ⨟ id$
id
k
l_0
$l_0 \| l_1$
l_1

Fig. 8: Decomposition of $f_0 ⨟ (g \otimes (h ⨟ (k \otimes (l_0 ⨟ l_1)))) ⨟ f_1$.

This section studies decomposition of morphisms in a *monoidal* category, in the same way we studied decomposition of morphisms in a category before. We present an algebraic structure for decomposing both sequential and parallel compositions: *produoidal categories.*

4.1 Produoidal Categories

Produoidal categories, first defined by Booker and Street [BS13], provide an algebraic structure for the interaction of sequential and parallel decomposition. A produoidal category $\mathbb{V}$ not only contains *morphisms*, $\mathbb{V}(X;Y)$, *sequential splits*, $\mathbb{V}(X;Y_0 \triangleleft Y_1)$, and *sequential units*, $\mathbb{V}(X;N)$, as a promonoidal category does; it also contains *parallel splits*, $\mathbb{V}(X;Y_0 \otimes Y_1)$ and *parallel units*, $\mathbb{V}(X;I)$.

Remark 4.1 (Nesting virtual structures). Notation for nesting functorial structures, say $(\triangleleft)$ and $(\otimes)$, is straightforward: we use expressions like $(X_1 \otimes Y_1) \triangleleft (X_2 \otimes Y_2)$ without a second thought. Nesting the virtual structures $(\triangleleft)$ and $(\otimes)$ is more subtle: defining $\mathbb{V}(\bullet; X \otimes Y)$ and $\mathbb{V}(\bullet; X \triangleleft Y)$ for each pair of objects X and Y does not itself define what something like $\mathbb{V}(\bullet; (X_1 \otimes Y_1) \triangleleft (X_2 \otimes Y_2))$ means. Recall that, in the virtual case, $X_1 \triangleleft Y_1$ and $X_1 \otimes Y_1$ are not objects themselves: they are just names for the profunctors $\mathbb{V}(\bullet; X_1 \triangleleft Y_1)$ and $\mathbb{V}(\bullet; X_1 \otimes Y_1)$.

Instead, when we write $\mathbb{V}(\bullet; (X_1 \otimes Y_1) \triangleleft (X_2 \otimes Y_2))$, we formally mean $\mathbb{V}(\bullet; \bullet_1 \triangleleft \bullet_2) \diamond \mathbb{V}(\bullet_1; X_1 \otimes Y_1) \diamond \mathbb{V}(\bullet_2; X_2 \otimes Y_2)$. By convention, nesting virtual structures means profunctor composition in this text.

Definition 4.2 (Produoidal category)**.** A *produoidal category* is a category $\mathbb{V}$ endowed with two promonoidal structures,

$$\mathbb{V}(\bullet; \bullet \otimes \bullet) \colon \mathbb{V} \times \mathbb{V} \mathrel{\bullet\!\!-\!\!\circ} \mathbb{V}, \text{ and } \mathbb{V}(\bullet; I) \colon 1 \mathrel{\bullet\!\!-\!\!\circ} \mathbb{V},$$
$$\mathbb{V}(\bullet; \bullet \triangleleft \bullet) \colon \mathbb{V} \times \mathbb{V} \mathrel{\bullet\!\!-\!\!\circ} \mathbb{V}, \text{ and } \mathbb{V}(\bullet; N) \colon 1 \mathrel{\bullet\!\!-\!\!\circ} \mathbb{V},$$

such that one laxly distributes over the other. This is to say that it is endowed with the following natural *laxators*,

$$\psi_2 \colon \mathbb{V}(\bullet; (X \triangleleft Y) \otimes (Z \triangleleft W)) \to \mathbb{V}(\bullet; (X \otimes Z) \triangleleft (Y \otimes W)),$$
$$\psi_0 \colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; I \triangleleft I),$$
$$\varphi_2 \colon \mathbb{V}(\bullet; N \otimes N) \to \mathbb{V}(\bullet; N),$$
$$\varphi_0 \colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; N).$$

Laxators, together with unitors and associators, must satisfy coherence conditions (see Appendix, Definition I.5).

Definition 4.3 (Produoidal functor)**.** Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ and $\mathbb{W}_{\oslash,J,\blacktriangleleft,M}$ be produoidal categories. A *produoidal functor* F is a functor between the two categories $F : \mathbb{V}(\bullet,\bullet) \to \mathbb{W}(\bullet,\bullet)$ together with natural transformations

$$F_\otimes : \mathbb{V}(A; B \otimes C) \to \mathbb{W}(FA; FB \oslash FC),$$
$$F_I : \mathbb{V}(A; I) \to \mathbb{W}(FA; J),$$
$$F_\triangleleft : \mathbb{V}(A; B \triangleleft C) \to \mathbb{W}(FA; FB \blacktriangleleft FC), \text{ and}$$
$$F_N : \mathbb{V}(A; N) \to \mathbb{W}(FA; M),$$

preserving coherence isomorphisms for each promonoidal structure, and the laxators. Denote by **Produo** the category of produoidal categories and produoidal functors.

4.2 Monoidal Contour of a Produoidal Category

Any produoidal category freely generates a monoidal category, its *monoidal contour*. Again, this is interpreted as a monoidal category tracking the processes of parallel and sequential decomposition described by the produoidal category. And again, the construction follows a pleasant geometric pattern, where we follow the shape of the decomposition, now in both the parallel and sequential dimensions.

Definition 4.4 (Monoidal contour)**.** The *contour* of a produoidal category $\mathbb{B}$ is the monoidal category $\mathcal{D}\mathbb{B}$ that has two objects, X^L (left-handed) and X^R (right-handed), for each object $X \in \mathbb{B}_{\text{obj}}$; and has arrows those that arise from *contouring* both sequential and parallel decompositions of the promonoidal category.

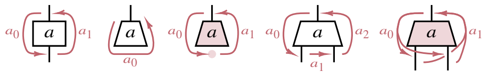


Fig. 9: Generators of the monoidal category of contours.

Specifically, it is freely presented by *(i)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L)$, $a_1 \in \mathcal{D}\mathbb{B}(X^R; A^R)$ for each morphism $a \in \mathbb{B}(A; X)$; *(ii)* a morphism $a_0 \in \mathcal{D}\mathbb{B}(A^L; A^R)$, for each sequential unit $a \in \mathbb{C}(A; N)$; *(iii)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; I)$ and $a_0 \in \mathcal{D}\mathbb{B}(I; A^R)$, for each parallel unit $a \in \mathbb{B}(A; I)$; *(iv)* a triple of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L)$, $a_1 \in \mathcal{D}\mathbb{B}(X^R; Y^L)$, $a_2 \in \mathcal{D}\mathbb{B}(Y^R; A^R)$ for each sequential split $a \in \mathbb{B}(A; X \triangleleft Y)$; and *(v)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L \otimes Y^L)$ and $a_1 \in \mathcal{D}\mathbb{B}(X^R \otimes Y^R; A^R)$ for each parallel split $a \in \mathbb{B}(A; X \otimes Y)$, see Figure 9.

We impose the same equations as in the categorical contour coming from the associator and unitor of the $\triangleleft$ structure; but moreover, we impose the following new equations, coming from the $\otimes$ structure: For each application of associativity, $\alpha(a ⨟_1 b) = c ⨟_2 d$, we impose the equations $a_0 ⨟ (b_0 \otimes \mathrm{id}) = c_0 ⨟ (\mathrm{id} \otimes d_0)$ and $(b_1 \otimes \mathrm{id}) ⨟ a_1 = (\mathrm{id} \otimes d_1) ⨟ c_1$. These follow from Figure 10.

For each application of unitality, $\lambda(a ⨟_1 b) = c = \rho(d ⨟_2 e)$, we impose the equations $a_0 ⨟ (b_0 \otimes \mathrm{id}) = c_0 = d_0 ⨟ (\mathrm{id} \otimes e_0)$ and $(b_1 \otimes \mathrm{id}) ⨟ a_1 = c_1 = (\mathrm{id} \otimes e_1) ⨟ d_1$. These follow from Figure 11.

For each application of the laxator, $\psi_2(a \,|\, b \,|\, c) = (d \,|\, e \,|\, f)$, we impose the equation $a_0 ⨟ (b_0 \otimes c_0) = d_0 ⨟ e_0$, the middle

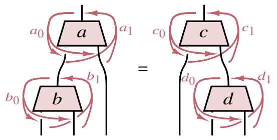


Fig. 10: Equation between contours from ⊗ associator.

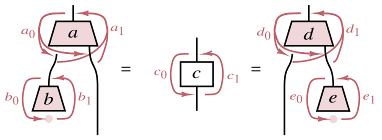


Fig. 11: Equations from ⊗ unitor.

equation $b_1 \otimes c_1 = e_1 \mathbin{⨟} d_1 \mathbin{⨟} f_0$, and $(b_2 \otimes c_2) \mathbin{⨟} a_1 = f_1 \mathbin{⨟} d_2$. These follow Figure 12. We finally impose similar equations for the rest of the laxators, see Definition D.1 for details.

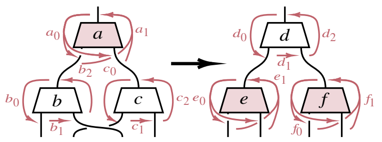


Fig. 12: Equations from the laxator ψ_2.

Proposition 4.5. *Monoidal contour extends to a functor $\mathcal{D}$:* **Produo → Mon***.*

Proof. See Appendix, Proposition D.2. □

4.3 Produoidal Category of Spliced Monoidal Arrows

Again, we want to go the other way around: given a monoidal category, what is the produoidal category that tracks decomposition of arrows in that monoidal category? This subsection finds a right adjoint to the monoidal contour construction: the produoidal category of *spliced monoidal arrows.*

Definition 4.6. Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. The produoidal category of *spliced monoidal arrows*, $\mathcal{T}\mathbb{C}$, has as objects pairs of objects of $\mathbb{C}$. It uses the following profunctors to define sequential splits, parallel splits, sequential units, parallel units and morphisms.

$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y, B);$$
$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y} \triangleleft \tfrac{X'}{Y'}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; B);$$
$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y} \otimes \tfrac{X'}{Y'}\right) = \mathbb{C}(A; X \otimes X') \times \mathbb{C}(Y \otimes Y'; B);$$
$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; N\right) = \mathbb{C}(A; B);$$
$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; I\right) = \mathbb{C}(A; I) \times \mathbb{C}(I; B).$$

In other words, morphisms are *pairs of arrows* $f: A \to X$ and $g: Y \to B$. sequential splits are *triples of arrows* $f: A \to X$, $g: Y \to X'$ and $h: Y' \to B$. Parallel splits are *pairs of arrows* $f: A \to X \otimes X'$ and $g: Y \otimes Y' \to B$. Sequential units are *arrows* $f: A \to B$. parallel units are pairs of arrows $f: A \to I$ and $g: I \to B$. In summary, we have

morphisms,	$f \mathbin{⨟} \square \mathbin{⨟} g$	$\in \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y}\right)$;
sequential splits,	$f \mathbin{⨟} \square \mathbin{⨟} g \mathbin{⨟} \square \mathbin{⨟} h$	$\in \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y} \triangleleft \tfrac{X'}{Y'}\right)$;
parallel splits,	$f \mathbin{⨟} (\square \otimes \square) \mathbin{⨟} h$	$\in \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y} \otimes \tfrac{X'}{Y'}\right)$;
sequential units,	f	$\in \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; N\right)$.
and parallel units,	$f \parallel g$	$\in \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; I\right)$.

Finally, the laxators unite two different connections between two gaps into a single one. For instance, the last laxator takes parallel sequences of holes,

$$f_0 \mathbin{⨟} ((h_0 \mathbin{⨟} \square \mathbin{⨟} h_1 \mathbin{⨟} \square \mathbin{⨟} h_2) \otimes (k_0 \mathbin{⨟} \square \mathbin{⨟} k_1 \mathbin{⨟} \square \mathbin{⨟} k_2)) \mathbin{⨟} f_1$$

into sequences of parallel holes,

$$f_0 \mathbin{⨟} (h_0 \otimes k_0) \mathbin{⨟} (\square \otimes \square) \mathbin{⨟} (h_1 \otimes k_1) \mathbin{⨟} (\square \otimes \square) \mathbin{⨟} (h_2 \otimes k_2) \mathbin{⨟} f_1.$$

See Appendix, Section D.2 for details.

Proposition 4.7. *Spliced monoidal arrows form a produoidal category with their sequential and parallel splits, units, and suitable coherence morphisms and laxators.*

Proof. See Appendix, Proposition D.3. □

Proposition 4.8. *Spliced monoidal arrows extends to a functor $\mathcal{T}$:* **Mon → Produo***.*

Proof. See Appendix, Proposition D.8. □

As in the categorical case, spliced monoidal arrows and monoidal contour again form an adjunction. This adjunction characterizes spliced monoidal arrows as a cofree construction.

Theorem 4.9. *There exists an adjunction between monoidal categories and produoidal categories, where the monoidal contour is the left adjoint, and the produoidal splice category is the right adjoint.*

Proof. See Appendix, Theorem D.9. □

4.4 Representable Parallel Structure

A produoidal category has two tensors, and neither is, in principle, representable. However, the cofree produoidal category over a category we have just constructed happens also to have a representable tensor, (⊗). Spliced monoidal arrows form a monoidal category.

Proposition 4.10. *Parallel splits and parallel units of spliced monoidal arrows are representable profunctors. Explicitly,*

$$\mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X}{Y} \otimes \tfrac{X'}{Y'}\right) \cong \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{X \otimes X'}{Y \otimes Y'}\right), \text{ and } \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; I\right) \cong \mathcal{T}\mathbb{C}\left(\tfrac{A}{B}; \tfrac{I}{I}\right).$$

In fact, these sets are equal by definition. However, there is a good reason to work in the full generality of produoidal categories: every produoidal category, representable or not, has an associated *normal* produoidal category, which may be again representable or not. Normalization is a canonical procedure to mix both tensors, (⊗) and (◁); and it will allow us to write *monoidal contexts* in Section 6, which form a produoidal category without representable structure.

Remark 4.11. This means $\mathcal{TC}$ has the structure of a *virtual duoidal category* [Shu17] or *monoidal multicategory*, defined by Aguiar, Haim and López Franco [AHLF18] as a pseudomonoid in the cartesian monoidal 2-category of multicategories.

5 Interlude: Normalization

Produoidal categories seem to contain too much structure: of course, we want to split things in two different ways, sequentially ($\triangleleft$) and in parallel ($\otimes$); but that does not necessarily mean that we want to keep track of two different types of units, parallel (I) and sequential (N). The atomic components of our decomposition algebra should be the same, without having to care if they are *atomic for sequential composition* or *atomic for parallel composition*.

Fortunately, there exists an abstract procedure that, starting from any produoidal category, constructs a new produoidal category where both units are identified. This procedure is known as *normalization*, and the resulting produoidal categories are called *normal*.

Definition 5.1 (Normal produoidal category)**.** A *normal produoidal category* is a produoidal category where the laxator $\varphi_0 \colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; N)$ is an isomorphism.

Normal produoidal categories form a category **nProduo** with produoidal functors between them and endowed with fully faithful forgetful functor $\mathcal{U} \colon \mathbf{nProduo} \to \mathbf{Produo}$.

Theorem 5.2. *Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a produoidal category. The profunctor $\mathcal{N}\mathbb{V}(\bullet;\bullet) = \mathbb{V}(\bullet; N \otimes \bullet \otimes N)$ forms a promonad. Moreover, the Kleisli category of this promonad is a normal produoidal category with the following splits and units.*

$$\begin{aligned}
\mathcal{N}\mathbb{V}(A;B) &= \mathbb{V}(A; N \otimes B \otimes N);\\
\mathcal{N}\mathbb{V}(A; B \otimes_N C) &= \mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N);\\
\mathcal{N}\mathbb{V}(A; B \triangleleft_N C) &= \mathbb{V}(A; (N \otimes B \otimes N) \triangleleft (N \otimes C \otimes N));\\
\mathcal{N}\mathbb{V}(A; I_N) &= \mathbb{V}(A; N);\\
\mathcal{N}\mathbb{V}(A; N_N) &= \mathbb{V}(A; N).
\end{aligned}$$

Proof. See Appendix, Theorem E.1. □

A normalization procedure for duoidal categories was given by Garner and López Franco [GF16]; our contribution is its produoidal counterpart. This novel produoidal normalization is better behaved than the duoidal one [GF16]: the latter does not always exist, but we show produoidal normalization does. Indeed, we prove that produoidal normalization forms an idempotent monad. The technical reason for this improvement is that the original required the existence of certain coequalizers in $\mathbb{V}$; produoidal normalization uses coequalizers in **Set**. Appendix E.3 outlines a relation between the two procedures.

Theorem 5.3. *Normalization extends to an idempotent monad.*

Proof. See Appendix, Theorem E.3. □

Theorem 5.4 (Free normal produoidal)**.** *Normalization determines an adjunction between produoidal categories and normal produoidal categories, $\mathcal{N} \colon$ **Produo** $\rightleftharpoons$ **nProduo** $\colon \mathcal{U}$. That is, $\mathcal{N}\mathbb{V}$ is the free produoidal category over $\mathbb{V}$.*

Proof. See Appendix, Theorem E.5. □

In the previous Section 4, we constructed the produoidal category of spliced monoidal arrows, which distinguishes between morphisms and morphisms with a hole in the monoidal unit. This is because the latter hole splits the morphism in two parts. Normalization equates both; it sews these two parts. In Section 6, we explicitly construct monoidal contexts, the normalization of spliced monoidal arrows.

5.1 Symmetric Normalization

Normalization is a generic procedure that applies to any produoidal category, it does not matter if the parallel split ($\otimes$) is symmetric or not. However, when $\otimes$ happens to be symmetric, we can also apply a more specialized normalization procedure.

Definition 5.5 (Symmetric produoidal category)**.** A *symmetric produoidal category* is a produoidal category $\mathbb{V}_{\triangleleft,N,\otimes,I}$ endowed with a natural isomorphism $\sigma \colon \mathbb{V}(A; B \otimes C) \cong \mathbb{V}(A; C \otimes B)$ satisfying the symmetry and hexagon equations.

Theorem 5.6. *Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a symmetric produoidal category. The profunctor $\mathcal{N}_\sigma\mathbb{V}(\bullet;\bullet) = \mathbb{V}(\bullet; N \otimes \bullet)$ forms a promonad. Moreover, the Kleisli category of this promonad is a normal symmetric produoidal category with the following splits and units.*

$$\begin{aligned}
\mathcal{N}_\sigma\mathbb{V}(A;B) &= \mathbb{V}(A; N \otimes B);\\
\mathcal{N}_\sigma\mathbb{V}(A; B \otimes_N C) &= \mathbb{V}(A; N \otimes B \otimes C);\\
\mathcal{N}_\sigma\mathbb{V}(A; B \triangleleft_N C) &= \mathbb{V}(A; (N \otimes B) \triangleleft (N \otimes C));\\
\mathcal{N}_\sigma\mathbb{V}(A; I_N) &= \mathbb{V}(A; N);\\
\mathcal{N}_\sigma\mathbb{V}(A; N_N) &= \mathbb{V}(A; N).
\end{aligned}$$

Proof. See Appendix, Theorem E.6. □

Theorem 5.7. *Normalization determines an adjunction between symmetric produoidal and normal symmetric produoidal categories, $\mathcal{N}_\sigma \colon$ **SymProduo** $\rightleftharpoons$ **nSymProduo** $\colon \mathcal{U}$. That is, $\mathcal{N}_\sigma\mathbb{V}$ is the free symmetric produoidal category over $\mathbb{V}$.*

Proof. See Appendix, Theorem E.11. □

6 Monoidal Context: Mixing $\triangleleft$ and $\otimes$ by normalization

Monoidal contexts formalize the notion of an incomplete morphism in a monoidal category. The category of monoidal contexts will have a rich algebraic structure: we shall be able to still compose contexts sequentially and in parallel and, at the same time, we shall be able to fill a context using another monoidal context. Perhaps surprisingly, then, the category of monoidal contexts is not even monoidal.

We justify this apparent contradiction in terms of profunctorial structure: the category is not monoidal, but it does have two promonoidal structures that precisely represent sequential and parallel composition. These structures form a normal produoidal category. In fact, we show it to be the normalization of the produoidal category of spliced monoidal arrows.

This section constructs explicitly the normal produoidal category of monoidal contexts.

6.1 The Category of Monoidal Contexts

A monoidal context, $\mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)$, represents a process from A to B with a hole admitting a process from X to Y. In this sense, monoidal contexts are similar to spliced monoidal arrows. The difference with spliced monoidal arrows is that monoidal contexts allow for communication to happen to the left and to the right of this hole.

Definition 6.1 (Monoidal context)**.** Let $(\mathbb{C},\otimes,I)$ be a monoidal category. *Monoidal contexts* are the elements of the following profunctor,

$$\mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2)\diamond\mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2;B).$$

In other words, a *monoidal context* from A to B, *with a hole* from X to Y, is an equivalence class consisting of a pair of objects $M,N\in\mathbb{C}_{\text{obj}}$ and a pair of morphisms $f\in\mathbb{C}(A;M\otimes X\otimes N)$ and $g\in\mathbb{C}(M\otimes Y\otimes N;B)$, quotiented by dinaturality of M and N (Figure 13). We write monoidal contexts as

$$(f\,⨟\,(\mathrm{id}_M\otimes\blacksquare\otimes\mathrm{id}_N)\,⨟\,g)\in\mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right).$$

In this notation, dinaturality explicitly means that

$$\begin{aligned}&(f\,⨟\,(m\otimes\mathrm{id}_X\otimes n)\,⨟\,(\mathrm{id}_W\otimes\blacksquare\otimes\mathrm{id}_H)\,⨟\,g) \quad =\\ &(f\,⨟\,(\mathrm{id}_M\otimes\blacksquare\otimes\mathrm{id}_N)\,⨟\,(m\otimes\mathrm{id}_Y\otimes n)\,⨟\,g).\end{aligned}$$

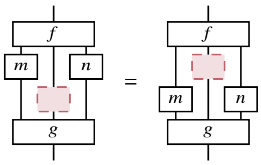


Fig. 13: Dinaturality for monoidal contexts.

Proposition 6.2. *Monoidal contexts form a category.*

Proof. We define composition of monoidal contexts by the following formula (illustrated in Figure 29, iii).

$$\begin{aligned}&(f\,⨟\,(\mathrm{id}_M\otimes\blacksquare\otimes\mathrm{id}_N)\,⨟\,g)\prec(h\,⨟\,(\mathrm{id}_{M'}\otimes\blacksquare\otimes\mathrm{id}_{N'})\,⨟\,k) \quad =\\ &f\,⨟\,(\mathrm{id}_M\otimes h\otimes\mathrm{id}_N)\,⨟\,(\mathrm{id}_{M\otimes M'}\otimes\blacksquare\otimes\mathrm{id}_{N\otimes N'})\\ &\quad ⨟\,(\mathrm{id}_M\otimes k\otimes\mathrm{id}_N)\,⨟\,g\end{aligned}$$

For each pair of objects, we define the identity monoidal context as $\mathrm{id}_A\,⨟\,\blacksquare\,⨟\,\mathrm{id}_B$ (illustrated in Figure 29, ii). We check that this composition is associative and unital in the Appendix, Proposition F.3. □

Remark 6.3. Even when we introduce $(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})$ as a piece of suggestive notation, we can still write $(g\otimes\blacksquare\otimes h)$ unambiguously, because of dinaturality,

$$(g\otimes\mathrm{id}\otimes h)\,⨟\,(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})=(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\,⨟\,(g\otimes\mathrm{id}\otimes h).$$

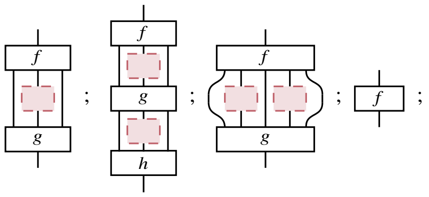


Fig. 14: Morphisms, sequential and parallel splits, and units of the splice monoidal arrow produoidal category.

6.2 The Normal Produoidal Algebra of Monoidal Contexts

Let us endow monoidal contexts with their normal produoidal structure.

Definition 6.4. The category of monoidal contexts, $\mathcal{M}\mathbb{C}$, has as objects pairs of objects of $\mathbb{C}$. We use the following profunctors to define sequential splits, parallel splits, units and morphisms.

$$\begin{aligned}\mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)&=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2)\diamond\mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2;B);\\ \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\triangleleft\begin{smallmatrix}X'\\Y'\end{smallmatrix}\right)&=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2)\diamond\\ &\quad\ \mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2;\bullet_3\otimes X'\otimes\bullet_4)\diamond\\ &\quad\ \mathbb{C}(\bullet_3\otimes Y'\otimes\bullet_4;B);\\ \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\otimes\begin{smallmatrix}X'\\Y'\end{smallmatrix}\right)&=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2\otimes X'\otimes\bullet_3)\diamond\\ &\quad\ \mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2\otimes Y'\otimes\bullet_3;B);\\ \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};N\right)&=\mathbb{C}(A;B).\end{aligned}$$

In other words, sequential splits are triples of arrows $f\colon A\to M\otimes X\otimes N$, $g\colon M\otimes Y\otimes N\to M'\otimes X'\otimes N'$ and $h\colon M'\otimes Y'\otimes N'\to B$, quotiented by dinaturality of M,M',N,N'. Parallel splits are pairs of arrows $f\colon A\to M\otimes X\otimes N\otimes X'\otimes O$ and $g\colon M\otimes Y\otimes N\otimes Y'\otimes O\to B$, quotiented by dinaturality of M,N,O. Units are simply arrows $f\colon A\to B$. In summary, we have

morphisms,	$f\,⨟\,(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\,⨟\,g$
sequential splits,	$f\,⨟\,(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\,⨟\,g\,⨟\,(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\,⨟\,h;$
parallel splits,	$f\,⨟\,(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\,⨟\,g;$
sequential units,	$f.$

Dinaturality for sequential splits and parallel splits is depicted the Appendix, Figures 30 and 31.

Proposition 6.5. *The category of monoidal contexts forms a normal produoidal category with its units, sequential and parallel splits.*

Proof. See Appendix, Proposition F.4. □

Theorem 6.6. *Monoidal contexts are the free normalization of the cofree produoidal category over a category. In other words, monoidal contexts are the normalization of spliced monoidal arrows,* $\mathcal{NT}\mathbb{C}\cong\mathcal{M}\mathbb{C}$.

Proof. See Appendix, Theorem F.12. □

7 Monoidal Lenses

Monoidal lenses are *symmetric* monoidal contexts. Again, the category of monoidal lenses has a rich algebraic structure; and again, most of this structure exists only virtually in terms of profunctors. In this case, though, the monoidal tensor *does* indeed exist: contrary to monoidal contexts, monoidal lenses form also a monoidal category.

This is perhaps why applications of monoidal lenses have grown popular in recent years [Ril18], with applications in decision theory [GHWZ18], supervised learning [CGG+22], [FJ19] and most notably in functional data accessing [Kme12], [PGW17], [BG18], [CEG+20]. The promonoidal structure of optics was ignored, even when, after now identifying for the first time its relation to the monoidal structure of optics, we argue that it could be potentially useful in these applications: e.g. in multi-stage decision problems, or in multi-stage data accessors.

This section explicitly constructs the normal symmetric produoidal category of *monoidal lenses*. We describe it for the first time by a universal property: it is the free symmetric normalization of the cofree produoidal category.

7.1 The Category of Monoidal Lenses

A monoidal lens of type $\mathcal{L}\mathbb{C}(\frac{A}{B}, \frac{X}{Y})$ represents a process in a symmetric monoidal category with a hole admitting a process from X to Y.

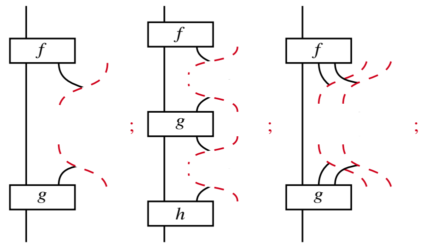


Fig. 15: Generic monoidal lens, sequential and parallel split.

Definition 7.1 (Monoidal Lens). Let $(\mathbb{C}, \otimes, I)$ be a symmetric monoidal category. *Monoidal lenses* are the elements of the following profunctor,

$$\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y}\right) = \mathbb{C}(A; \bullet \otimes X) \diamond \mathbb{C}(\bullet \otimes Y; B).$$

In other words, a *monoidal lens* from A to B, *with a hole* from X to Y, is an equivalence class consisting of a pair of objects $M, N \in \mathbb{C}_{\text{obj}}$ and a pair of morphisms $f \in \mathbb{C}(A; M \otimes X)$ and $g \in \mathbb{C}(M \otimes Y; B)$, quotiented by dinaturality of M. We write monoidal lenses as

$$f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare) \mathbin{⨟} g \in \mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y}\right).$$

Proposition 7.2. *Monoidal lenses form a normal symmetric produoidal category with the following morphisms, units, sequential and parallel splits.*

$$\begin{aligned}
\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y}\right) &= \mathbb{C}(A; \bullet \otimes X) \diamond \mathbb{C}(\bullet \otimes Y; B);\\
\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; N\right) &= \mathbb{C}(A; B);\\
\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y} \triangleleft \tfrac{X'}{Y'}\right) &= \mathbb{C}(A; \bullet_1 \otimes X) \diamond\\
&\quad \mathbb{C}(\bullet_1 \otimes Y; \bullet_2 \otimes X') \diamond \mathbb{C}(\bullet_2 \otimes Y'; B);\\
\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y} \otimes \tfrac{X'}{Y'}\right) &= \mathbb{C}(A; \bullet_1 \otimes X \otimes X') \diamond \mathbb{C}(\bullet_1 \otimes Y \otimes Y'; B).
\end{aligned}$$

Proof. See Appendix, Proposition G.1. □

Theorem 7.3. *Monoidal lenses are the free symmetric normalization of the cofree symmetric produoidal category over a monoidal category.*

Proof. See Appendix, Theorem G.9. □

Remark 7.4 (Representable parallel structure). The parallel splitting structure of monoidal lenses is representable,

$$\mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X}{Y} \otimes \tfrac{X'}{Y'}\right) = \mathcal{L}\mathbb{C}\left(\tfrac{A}{B} ; \tfrac{X \otimes X'}{Y \otimes Y'}\right).$$

Lenses over a symmetric monoidal category are known to be monoidal [Ril18], [Hed17], but it remained unexplained why a similar structure was not present in non-symmetric lenses. The contradiction can be solved by noting that both symmetric and non-symmetric lenses are indeed *promonoidal*, even if only symmetric optics provide a representable tensor.

Remark 7.5 (Session notation for lenses). We will write $!A = \left(\tfrac{A}{I}\right)$ and $?B = \left(\tfrac{I}{B}\right)$ for the objects of the produoidal category of lenses that have a monoidal unit as one of its objects. These are enough to express all objects because $!A \otimes ?B = \left(\tfrac{A}{B}\right)$; and, moreover, they satisfy the following properties definitionally.

$$\begin{aligned}
&\mathbb{C}(\bullet; ?A \triangleleft ?B) \cong \mathbb{C}(\bullet; ?A \otimes ?B); && !(A \otimes B) = !A \otimes !B;\\
&\mathbb{C}(\bullet; !A \triangleleft !B) \cong \mathbb{C}(\bullet; !A \otimes !B); && ?(A \otimes B) = ?A \otimes ?B;\\
&\mathbb{C}(\bullet; !A \triangleleft ?B) \cong \mathbb{C}(\bullet; !A \otimes ?B).
\end{aligned}$$

Proposition 7.6. *Let $(\mathbb{C}, \otimes, I)$ be a symmetric monoidal category. There exist monoidal functors $(!) \colon \mathbb{C} \to \mathcal{L}\mathbb{C}$ and $(?) \colon \mathbb{C}^{op} \to \mathcal{L}\mathbb{C}$.*

Proof. See Appendix, Proposition G.7. □

7.2 Protocol Analysis

Let us go back to our running example (Figure 1). We can now declare that the client and server have the following types, representing the order in which they communicate,

$$\begin{aligned}
💻 &\in \mathcal{L}\mathbb{C}\left(\tfrac{\text{Client}}{\text{Client}} ; !\text{Msg} \triangleleft ?\text{Msg} \triangleleft !\text{Msg}\right);\\
🗄 &\in \mathcal{L}\mathbb{C}\left(\tfrac{\text{Server}}{\text{Server}} ; ?\text{Msg} \triangleleft !\text{Msg} \triangleleft ?\text{Msg}\right).
\end{aligned}$$

Moreover, we can use the duoidal algebra to compose them. Indeed, tensoring client and server, we get the following codomain type,

$$(!\text{Msg} \triangleleft ?\text{Msg} \triangleleft !\text{Msg}) \otimes (?\text{Msg} \triangleleft !\text{Msg} \triangleleft ?\text{Msg}).$$

We then apply the laxators to mix inputs and outputs, obtaining

$$(\mathbf{!}\mathrm{Msg} \otimes \mathbf{?}\mathrm{Msg}) \triangleleft (\mathbf{?}\mathrm{Msg} \otimes \mathbf{!}\mathrm{Msg}) \triangleleft (\mathbf{!}\mathrm{Msg} \otimes \mathbf{?}\mathrm{Msg}),$$

and we finally apply the unitors to fill the communication holes with noisy channels.

$$\psi_2 \left(💻 \otimes 🗄 \right) \prec^3_\lambda \mathtt{NOISE}^3 \in \mathcal{L}\mathbb{C} \binom{\mathrm{Client}{\otimes}\mathrm{Server}}{\mathrm{Client}{\otimes}\mathrm{Server}} .$$

We end up obtaining the protocol as a single morphism Client ⊗ Server → Client ⊗ Server in whatever category we are using to program. Assuming the category of finite stochastic maps, this single morphism represents the distribution over the possible outcomes of the protocol. Finally, by dinaturality, we can reason over independent parts of the protocol.

Proposition 7.7. *Let* (💻) = $(\mathit{SYN} ⨟ (\mathrm{id}\otimes\blacksquare) ⨟ \mathit{ACK} ⨟ (\mathrm{id}\otimes\blacksquare))$. *The equalities in Figure 1 are a consequence of the dinaturality of a monoidal lens.*

Proof. We recognize the diagram in Figure 1 as representing the elements in the following equation.

$$\begin{aligned}
&\mathtt{SYN} ⨟ (\mathrm{id} \otimes \blacksquare) ⨟ \mathtt{ACK} ⨟ (\mathrm{id} \otimes \blacksquare) && = \\
&\mathtt{SYN}^* ⨟ (\mathtt{PRJ} \otimes \mathrm{id}) ⨟ \blacksquare ⨟ \mathtt{ACK} ⨟ (\mathrm{id} \otimes \blacksquare) && = \\
&\mathtt{SYN}^* ⨟ (\mathrm{id} \otimes \blacksquare) ⨟ (\mathtt{PRJ} \otimes \mathrm{id}) ⨟ \mathtt{ACK} ⨟ (\mathrm{id} \otimes \blacksquare) && = \\
&\mathtt{SYN}^* ⨟ (\mathrm{id} \otimes \blacksquare) ⨟ \mathtt{ACK}^* ⨟ (\mathrm{id} \otimes \blacksquare).
\end{aligned}$$

In the same way we would apply the *interchange law* in completed morphisms, we have applied dinaturality over PRJ. □

7.3 Cartesian Lenses

We have worked in full generality, but cartesian lenses are particularly important to applications in game theory [GHWZ18] and functional programming [Kme12], [PGW17]. We introduce their newly constructed produoidal structure.

Proposition 7.8 (Cartesian Lenses)**.** *Let* $(\mathbb{C}, \cdot, 1)$ *be a cartesian monoidal category. Its produoidal category of lenses is given by the following profunctors.*

$$\begin{aligned}
\mathcal{L}\mathbb{C} \left(\tbinom{A}{B}; \tbinom{X}{Y}\right) &\cong \mathbb{C}(A; X) \times \mathbb{C}(AY; B), \\
\mathcal{L}\mathbb{C} \left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) &\cong \mathbb{C}(A; X) \times \mathbb{C}(AY; X') \times \mathbb{C}(AYY'; B), \\
\mathcal{L}\mathbb{C} \left(\tbinom{A}{B}; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) &\cong \mathbb{C}(A; XX') \times \mathbb{C}(AYY'; B), \\
\mathcal{L}\mathbb{C} \tbinom{A}{B} &\cong \mathbb{C}(A; B).
\end{aligned}$$

Proof. See Appendix, Proposition G.8. □

8 Conclusions

Monoidal contexts are an algebra of incomplete processes, commonly generalizing lenses [Ril18] and spliced arrows [MZ22]. In the same way that the π-calculus allows input/output channels of an abstract model of computation, monoidal contexts allow input/output communication on arbitrary theories of processes, such as stochastic or partial functions, quantum processes or relational queries.

Monoidal contexts form a normal produoidal category: a highly structured and rich categorical algebra. Moreover, they are the universal such algebra on a monoidal category. This is good news for applications: the literature on concurrency is rich in frameworks; but the lack of *canonicity* may get us confused when trying to choose, design, or compare among them, as Abramsky [Abr05] has pointed out. Precisely characterizing the universal property of a model addresses this concern. This is also good news for the category theorist: not only is this an example shedding light on a relatively obscure structure; it is a paradigmatic such one.

We rely on two mathematical ideas: *monoidal* and *duoidal* categories on one hand, and *dinaturality* and *profunctorial* structures on the other. *Monoidal categories*, which could be accidentally dismissed as a toy version of cartesian categories, show that their string diagrams can bootstrap our conceptual understanding of new fundamental process structures, while keeping an abstraction over their implementation that cartesian categories cannot afford. Duoidal categories are such an example: starting to appear insistently in computer science [SS22], [HS23], they capture the posetal structure of process dependency and communication. *Dinaturality*, virtual structures and profunctors, even if sometimes judged arcane, show again that they can canonically capture a notion as concrete as process composition.

8.1 Further Work

Dependencies. Shapiro and Spivak [SS22] prove that normal symmetric duoidal categories with certain limits additionally have the structure of *dependence categories*: they can not only express dependence structures generated by $(\triangleleft)$ and $(\otimes)$, but arbitrary poset-mediated dependence structures. Produoidal categories are better behaved: the limits always exist, and we only require these are preserved by the coend.

Proposition 8.1. *Let* $\mathbb{V}$ *be a normal and* $\otimes$*-symmetric produoidal category with coends over* $\mathbb{V}$ *commuting with finite connected limits. Then,* $[\mathbb{V}^{\mathrm{op}}, \mathbf{Set}]$ *is a dependence category in the sense of Shapiro and Spivak [SS22].*

Proof sketch. See Appendix, Theorem H.1. □

Weakening dependence categories in this way combines the ideas of Shapiro and Spivak [SS22] with those of Hefford and Kissinger [HK22], who employ virtual objects to deal with the non-existence of tensor products in models of spacetime.

Language theory. Melliès and Zeilberger [MZ22] used a multicategorical form of splice-contour adjunction (Remark 3.3) to give a novel proof of the Chomsky-Schützenberger representation theorem, generalized to context-free languages in categories. Our produoidal splice-contour adjunction (Section 4), combined with recent work on languages of morphisms in monoidal categories [ES22] opens the way for a vertical categorification of the Chomsky-Schützenberger theorem, which we plan to elaborate in future work.

String diagrams for concurrency. Nester et al. [Nes23], [BNR22] have recently introduced an alternative description of lenses in terms of *proarrow equipments*, which have a good 2-dimensional syntax [Mye16] we can use for send/receive types $(\mathbf{!}/\mathbf{?})$. We have shown how this structure arises universally in symmetric monoidal categories. It remains as further work to determine a good 2-dimensional syntax for concurrent programs with *iteration* and *internal/external choice*.

9 Acknowledgements

We thank Pawel Sobocinski, Fosco Loregian, Chad Nester and David Spivak for discussion.

Matt Earnshaw and Mario Román were supported by the European Social Fund Estonian IT Academy research measure (project 2014-2020.4.05.19-0001). James Hefford is supported by University College London and the EPSRC [grant number EP/L015242/1].

References

[Abr05] Samson Abramsky. What are the fundamental structures of concurrency?: We still don't know! In Luca Aceto and Andrew D. Gordon, editors, *Proceedings of the Workshop "Essays on Algebraic Process Calculi", APC 25, Bertinoro, Italy, August 1-5, 2005*, volume 162 of *Electronic Notes in Theoretical Computer Science*, pages 37–41. Elsevier, 2005.

[AC09] Samson Abramsky and Bob Coecke. Categorical quantum mechanics. In Kurt Engesser, Dov M. Gabbay, and Daniel Lehmann, editors, *Handbook of Quantum Logic and Quantum Structures*, pages 261–323. Elsevier, Amsterdam, 2009.

[AHLF18] Marcelo Aguiar, Mariana Haim, and Ignacio López Franco. Monads on higher monoidal categories. *Applied Categorical Structures*, 26(3):413–458, Jun 2018.

[AM10] Marcelo Aguiar and Swapneel Arvind Mahajan. *Monoidal functors, species and Hopf algebras*, volume 29. American Mathematical Society Providence, RI, 2010.

[BDSPV15] Bruce Bartlett, Christopher L. Douglas, Christopher J. Schommer-Pries, and Jamie Vicary. Modular categories as representations of the 3-dimensional bordism 2-category, 2015.

[Bén00] Jean Bénabou. Distributors at work. *Lecture notes written by Thomas Streicher*, 11, 2000.

[BG18] Guillaume Boisseau and Jeremy Gibbons. What you needa know about yoneda: Profunctor optics and the yoneda lemma (functional pearl). *Proceedings of the ACM on Programming Languages*, 2(ICFP):1–27, 2018.

[BNR22] Guillaume Boisseau, Chad Nester, and Mario Román. Cornering optics. volume abs/2205.00842, 2022.

[BPSZ19] Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Graphical affine algebra. In *34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019, Vancouver, BC, Canada, June 24-27, 2019*, pages 1–12. IEEE, 2019.

[BS13] Thomas Booker and Ross Street. Tannaka duality and convolution for duoidal categories. *Theory and Applications of Categories*, 28(6):166–205, 2013.

[BSS18] Filippo Bonchi, Jens Seeber, and Pawel Sobocinski. Graphical conjunctive queries. In Dan R. Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic, CSL 2018, September 4-7, 2018, Birmingham, UK*, volume 119 of *LIPIcs*, pages 13:1–13:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[CEG+20] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor optics, a categorical update. *CoRR*, abs/2001.07488, 2020.

[CFS16] Bob Coecke, Tobias Fritz, and Robert W. Spekkens. A mathematical theory of resources. *Inf. Comput.*, 250:59–86, 2016.

[CGG+22] Geoffrey S. H. Cruttwell, Bruno Gavranović, Neil Ghani, Paul Wilson, and Fabio Zanasi. Categorical foundations of gradient-based learning. In *European Symposium on Programming*, pages 1–28. Springer, Cham, 2022.

[CJ19] Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019.

[CL02] J. Robin B. Cockett and Stephen Lack. Restriction categories I: categories of partial maps. *Theoretical Computer Science*, 270(1-2):223–259, 2002.

[CP09] J. Robin B. Cockett and Craig A. Pastro. The logic of message-passing. *Sci. Comput. Program.*, 74(8):498–533, 2009.

[CS97] J. Robin B. Cockett and Robert A. G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

[CS10] G.S.H. Cruttwell and Michael A. Shulman. A unified framework for generalized multicategories. *Theory and Applications of Categories*, 24:580–655, 2010.

[Day70a] Brian Day. *Construction of Biclosed Categories*. PhD thesis, University of New South Wales, 1970.

[Day70b] Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, volume 137, pages 1–38, Berlin, Heidelberg, 1970. Springer Berlin Heidelberg.

[DLdFR22] 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*, LICS '22, New York, NY, USA, 2022. Association for Computing Machinery.

[ES22] Matthew Earnshaw and Pawel Sobociński. Regular Monoidal Languages. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 44:1–44:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FGM+07] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 29(3):17–es, 2007.

[FJ19] Brendan Fong and Michael Johnson. Lenses and learners. *arXiv preprint arXiv:1903.03671*, 2019.

[Fri20] Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020.

[GF16] Richard Garner and Ignacio López Franco. Commutativity. *Journal of Pure and Applied Algebra*, 220(5):1707–1751, 2016.

[GH99] Simon J. Gay and Malcolm Hole. Types and subtypes for client-server interactions. In S. Doaitse Swierstra, editor, *Programming Languages and Systems, 8th European Symposium on Programming, ESOP'99, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS'99, Amsterdam, The Netherlands, 22-28 March, 1999, Proceedings*, volume 1576 of *Lecture Notes in Computer Science*, pages 74–90. Springer, 1999.

[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.

[Gui80] René Guitart. Tenseurs et machines. *Cahiers de topologie et géométrie différentielle catégoriques*, 21(1):5–62, 1980.

[Has97] Masahito Hasegawa. *Models of sharing graphs: a categorical semantics of let and letrec*. PhD thesis, University of Edinburgh, UK, 1997.

[HC22] James Hefford and Cole Comfort. Coend optics for quantum combs. *arXiv preprint arXiv:2205.09027*, 2022.

[Hed17] Jules Hedges. Coherence for lenses and open games. *arXiv preprint arXiv:1704.02230*, 2017.

[HK22] James Hefford and Aleks Kissinger. On the pre- and promonoidal structure of spacetime. *arXiv preprint arXiv.2206.09678*, 2022.

[HLV+16] Hans Hüttel, Ivan Lanese, Vasco T. Vasconcelos, Luís Caires, Marco Carbone, Pierre-Malo Deniélou, Dimitris Mostrous, Luca Padovani, António Ravara, Emilio Tuosto, Hugo Torres Vieira, and Gianluigi Zavattaro. Foundations of session types and behavioural contracts. *ACM Comput. Surv.*, 49(1):3:1–3:36, 2016.

[Hon93] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer, 1993.

[HS23] Chris Heunen and Jesse Sigal. Duoidally enriched Freyd categories. *arXiv preprint arXiv:2301.05162*, 2023.

[Hug00] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000.

[HYC08] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In George C. Necula and Philip Wadler, editors, *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008*, pages 273–284. ACM, 2008.

[JRW12] Michael Johnson, Robert Rosebrugh, and Richard J. Wood. Lenses, fibrations and universal translations. *Mathematical structures in computer science*, 22(1):25–42, 2012.

[JS91] André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991.

[Kme12] Edward Kmett. lens library, version 4.16. *Hackage https://hackage. haskell. org/package/lens-4.16*, 2018, 2012.

[KPT96] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. In Hans-Juergen Boehm and Guy L. Steele Jr., editors, *Conference Record of POPL'96: The 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Papers Presented at the Symposium, St. Petersburg Beach, Florida, USA, January 21-24, 1996*, pages 358–371. ACM Press, 1996.

[KU17] Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20-23, 2017*, pages 1–12. IEEE Computer Society, 2017.

[Lor21] Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021.

[Mac78] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978.

[Mog91] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[Mye16] David Jaz Myers. String diagrams for double categories and equipments, 2016.

[MZ22] Paul-André Melliès and Noam Zeilberger. Parsing as a Lifting Problem and the Chomsky-Schützenberger Representation Theorem. In *MFPS 2022-38th conference on Mathematical Foundations for Programming Semantics*, 2022.

[Nes23] Chad Nester. Concurrent Process Histories and Resource Transducers. *Logical Methods in Computer Science*, Volume 19, Issue 1, January 2023.

[NS22] Nelson Niu and David I. Spivak. Polynomial functors: A general theory of interaction. *In preparation*, 2022.

[Pat01] Ross Paterson. A new notation for arrows. In Benjamin C. Pierce, editor, *Proceedings of the Sixth ACM SIGPLAN International Conference on Functional Programming (ICFP '01), Firenze (Florence), Italy, September 3-5, 2001*, pages 229–240. ACM, 2001.

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *Art Sci. Eng. Program.*, 1(2):7, 2017.

[Pos81] J. Postel. Transmission control protocol. RFC 793, RFC Editor, 9 1981.

[PS93] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings of the Eighth Annual Symposium on Logic in Computer Science (LICS '93), Montreal, Canada, June 19-23, 1993*, pages 376–385. IEEE Computer Society, 1993.

[PS07] Craig Pastro and Ross Street. Doubles for Monoidal Categories. *arXiv preprint arXiv:0711.1859*, 2007.

[PSV21] Evan Patterson, David I. Spivak, and Dmitry Vagner. Wiring diagrams as normal forms for computing in symmetric monoidal categories. *Electronic Proceedings in Theoretical Computer Science*, page 49–64, Feb 2021.

[Ril18] Mitchell Riley. Categories of Optics. *arXiv preprint arXiv:1809.00738*, 2018.

[Rom20] Mario Román. Comb Diagrams for Discrete-Time Feedback. *CoRR*, abs/2003.06214, 2020.

[Rom21] Mario Román. Open diagrams via coend calculus. *Electronic Proceedings in Theoretical Computer Science*, 333:65–78, Feb 2021.

[Rom22] Mario Román. Promonads and string diagrams for effectful categories. In *ACT '22: Applied Category Theory, Glasgow, United Kingdom, 18 - 22 July, 2022*, volume abs/2205.07664, 2022.

[Shu16] Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016.

[Shu17] Michael Shulman. Duoidal category (nlab entry), section 2., 2017. https://ncatlab.org/nlab/show/duoidal+category, Last accessed on 2022-12-14.

[Spi13] David I. Spivak. The operad of wiring diagrams: formalizing a graphical language for databases, recursion, and plug-and-play circuits. *CoRR*, abs/1305.0297, 2013.

[SS22] Brandon T. Shapiro and David I. Spivak. Duoidal structures for compositional dependence. *arXiv preprint arXiv:2210.01962*, 2022.

[SSV20] Patrick Schultz, David I. Spivak, and Christina Vasilakopoulou. Dynamical systems and sheaves. *Applied Categorical Structures*, 28(1):1–57, 2020.

[SW01] Davide Sangiorgi and David Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.

[VC22] André Videla and Matteo Capucci. Lenses for composable servers. *CoRR*, abs/2203.15633, 2022.

Appendix A
Introduction

A.1 Three Way handshake Implementation

The following can be interpreted as pseudocode using the linear type theory of symmetric monoidal categories [Shu16]. The type theory of symmetric monoidal categories (Section A.1) uses declarations such as `(x , y) <- f(a, b, c)` to represent morphisms such as $f\colon A \otimes B \otimes C \to X \otimes Y$.

$$\textsc{Gen}\quad \frac{f \in \mathcal{G}(A_1, \dots, A_n; B) \qquad \Gamma_1 \vdash x_1 : A_1 \dots \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1, \dots, \Gamma_n) \vdash f(x_1, \dots, x_n) : B}$$

$$\textsc{Pair}\quad \frac{\Gamma_1 \vdash x_1 : A_1 \ \dots\ \Gamma_n \vdash x_n : A_n}{\mathrm{Shuf}(\Gamma_1, \dots, \Gamma_n) \vdash [x_1, ..., x_n] : A_1 \otimes ... \otimes A_n} \qquad \textsc{Var}\quad \frac{}{x : A \vdash x : A}$$

$$\textsc{Split}\quad \frac{\Delta \vdash m : A_1 \otimes \cdots \otimes A_n \qquad \Gamma, x_1 : A_1, \dots, x_n : A_n \vdash z : C}{\mathrm{Shuf}(\Gamma, \Delta) \vdash [x_1, \dots, x_n] \ \leftarrow\ m\ ;\ z : C}$$

Fig. 16: Type theory of symmetric monoidal categories [Shu16].

We can interpret pseudocode as talking about the type theory of monoidal categories. Usually, we will need some extra structure: such as `if-then-else` or explicit functions. It has been found in programming that a good level of concreteness for monoidal categories is given by the Kleisli categories of commutative monads, sometimes abstracted by Freyd categories [Mog91], [Hug00], see [Rom22] for a comparison with plain monoidal categories and string diagrams. For convenience, we assume this setting in the following code, but note that it is not strictly necessary, and that a type-theoretic implementation of monoidal categories would work just the same.

The following code inspired by Haskell's do-notation [Hug00] and it has been tested in the Glasgow Haskell Compiler, version 9.2.5.

```
syn :: Client ~> (Client, Syn, Ack)
syn(client) = do
  client <- random
  return (client, client, 0)

synack :: (Syn, Ack, Server) ~> (Syn, Ack, Server)
synack(syn, ack, server) = do
  server <- random
  return (if syn == 0 then (0,0,0) else (server, ack+1, server))

noise :: Noise -> (Syn, Ack) ~> (Syn, Ack)
noise k (syn,ack) = do
  noise <- binomial k
  return (if noise then (0,0) else (syn,ack))

ack :: (Client, Syn, Ack) ~> (Client, Syn, Ack)
ack(client, syn, ack) = do
  return (if client+1 /= ack then (0,0,0) else (client+1, syn+1, client))
  receive :: (Syn, Ack, Server) ~> Server
  receive(syn, ack, server) = do
    return (if server+1 /= ack then 0 else server)
```

We can use the produoidal category of lenses to provide a modular description of this protocol.

The programmer will not need to know about produoidal categories: they will be able to define *splits* of a process; they will be able to read the type of the *split* in terms of the send-receive steps of the protocol; they will be able to combine them, and the typechecker should produce an error whenever dinaturality is not respected. In fact, in the following code, naively combining client and server in a way that does not preserve dinaturality will produce a type error because GHC will not be able to match the types. We present the description of the protocol, encoding send/receive types.

```
protocol ::
  Split (Kleisli Distribution) Client Client
    (Syn, Ack) -- !
    (Syn, Ack) -- ?
    (Syn, Ack) -- !
    ()         -- ?
  -> Split (Kleisli Distribution) Server Server
    ()         -- !
    (Syn, Ack) -- ?
    (Syn, Ack) -- !
    (Syn, Ack) -- ?
  -> (Client, Server) ~> (Client, Server)
protocol
  (Split (Kleisli client1) (Kleisli client2) (Kleisli client3))
  (Split (Kleisli server1) (Kleisli server2) (Kleisli server3))
  (client , server) = do
    (server, ())    <- server1(server)
    (client, (s,a)) <- client1(client)
    (s, a) <- noise 0.1 (s,a)
    (server, (s,a)) <- server2(server, (s,a))
    (s, a) <- noise 0.1 (s,a)
    (client, (s,a)) <- client2(client, (s,a))
    (s, a) <- noise 0.1 (s,a)
    (server)        <- server3(server, (s,a))
    (client)        <- client3(client, ())
    return (client, server)
```

The following Figure 17 and Figure 18 show the separate Haskell code for the client and server modules.

```
client :: Split (Kleisli Distribution) Client Client
  (Syn, Ack) -- !
  (Syn, Ack) -- ?
  (Syn, Ack) -- !
  ()         -- ?
client = Split {

    -- Part 1: Send a SYN message.
    part1 = Kleisli $ \client -> do
        client <- pure 10
        return (client, (client, 0))

    -- Part 2: Receive ACK, send ACK.
    , part2 = Kleisli $ \(client, (syn, ack)) -> do
        return (if client+1 /= syn then (0,(0,0)) else (client, (client+1, ack+1)))

    -- Part 3: Close protocol.
    , part3 = Kleisli $ \(client, ()) -> do
        return client

    }
```

Fig. 17: Haskell code for the client module.

```
server :: Split (Kleisli Distribution) Server Server
  ()         -- send    ==>
  (Syn, Ack) -- receive <==
  (Syn, Ack) -- send    ==>
  (Syn, Ack) -- receive <==
server = Split

    -- Part 1: Open protocol.
    { part1 = Kleisli $ \server -> do
        return (server, ())

    -- Part 2: Receive SYN and send ACK.
    , part2 = Kleisli $ \(server, (syn, ack)) -> do
        server <- pure 20
        return (if syn == 0 then (0,(0,0)) else (server, (syn+1, server)))

    -- Part 3: Receive ACK.
    , part3 = Kleisli $ \(server, (syn, ack)) -> do
        return (if server+1 /= ack then 0 else server)
    }
```

Fig. 18: Code for the server module.

```
data Split c a b x y s t where
    Split :: { part1 :: c a       (m , x)
             , part2 :: c (m , y) (n , s)
             , part3 :: c (n , t) b
             } -> Split c a b x y s t

data Unit c a b where
    Unit :: { unit :: c a b } -> Unit c a b

data Context c a b x y where
    Context :: { partA :: c a (m , x)
               , partB :: c (m , y) (m , b)
               } -> Context c a b x y

type (a ~> b) = (a -> Distribution b)
```

Fig. 19: Code describing the profunctors of monoidal lenses.

Appendix B
Profunctors and virtual structures

Definition B.1. A *profunctor* $(P, \prec, \succ)$ between two categories $\mathbb{A}$ and $\mathbb{B}$, written $P(\bullet;\bullet)\colon \mathbb{A} \bullet\!\!-\!\!\circ\ \mathbb{B}$, is a family of sets $P(B;A)$ indexed by objects $\mathbb{A}$ and $\mathbb{B}$, and endowed with jointly functorial left and right actions of the morphisms of $\mathbb{A}$ and $\mathbb{B}$, respectively [Bén00], [Lor21].

Explicitly, the types of these actions are $(\succ)\colon \mathbb{B}(B', B) \times P(B, A) \to P(B', A)$, and $(\prec)\colon P(B, A) \times \mathbb{A}(A, A') \to P(B, A')$. These must

- satisfy compatibility, $(f \succ p) \prec g = f \succ (p \prec g)$,
- preserve identities, $id \succ p = p$, and $p \prec id = p$,
- and preserve compositions, $(p \prec f) \prec g = p \prec (f \mathbin{⨾} g)$ and $f \succ (g \succ p) = (f \mathbin{⨾} g) \succ p$.

Remark B.2. More succinctly, a profunctor $P\colon \mathbb{A} \bullet\!\!-\!\!\circ\ \mathbb{B}$ is a functor $P\colon \mathbb{B}^{op} \times \mathbb{A} \to \mathbf{Set}$. Analogously, a profunctor $P\colon \mathbb{A} \circ\!\!-\!\!\bullet\ \mathbb{B}$ is a functor $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathbf{Set}$, or a profunctor $P\colon \mathbb{B} \bullet\!\!-\!\!\circ\ \mathbb{A}$.[3] When presented as a family of sets with a pair of actions, profunctors are sometimes called bimodules.

Theorem B.3 (Yoneda isomorphisms). *Let $\mathbb{C}$ be a category. There exist bijections between the following sets defined by coends. These are natural in the copresheaf $F\colon \mathbb{C} \to \mathbf{Set}$, the presheaf $G\colon \mathbb{C}^{op} \to \mathbf{Set}$ and $A \in \mathbb{C}$,*

$$\int^X \mathbb{C}(X;A) \times F(X) \overset{y_1}{\cong} F(A); \qquad \int^X \mathbb{C}(A;X) \times G(X) \overset{y_2}{\cong} G(A);$$

and they are defined by $y_1(f \mid \alpha) = F(f)(\alpha)$ and $y_2(g \mid \beta) = G(g)(\beta)$. These are called Yoneda reductions *or* Yoneda isomorphisms*, because they appear in the proof of Yoneda lemma. Moreover, any formal diagram constructed out of these reductions, products, identities and compositions commutes.*

B.1 Promonads

Definition B.4. A *promonad* $(P, \star, ^\circ)$ over a category $\mathbb{C}$ is a profunctor $P\colon \mathbb{C} \circ\!\!-\!\!\bullet\ \mathbb{C}$ together with two natural transformations representing inclusion $(^\circ)\colon \mathbb{C}(X;Y) \to P(X;Y)$ and multiplication $(\star)\colon P(X;Y) \times P(Y;Z) \to P(X;Z)$, and such that

- the left action is premultiplication, $f^\circ \star p = f \succ p$,
- the right action is postmultiplication, $p \star f^\circ = p \prec f$,
- multiplication is dinatural, $p \star (f \succ q) = (p \prec f) \star q$,
- and multiplication is associative, $(p_1 \star p_2) \star p_3 = p_1 \star (p_2 \star p_3)$.

Equivalently, promonads are monoids in the category of endoprofunctors. Every promonad induces a category, its *Kleisli* category, with the same objects as the original $\mathbb{C}$, but with hom-sets given by the promonad, $P(\bullet;\bullet)$. [Rom22]

B.2 Multicategories

Multicategories. We can explain promonoidal categories in terms of their better-known relatives: *multicategories*. Multicategories can be used to describe (non-necessarily-coherent) decomposition. They contain *multimorphisms*, $X \to Y_0, \dots, Y_n$ that represent a way of decomposing an object X into a list of objects $Y_0, \dots, Y_n$.

Definition B.5 (Multicategory). A *multicategory* is a category $\mathbb{C}$ endowed with a set of multimorphisms, $\mathbb{C}(X; Y_0, \dots, Y_n)$ for each list of objects $X_0, \dots, X_n, Y$ in $\mathbb{C}_{\text{obj}}$, and a composition Figure 20 operation

$$(⨾)^{n,m}_{Y_k}\colon \mathbb{C}(X; Y_0, \dots, Y_n) \times \mathbb{C}(Y_i; Z_0, \dots, Z_m) \to \mathbb{C}(Z; Y_0, \dots, X_0, \dots, X_m, \dots, Y_m).$$

Composition is unital, meaning $id_{X_i} ⨾ f = f ⨾ id_Y$ for any f making the equation formally well-typed. Composition is also *associative*, meaning $(h ⨾ g) ⨾ f = h ⨾ (g ⨾ f)$; and $g ⨾ (h ⨾ f) = h ⨾ (g ⨾ f)$ holds whenever it is formally well-typed.

[3]Notation for profunctors conflicts in the literature. To side-step this problem, we use the symbols $(\circ\!\!-\!\!\bullet)$ and $(\bullet\!\!-\!\!\circ)$, where $\circ$ marks the contravariant (op) argument. This idea we take from Mike Shulman.

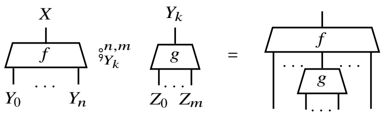


Fig. 20: Multicategorical composition.

Proposition B.6. *Multicategorical composition is dinatural on the object we are composing along. This is to say that composition,* $(⨾)^{n,m}_{k}$*, induces a well-defined and dinatural* composition operation *on the coend the variable* Y_k *we are composing along.*

$$(⨾)^{n,m}_{\bullet_k} \colon \left(\int^{Y_k \in \mathbb{C}} \mathbb{C}(X;Y_0,\dots,Y_n) \times \mathbb{C}(Y_k;Z_0,\dots,Z_m)\right) \to \mathbb{C}(Z;Y_0,\dots,X_0,\dots,X_m,\dots,Y_m).$$

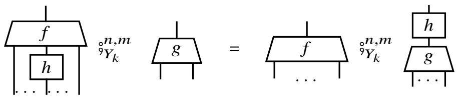


Fig. 21: Multicategorical composition is dinatural.

Proof. This is a direct consequence of the associativity of composition for multicategories, inducing an isomorphism. □

Remark B.7. A promonoidal category is a multicategory where *dinatural composition* is invertible.

Duomulticategories describe the interaction between two kinds of decomposition: a *sequential* one and a *parallel* one. We can mix this two ways of decomposing: for instance, we can decompose X sequentially and then decompose each one of its factors in parallel, finally decompose the last one of these sequentially again.

$$\mathbb{C}(X;(Y_0 \cdot Y_1),(Y_2 \cdot (Y_3,Y_4))).$$

Definition B.8 (Duomulticategory)**.** A *duomulticategory* is a category $\mathbb{C}$ endowed with a set of multimorphisms, $\mathbb{C}(X;E(Y_0,\dots,Y_n))$, for each list of objects $Y_0,\dots,Y_n$ in $\mathbb{C}_{\text{obj}}$ and each expression E on two monoids. Moreover, it is endowed with a dinatural composition operation

$$(⨾)^{n,m}_{Y_k} \colon \int^{Y_i} \mathbb{C}(X;E_1[Y_0,\dots,Y_n]) \times \mathbb{C}(Y_i;E_2[Z_0,\dots,Z_m]) \longrightarrow \mathbb{C}(X;E_1[Y_0,\dots,E_2[X_0,\dots,X_m],\dots,Y_m),$$

and laxators relating sequential and parallel composition,

$$\mathbb{C}(X;E_1[Y_0,\dots,((Z_0,Z_1)\cdot(Z_2,Z_3)),\dots,Y_n]) \longrightarrow \mathbb{C}(X;E_1[Y_0,\dots,((Z_0\cdot Z_2),(Z_1\cdot Z_3)),\dots,Y_n]).$$

Remark B.9. In the same sense that a promonoidal category is a category where dinatural composition is invertible in a specific sense, a produoidal category can be conjectured to be a duomulticategory where dinatural composition is invertible, inducing an isomorphism.

Appendix C
Sequential Context

Proposition C.1 (From Proposition 3.2)**.** *Contour gives a functor* $\mathcal{C}$: **Promon** $\to$ **Cat**.

Proof. Definition 3.1 defines the action on promonoidal categories. We define the action on promonoidal functors. Given a promonoidal functor $F : \mathbb{V} \to \mathbb{W}$, define the functor $\mathcal{C}F : \mathcal{C}\mathbb{V} \to \mathcal{C}\mathbb{W}$ by the following morphism of presentations:

$$X^L \mapsto F(X)^L;\ X^R \mapsto F(X)^R$$

for each $a \in \mathbb{V}(A; N)$, $a_0 : A^L \to A^R \mapsto F_N(a)_0$

for each $b \in \mathbb{V}(X; B)$, $b_0 : B^L \to X^L \mapsto F(b)_0$; $b_1 : X^R \to B^R \mapsto F(b)_1$

for each $c \in \mathbb{V}(C; Y \triangleleft Z)$, $c_0 : C^L \to Y^L \mapsto F_{\triangleleft}(c)_0$; $c_1 : Y^R \to Z^L \mapsto F_{\triangleleft}(c)_1$; $c_2 : Z^R \to C^R \mapsto F_{\triangleleft}(c)_2$.

It follows from $F : \mathbb{V} \to \mathbb{W}$ being a promonoidal functor that the contour equations of Definition 3.1 hold between the images of generators, so this defines a functor. In particular when $\mathbf{Id}_{\mathbb{V}} : \mathbb{V} \to \mathbb{V}$ is an identity, it is an identity functor. Let $G : \mathbb{U} \to \mathbb{V}$ be another promonoidal functor, then $\mathcal{C}(G \fatsemi F) = \mathcal{C}(G) \fatsemi \mathcal{C}(F)$ follows from the composition of promonoidal functors. □

Proposition C.2 (From Proposition 3.5)**.** *Spliced arrows form a promonoidal category with their sequential splits, units, and suitable coherence morphisms.*

Proof. In Lemma C.3, we construct the associator out of Yoneda isomorphisms. In Lemmas C.4 and C.5, we construct both unitors. As they are all constructed with Yoneda isomorphisms, they must satisfy the coherence equations. □

Lemma C.3 (Promonoidal splice associator)**.** *We can construct a natural isomorphism,*

$$\alpha\colon \int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathbb{SC}} \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{A}{B}; \genfrac{}{}{0pt}{}{X}{Y} \triangleleft \genfrac{}{}{0pt}{}{U}{V}\right) \times \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{U}{V}; \genfrac{}{}{0pt}{}{X'}{Y'} \triangleleft \genfrac{}{}{0pt}{}{X''}{Y''}\right) \cong \int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathbb{SC}} \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{A}{B}; \genfrac{}{}{0pt}{}{U}{V} \triangleleft \genfrac{}{}{0pt}{}{X''}{Y''}\right) \times \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{U}{V}; \genfrac{}{}{0pt}{}{X}{Y} \triangleleft \genfrac{}{}{0pt}{}{X'}{Y'}\right),$$

exclusively from Yoneda isomorphisms. This isomorphism is defined by stating that $\alpha(\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2\rangle | \langle g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi g_2\rangle) = (\langle h_0 \fatsemi \square \fatsemi h_1 \fatsemi \square \fatsemi h_2\rangle | \langle k_0 \fatsemi \square \fatsemi k_1 \fatsemi \square \fatsemi k_2\rangle)$ *if and only if*

$$\langle f_0 \fatsemi g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi g_2 \fatsemi f_1 \fatsemi \square \fatsemi g_2\rangle = \langle h_0 \fatsemi \square \fatsemi h_1 \fatsemi k_0 \fatsemi \square \fatsemi k_1 \fatsemi \square \fatsemi k_2 \fatsemi h_2\rangle.$$

Proof. We will show that both sides of the equation are isomorphic to $\mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; X'') \times \mathbb{C}(X''; B)$; that is, the set of quadruples of morphisms $\langle p_0 \fatsemi \square \fatsemi p_1 \fatsemi \square \fatsemi p_2 \fatsemi \square \fatsemi p_3\rangle$ where $p_0\colon A \to X$, $p_1\colon Y \to X'$, $p_2\colon Y' \to X'$ and $p_3\colon Y'' \to B$.

Indeed, the following coend calculus computation constructs an isomorphism,

$$\begin{aligned}
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{A}{B}; \genfrac{}{}{0pt}{}{X}{Y} \triangleleft \genfrac{}{}{0pt}{}{U}{V}\right) \times \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{U}{V}; \genfrac{}{}{0pt}{}{X'}{Y'} \triangleleft \genfrac{}{}{0pt}{}{X''}{Y''}\right) && = \quad \text{(by definition)}\\
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A; X) \times \mathbb{C}(Y; U) \times \mathbb{C}(V; B) \times \mathbb{C}(U; X') \times \mathbb{C}(Y'; X'') \times \mathbb{C}(Y''; V) && = \quad \text{(by definition)}\\
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A; X) \times \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{Y}{B}; \genfrac{}{}{0pt}{}{U}{V}\right) \times \mathbb{C}(U; X') \times \mathbb{C}(Y'; X'') \times \mathbb{C}(Y''; V) && \cong \quad \text{(by Yoneda reduction)}\\
&\mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; X'') \times \mathbb{C}(Y''; B),
\end{aligned}$$

that sends a pair $\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2\rangle | \langle g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi g_2\rangle$, quotiented by the equivalence relation generated by $\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi n \fatsemi \square \fatsemi m \fatsemi f_2\rangle | \langle g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi g_2\rangle = \langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2\rangle | \langle n \fatsemi g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi m \fatsemi g_2\rangle$, to the canonical form $\langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi g_0 \fatsemi \square \fatsemi g_1 \fatsemi \square \fatsemi g_2 \fatsemi f_2\rangle$.

In the same way, the following coend calculus computation constructs the second isomorphism,

$$\begin{aligned}
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{A}{B}; \genfrac{}{}{0pt}{}{U}{V} \triangleleft \genfrac{}{}{0pt}{}{X''}{Y''}\right) \times \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{U}{V}; \genfrac{}{}{0pt}{}{X}{Y} \triangleleft \genfrac{}{}{0pt}{}{X'}{Y'}\right) && \overset{\text{def}}{=}\\
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A; U) \times \mathbb{C}(V; X'') \times \mathbb{C}(Y''; B) \times \mathbb{C}(U; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; V) && \overset{\text{def}}{=}\\
&\int^{\genfrac{}{}{0pt}{}{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\genfrac{}{}{0pt}{}{A}{X''}; \genfrac{}{}{0pt}{}{U}{V}\right) \times \mathbb{C}(Y''; B) \times \mathbb{C}(U; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; V) && \overset{y_1}{\cong}
\end{aligned}$$

$$\mathbb{C}(A;X) \times \mathbb{C}(Y;X') \times \mathbb{C}(Y';X'') \times \mathbb{C}(Y'';B),$$

that sends a pair $\langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2 \rangle | \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \rangle$, quotiented by the equivalence relation generated by $\langle h_0 \mathbin{⨾} n \mathbin{⨾} \square \mathbin{⨾} m \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2 \rangle | \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \rangle = \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2 \rangle | \langle n \mathbin{⨾} k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \mathbin{⨾} m \rangle$, to the canonical form $\langle h_0 \mathbin{⨾} k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2 \rangle$.

In summary, we have that $\alpha(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1 \mathbin{⨾} \square \mathbin{⨾} f_2 \rangle | \langle g_0 \mathbin{⨾} \square \mathbin{⨾} g_1 \mathbin{⨾} \square \mathbin{⨾} g_2 \rangle) = (\langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2 \rangle | \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \rangle)$ if and only if

$$\langle f_0 \mathbin{⨾} g_0 \mathbin{⨾} \square \mathbin{⨾} g_1 \mathbin{⨾} \square \mathbin{⨾} g_2 \mathbin{⨾} f_1 \mathbin{⨾} \square \mathbin{⨾} g_2 \rangle = \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2 \mathbin{⨾} h_2 \rangle,$$

which is what we wanted to prove. □

Lemma C.4 (Promonoidal splice left unitor)**.** *We can construct a natural isomorphism,*

$$\lambda\colon \int^{\binom{U}{V} \in \mathbb{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{U}{V} \triangleleft \tbinom{X}{Y}\right) \times \mathcal{S}\mathbb{C}\left(\tbinom{U}{V}; N\right) \cong \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right),$$

exclusively from Yoneda isomorphisms. This isomorphism is defined by $\lambda(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1 \mathbin{⨾} \square \mathbin{⨾} f_2 \rangle | g) = \langle f_0 \mathbin{⨾} g \mathbin{⨾} f_1 \mathbin{⨾} \square \mathbin{⨾} f_2 \rangle$.

Proof. Indeed, the following coend calculus derivation constructs the isomorphism.

$$\begin{aligned}
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{U}{V} \triangleleft \tbinom{X}{Y}\right) \times \mathcal{S}\mathbb{C}\left(\tbinom{U}{V}; N\right) &&= \quad \text{(by definition)}\\
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A;U) \times \mathbb{C}(V;X) \times \mathbb{C}(Y;B) \times \mathbb{C}(U;V) &&= \quad \text{(by definition)}\\
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{X}; \tbinom{U}{V}\right) \times \mathbb{C}(Y;B) \times \mathbb{C}(U;V) &&\cong \quad \text{(by Yoneda reduction)}\\
&\mathbb{C}(A;X) \times \mathbb{C}(Y;B).
\end{aligned}$$

Thus, it is constructed by a Yoneda isomorphism. □

Lemma C.5 (Promonoidal splice right unitor)**.** *We can construct a natural isomorphism,*

$$\rho\colon \int^{\binom{U}{V} \in \mathbb{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{U}{V}\right) \times \mathcal{S}\mathbb{C}\left(\tbinom{U}{V}; N\right) \cong \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right),$$

exclusively from Yoneda isomorphisms. This isomorphism is defined by $\rho(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1 \mathbin{⨾} \square \mathbin{⨾} f_2 \rangle | g) = \langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1 \mathbin{⨾} g \mathbin{⨾} f_2 \rangle$.

Proof. Indeed, the following coend calculus derivation constructs the isomorphism.

$$\begin{aligned}
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{U}{V}\right) \times \mathcal{S}\mathbb{C}\left(\tbinom{U}{V}; N\right) &&= \quad \text{(by definition)}\\
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A;X) \times \mathbb{C}(Y;U) \times \mathbb{C}(V;B) \times \mathbb{C}(U;V) &&= \quad \text{(by definition)}\\
&\int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathbb{C}(A;X) \times \mathcal{S}\mathbb{C}\left(\tbinom{Y}{B}; \tbinom{U}{V}\right) \times \mathbb{C}(U;V) &&\cong \quad \text{(by Yoneda reduction)}\\
&\mathbb{C}(A;X) \times \mathbb{C}(Y;B).
\end{aligned}$$

Thus, it is constructed by a Yoneda isomorphism. □

Proposition C.6 (From Proposition 3.6)**.** *Splice gives a functor* $\mathcal{S} : \mathbf{Cat} \to \mathbf{Promon}$.

Proof. Definition 3.4 defines the action on categories. We define the action on functors. Given a functor $F : \mathbb{C} \to \mathbb{D}$, define the promonoidal functor $\mathcal{S}F : \mathcal{S}\mathbb{C} \to \mathcal{S}\mathbb{D}$ by

$$\begin{aligned}
&{A \atop B} \mapsto {FA \atop FB}, \\
&\mathcal{S}F := F_{A,X} \times F_{Y,B} : \mathcal{S}\mathbb{C}({A \atop B}, {X \atop Y}) \to \mathcal{S}\mathbb{D}({FA \atop FB}, {FX \atop FY}), \\
&\mathcal{S}F_{\triangleleft} := F_{A,X} \times F_{Y,X'} \times F_{Y',B} : \mathcal{S}\mathbb{C}({A \atop B}, {X \atop Y} \triangleleft {X' \atop Y'}) \to \mathcal{S}\mathbb{D}({FA \atop FB}, {FX \atop FY} \triangleleft {FX' \atop FY'}), \\
&\mathcal{S}F_{N} := F_{A,B} : \mathcal{S}\mathbb{C}({A \atop B}, N) \to \mathcal{S}\mathbb{D}({FA \atop FB}, N).
\end{aligned}$$

It follows from the promonoidal structure on spliced arrows (Proposition C.2) that this preserves coherence maps. If $\mathbf{Id}_{\mathbb{C}} : \mathbb{C} \to \mathbb{C}$ is an identity functor, then it defines the identity $\mathbf{Id}_{\mathcal{S}\mathbb{C}}$, which has underlying functor the identity and identity natural transformations. If $G : \mathbb{B} \to \mathbb{C}$ is another functor, then $\mathcal{S}(G \fatsemi F) = \mathcal{S}G \fatsemi \mathcal{S}F$ follows from composition of functors. □

Theorem C.7 (From Theorem 3.7). *There exists an adjunction between categories and promonoidal categories, where the contour of a promonoidal is the left adjoint, and the splice category is the right adjoint.*

Proof. Let $\mathbb{C}$ be a category and let $\mathbb{B}$ be a promonoidal category. We will show that the promonoidal functors $\mathbb{B} \to \mathcal{S}\mathbb{C}$ are in natural correspondence with the functors $\mathcal{C}\mathbb{B} \to \mathbb{C}$. We first observe that the category $\mathcal{C}\mathbb{B}$ is freely presented; thus, a functor $\mathcal{C}\mathbb{B} \to \mathbb{C}$ amounts to a choice of some objects and some morphisms in $\mathbb{C}$ satisfying some equations. Explicitly, by the definition of contour, a functor $\mathcal{C}\mathbb{B} \to \mathbb{C}$ amounts to

- for each $X \in \mathbb{B}_{\text{obj}}$, a choice of objects $X^L, X^R \in \mathbb{C}_{\text{obj}}$;
- for each element $a \in \mathbb{B}(X)$, a choice of morphisms $a_0 \in \mathbb{C}(X^L, X^R)$;
- for each morphism $a \in \mathbb{B}(A; X)$, a choice of morphisms $a_0 \in \mathbb{C}(A^L; X^L)$ and $a_1 \in \mathbb{C}(X^R; A^R)$;
- for each split $a \in \mathbb{C}(A; X \triangleleft Y)$, a choice of morphisms $a_0 \in \mathbb{C}(A^L; X^L)$, $a_1 \in \mathbb{C}(X^R; Y^L)$ and $a_2 \in \mathbb{C}(Y^R; A^R)$;
- the choice must be such that $\alpha(a \mid b) = (c \mid d)$ implies $a_0 = c_0 \fatsemi d_0$; $a_1 \fatsemi b_0 = d_1$ and $b_1 = d_2 \fatsemi c_1$; $a_2 \fatsemi b_2 = c_2$;
- the choice must be such that $\rho(a|b) = c = \lambda(d|e)$ implies $a_0 = c_0 = d_0 \fatsemi e_0 \fatsemi d_1$ and $a_1 \fatsemi b_0 \fatsemi a_2 = c_1 = d_2$.

On the other hand, a promonoidal functor $\mathbb{B} \to \mathcal{S}\mathbb{C}$, also amounts to

- for each $X \in \mathbb{B}_{\text{obj}}$, an object $FX = (X^L, X^R) \in \mathcal{S}\mathbb{C}_{\text{obj}}$, which is a pair of objects of $\mathbb{C}_{\text{obj}}$;
- for each element $a \in \mathbb{B}(X)$, a morphism $F(a) = a_0 \in \mathcal{S}\mathbb{C}(FX)$;
- for each element $a \in \mathbb{B}(A; X)$, a splice $F(a) = \langle a_0 \fatsemi \square \fatsemi a_1 \rangle \in \mathcal{S}\mathbb{C}(FA; FX)$;
- for each element $a \in \mathbb{B}(A; X \triangleleft Y)$, a splice $F(a) = \langle a_0 \fatsemi \square \fatsemi a_1 \fatsemi \square \fatsemi a_2 \rangle \in \mathcal{S}\mathbb{C}(FA; FX \triangleleft FY)$;
- preserving associativity, with $\alpha(a \mid b) = (c \mid d)$ implying $\alpha(F(a) \mid F(b)) = F(c) \mid F(d)$;
- preserving unitality, with $\rho(a \mid b) = c = \lambda(d \mid e)$ implying $\rho(F(a) \mid F(b)) = F(c) = \lambda(F(d) \mid F(e))$;

by the definition of splice, its associativity and unitality, the structure on each one of these points is exactly equal. □

C.1 *Spliced arrow multicategory*

As a consequence of the previous discussion, the n-morphisms are the sequences of arrows in $\mathbb{C}$ separated by n gaps; the sequence of arrows goes from A to B, with holes typed by $\{X_i, Y_i\}_{i \in [1,\dots,n]}$. In other words,

$$\mathcal{S}\mathbb{C}\left({A \atop B} ; {X_1 \atop Y_1} \otimes \dots \otimes {X_n \atop Y_n}\right) = \mathbb{C}(A; X_1) \times \left(\prod_{k=1}^{n-1} \mathbb{C}(Y_k, X_{k+1})\right) \times \mathbb{C}(Y_n, B).$$

Composition in the multicategory is defined by substitution of a spliced arrow into one of the gaps of the second; the identity is just $\mathrm{id}_A - \mathrm{id}_B$, the spliced arrow with a single gap typed by (A, B).

Proposition C.8. *The multicategory of spliced arrows, $\mathcal{S}\mathbb{C}$, is precisely the promonoidal category induced by the duality $\mathbb{C}^{\text{op}} \dashv \mathbb{C}$ in the monoidal bicategory of profunctors: a promonoidal category over $\mathbb{C} \times \mathbb{C}^{\text{op}}$.*

Appendix D
Parallel-Sequential Context

D.1 *Monoidal Contour*

Definition D.1 (Monoidal contour, from Definition 4.4)**.** The *contour* of a produoidal category $\mathbb{B}$ is the monoidal category $\mathcal{D}\mathbb{B}$ that has two objects, X^L (left-handed) and X^R (right-handed), for each object $X \in \mathbb{B}_{obj}$; and has arrows those that arise from *contouring* both sequential and parallel decompositions of the promonoidal category.

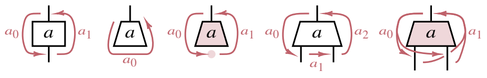


Fig. 22: Generators of the monoidal category of contours.

Specifically, it is freely presented by *(i)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L)$, $a_1 \in \mathcal{D}\mathbb{B}(X^R; A^R)$ for each morphism $a \in \mathbb{B}(A; X)$; *(ii)* a morphism $a_0 \in \mathcal{D}\mathbb{B}(A^L; A^R)$, for each sequential unit $a \in \mathbb{C}(A; N)$; *(iii)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; I)$ and $a_0 \in \mathcal{D}\mathbb{B}(I; A^R)$, for each parallel unit $a \in \mathbb{B}(A; I)$; *(iv)* a triple of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L)$, $a_1 \in \mathcal{D}\mathbb{B}(X^R; Y^L)$, $a_2 \in \mathcal{D}\mathbb{B}(Y^R; A^R)$ for each sequential split $a \in \mathbb{B}(A; X \triangleleft Y)$; and *(v)* a pair of morphisms $a_0 \in \mathcal{D}\mathbb{B}(A^L; X^L \otimes Y^L)$ and $a_1 \in \mathcal{D}\mathbb{B}(X^R \otimes Y^R; A^R)$ for each parallel split $a \in \mathbb{B}(A; X \otimes Y)$, see Figure 22.

For each equality $a \mathbin{⨾}^2 b = c \mathbin{⨾}^1 d$, we impose the equations $a_0 = c_0 ⨾ d_0$; $a_1 ⨾ b_0 = d_1$ and $b_1 = d_2 ⨾ c_1$; $a_2 ⨾ b_2 = c_2$. For each equality $a \mathbin{⨾}^2 b = c = d \mathbin{⨾}^1 e$, we impose $a_0 = c_0 = d_0 ⨾ e_0 ⨾ d_1$ and $a_1 ⨾ b_0 ⨾ a_2 = c_1 = d_2$. These follow from Figure 7.

For each application of associativity, $\alpha(a ⨾_1 b) = c ⨾_2 d$, we impose the equations $a_0 ⨾ (b_0 \otimes \mathrm{id}) = c_0 ⨾ (\mathrm{id} \otimes d_0)$ and $(b_1 \otimes \mathrm{id}) ⨾ a_1 = (\mathrm{id} \otimes d_1) ⨾ c_1$. These follow from Figure 23.

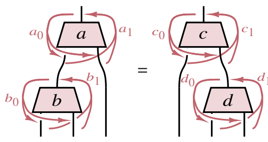


Fig. 23: Equation from associativity.

For each application of unitality, $\lambda(a ⨾_1 b) = c = \rho(d ⨾_2 e)$, we impose the equations $a_0 ⨾ (b_0 \otimes \mathrm{id}) = c_0 = d_0 ⨾ (\mathrm{id} \otimes e_0)$ and $(b_1 \otimes \mathrm{id}) ⨾ a_1 = c_1 = (\mathrm{id} \otimes e_1) ⨾ d_1$. These follow from Figure 24.

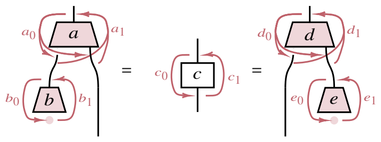


Fig. 24: Equations from unitality.

For each application of the laxator, $\psi_2(a \,|\, b \,|\, c) = (d \,|\, e \,|\, f)$, we impose the equation $a_0 ⨾ (b_0 \otimes c_0) = d_0 ⨾ e_0$, the middle equation $b_1 \otimes c_1 = e_1 ⨾ d_1 ⨾ f_0$, and $(b_2 \otimes c_2) ⨾ a_1 = f_1 ⨾ d_2$. These follow Figure 25.

For each application of the laxator, $\psi_0(a) = (b \,|_1\, c \,|_2\, d)$, we impose an equation $a_0 = b_0 ⨾ c_0$, an equation $\mathrm{id} = c_1 ⨾ b_1 ⨾ d_0$, and an equation $a_1 = d_1 ⨾ b_2$. This follows Figure 26.

For each application of the laxator, $\varphi_2(a \,|_1\, b \,|_2\, c) = d$, we impose an equation $a_0 ⨾ (b_0 \otimes c_0) ⨾ a_1 = d_0$. This follows Figure 27.

For each application of the laxator, $\varphi_0(a) = b$, we impose an equation $a_0 ⨾ a_1 = b_0$. This follows Figure 28.

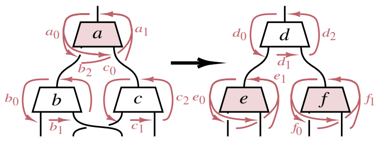


Fig. 25: Equations for the first laxator.

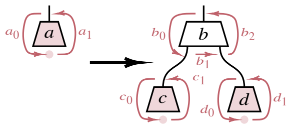


Fig. 26: Equations for the second laxator.

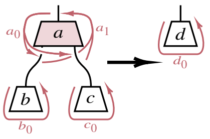


Fig. 27: Equations for the third laxator.

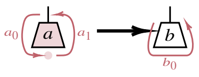


Fig. 28: Equations for the fourth laxator.

Proposition D.2 (From Proposition 4.5)**.** *Monoidal contour gives a functor $\mathcal{D}$:* **Produo** $\to$ **Mon***.*

Proof. Definition 4.4 defines the action on produoidal categories. We define the action on produoidal functors. Given a produoidal functor $F : \mathbb{V} \to \mathbb{W}$, define the strict monoidal functor $\mathcal{D}F : \mathcal{D}\mathbb{V} \to \mathcal{D}\mathbb{W}$ by the following morphism of presentations:

- the objects X^L and X^R are mapped to $F(X)^L$ and $F(X)^R$;
- for each $a \in \mathbb{V}(A;N)$, the morphism $a_0 : A^L \to A$ is mapped to $F_N(a)_0$;
- for each $b \in \mathbb{V}(A;I)$, both $b_0 : A^L \to I$ and $b_1 : I \to A^R$ are mapped to $F_I(b)_0$ and $F_I(b)_1$;
- for each $c \in \mathbb{V}(X;B)$, the morphisms $c_0 : B^L \to X^L, c_1 : X^R \to B^R$ are mapped to $F(c)_0$ and $F(c)_1$;
- for each $d \in \mathbb{V}(C;Y \otimes Z)$, the morphisms $d_0 : C^L \to Y^L$, $d_1 : Y^R \to Z^L$ and $d_2 : Z^R \to C^R$ are mapped to $F_\triangleleft(d)_0$, $F_\triangleleft(d)_1$ and $F_\triangleleft(d)_2$;
- for each $e \in \mathbb{V}(C;Y \triangleleft Z)$, the morphisms $e_0 : C^L \to Y^L$, $e_1 : Y^R \to Z^L$ and $e_2 : Z^R \to C^R$ are mapped to $F_\triangleleft(e)_0$, $F_\triangleleft(e)_1$ and $F_\triangleleft(e)$.

It follows from $F : \mathbb{V} \to \mathbb{W}$ being a produoidal functor that the contour equations of Definition 3.1 hold between the images of generators, so this assignment extends freely to a strict monoidal functor. In particular when $\mathbf{Id}_\mathbb{V} : \mathbb{V} \to \mathbb{V}$ is an identity, it is an identity functor. Let $G : \mathbb{U} \to \mathbb{V}$ be another produoidal functor, then $\mathcal{C}(G \mathbin{⨾} F) = \mathcal{C}(G) \mathbin{⨾} \mathcal{C}(F)$ follows from the composition of produoidal functors. □

D.2 Spliced Monoidal Arrows

Proposition D.3 (From Proposition 4.7)**.** *Spliced monoidal arrows form a produoidal category with their sequential and parallel splits, units, and suitable coherence morphisms and laxators.*

Proof. We use the laxators constructed in Lemmas D.4 to D.7. Because these laxators are constructed out of compositions and Yoneda lemma, they do satisfy all formal coherence equations. □

Lemma D.4 (Produoidal splice, first laxator)**.** *We can construct a natural transformation,*

$$\psi_2 \colon \mathcal{T}\mathbb{C}\left(\tbinom{A}{B}; \left(\tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) \otimes \left(\tbinom{U}{V} \triangleleft \tbinom{U'}{V'}\right)\right) \to \mathcal{T}\mathbb{C}\left(\tbinom{A}{B}; \left(\tbinom{X}{Y} \otimes \tbinom{U}{V}\right) \triangleleft \left(\tbinom{X'}{Y'} \otimes \tbinom{U'}{V'}\right)\right),$$

exclusively from compositions and Yoneda isomorphisms. This laxator is defined by

$\psi_2(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1\rangle \mid \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2\rangle \mid \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2\rangle) = \langle g_0 \mathbin{⨾} \square \mathbin{⨾} g_1 \mathbin{⨾} \square \mathbin{⨾} g_2\rangle | \langle p_0 \mathbin{⨾} \square \mathbin{⨾} p_1\rangle | \langle q_0 \mathbin{⨾} \square \mathbin{⨾} q_1\rangle$

if and only if

$$\langle f_0 \mathbin{⨾} (h_0 \otimes k_0) \mathbin{⨾} \square \mathbin{⨾} h_1 \otimes k_1 \mathbin{⨾} \square \mathbin{⨾} (h_2 \otimes k_2) \mathbin{⨾} f_1\rangle = \langle g_0 \mathbin{⨾} p_0 \mathbin{⨾} \square \mathbin{⨾} p_1 \mathbin{⨾} g_1 \mathbin{⨾} q_0 \mathbin{⨾} \square \mathbin{⨾} q_1 \mathbin{⨾} g_2\rangle.$$

Proof. We will show that the right hand side is isomorphic to the following set. Then, we construct a map from the left hand to this same set,

$$\mathbb{C}(A; X \otimes X') \times \mathbb{C}(Y \otimes Y'; U \otimes U') \times \mathbb{C}(V \otimes V'; B).$$

Indeed the following coend derivation constructs an isomorphism.

$$\begin{aligned}
&\mathcal{TC}\left(\tbinom{A}{B}; \left(\tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \triangleleft \left(\tbinom{U}{V} \otimes \tbinom{U'}{V'}\right)\right) && \overset{\text{def}}{=} \\
&\int^{\binom{Z}{W}, \binom{Z'}{W'} \in \mathcal{TC}} \mathcal{TC}\left(\tbinom{A}{B}; \tbinom{Z}{W} \triangleleft \tbinom{Z'}{W'}\right) \times \mathcal{TC}\left(\tbinom{Z}{W}; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \times \mathcal{TC}\left(\tbinom{Z'}{W'}; \tbinom{U}{V} \otimes \tbinom{U'}{V'}\right) && \overset{\text{def}}{=} \\
&\int^{\binom{Z}{W}, \binom{Z'}{W'} \in \mathcal{TC}} \mathcal{TC}\left(\tbinom{A}{B}; \tbinom{Z}{W} \triangleleft \tbinom{Z'}{W'}\right) \times \mathbb{C}(Z; X \otimes X') \times \mathbb{C}(Y \otimes Y'; W) \times \mathbb{C}(Z'; U \otimes U') \times \mathbb{C}(V \otimes V'; W') && \overset{\text{def}}{=} \\
&\int^{\binom{Z}{W}, \binom{Z'}{W'} \in \mathcal{TC}} \mathcal{TC}\left(\tbinom{A}{B}; \tbinom{Z}{W} \triangleleft \tbinom{Z'}{W'}\right) \times \mathcal{TC}\left(\tbinom{Z}{W}; \tbinom{X \otimes X'}{Y \otimes Y'}\right) \times \mathcal{TC}\left(\tbinom{Z'}{W'}; \tbinom{U \otimes U'}{V \otimes V'}\right) && \overset{y_1}{\cong} \\
&\mathcal{TC}\left(\tbinom{A}{B}; \tbinom{X \otimes X'}{Y \otimes Y'} \triangleleft \tbinom{U \otimes U'}{V \otimes V'}\right) && \overset{\text{def}}{=} \\
&\mathbb{C}(A; X \otimes X') \times \mathbb{C}(Y \otimes Y'; U \otimes U') \times \mathbb{C}(V \otimes V'; B).
\end{aligned}$$

The isomorphism sends the triple $(\langle g_0⨾\square⨾g_1⨾\square⨾g_2\rangle|\langle p_0⨾\square⨾p_1\rangle|\langle q_0⨾\square⨾q_1\rangle)$ to $\langle g_0⨾p_0⨾\square⨾p_1⨾g_1⨾q_0⨾\square⨾q_1⨾g_2\rangle$. On the other hand, we define a map from the left hand side of the equation to this set, given by

$$\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1\rangle \mid \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2\rangle \mid \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2\rangle \mapsto \langle f_0 \mathbin{⨾} (h_0 \otimes k_0) \mathbin{⨾} \square \mathbin{⨾} h_1 \otimes k_1 \mathbin{⨾} \square \mathbin{⨾} (h_2 \otimes k_2) \mathbin{⨾} f_1\rangle.$$

In conclusion, composing both the isomorphism and the map, $\psi_2(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1\rangle \mid \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2\rangle \mid \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2\rangle) = \langle g_0 \mathbin{⨾} \square \mathbin{⨾} g_1 \mathbin{⨾} \square \mathbin{⨾} g_2\rangle | \langle p_0 \mathbin{⨾} \square \mathbin{⨾} p_1\rangle | \langle q_0 \mathbin{⨾} \square \mathbin{⨾} q_1\rangle$ if and only if

$$\langle f_0 \mathbin{⨾} (h_0 \otimes k_0) \mathbin{⨾} \square \mathbin{⨾} (h_1 \otimes k_1) \mathbin{⨾} \square \mathbin{⨾} (h_2 \otimes k_2) \mathbin{⨾} f_1\rangle = \langle g_0 \mathbin{⨾} p_0 \mathbin{⨾} \square \mathbin{⨾} p_1 \mathbin{⨾} g_1 \mathbin{⨾} q_0 \mathbin{⨾} \square \mathbin{⨾} q_1 \mathbin{⨾} g_2\rangle,$$

which is what we wanted to prove. □

Lemma D.5 (Produoidal splice, second laxator)**.** *We can construct a natural transformation,*

$$\psi_0 \colon \mathcal{TC}\left(\tbinom{A}{B}; I\right) \to \mathcal{TC}\left(\tbinom{A}{B}; I \triangleleft I\right),$$

exclusively from compositions and Yoneda isomorphisms. This laxator is defined by $\psi_0(\langle f_0 \mathbin{⨾} \square \mathbin{⨾} f_1\rangle \mid \langle h_0 \mathbin{⨾} \square \mathbin{⨾} h_1 \mathbin{⨾} \square \mathbin{⨾} h_2\rangle \mid \langle k_0 \mathbin{⨾} \square \mathbin{⨾} k_1 \mathbin{⨾} \square \mathbin{⨾} k_2\rangle) = \langle g_0 \mathbin{⨾} \square \mathbin{⨾} g_1 \mathbin{⨾} \square \mathbin{⨾} g_2\rangle | \langle p_0 \mathbin{⨾} \square \mathbin{⨾} p_1\rangle | \langle q_0 \mathbin{⨾} \square \mathbin{⨾} q_1\rangle$ *if and only if*

$$\langle f_0 \mathbin{⨾} (h_0 \otimes k_0) \mathbin{⨾} \square \mathbin{⨾} (h_1 \otimes k_1) \mathbin{⨾} \square \mathbin{⨾} (h_2 \otimes k_2) \mathbin{⨾} f_1\rangle = \langle g_0 \mathbin{⨾} p_0 \mathbin{⨾} \square \mathbin{⨾} p_1 \mathbin{⨾} g_1 \mathbin{⨾} q_0 \mathbin{⨾} \square \mathbin{⨾} q_1 \mathbin{⨾} g_2\rangle.$$

Proof. We will show that the right hand side is isomorphic to the following set. Then, we construct a map from the left hand to this same set, $\mathbb{C}(A; I) \times \mathbb{C}(I; I) \times \mathbb{C}(I; B)$. Indeed the following coend derivation constructs an isomorphism.

$$\begin{aligned}
&\mathcal{TC}\left(\tbinom{A}{B}; I \triangleleft I\right) && \overset{\text{def}}{=} \\
&\int^{\binom{Z}{W}, \binom{Z'}{W'} \in \mathcal{TC}} \mathcal{TC}\left(\tbinom{A}{B}; \tbinom{Z}{W} \triangleleft \tbinom{Z'}{W'}\right) \times \mathcal{TC}\left(\tbinom{Z}{W}; I\right) \times \mathcal{TC}\left(\tbinom{Z'}{W'}; I\right) && \overset{\text{def}}{=} \\
&\int^{\binom{Z}{W}, \binom{Z'}{W'} \in \mathcal{TC}} \mathcal{TC}\left(\tbinom{A}{B}; \tbinom{Z}{W} \triangleleft \tbinom{Z'}{W'}\right) \times \mathcal{TC}\left(\tbinom{Z}{W}; \tbinom{I}{I}\right) \times \mathcal{TC}\left(\tbinom{Z'}{W'}; \tbinom{I}{I}\right) && \overset{y_1}{\cong}
\end{aligned}$$

$$\mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}I\\I\end{smallmatrix} \triangleleft \begin{smallmatrix}I\\I\end{smallmatrix}\right) \overset{\text{def}}{=} \mathbb{C}(A;I) \times \mathbb{C}(I;I) \times \mathbb{C}(I;B).$$

This isomorphism sends the triple $\langle b_0 ⨾ \square ⨾ b_1 ⨾ \square ⨾ b_2 \rangle | \langle c_0 ⨾ \square ⨾ c_1 \rangle | \langle d_0 ⨾ \square ⨾ d_1 \rangle$ to $\langle b_0 ⨾ c_0 ⨾ \square ⨾ c_1 ⨾ b_1 ⨾ d_0 ⨾ \square ⨾ d_1 ⨾ b_2 \rangle$. On the other hand, we define a map from the left hand side of the equation to this set, given by

$$\langle a_0 ⨾ \square ⨾ a_1 \rangle \mapsto \langle a_0 ⨾ \square ⨾ \mathrm{id}_I ⨾ \square ⨾ a_1 \rangle.$$

In conclusion, composing both the isomorphism and this function, we get that $\psi_0 \langle a_0 ⨾ \square ⨾ a_1 \rangle = \langle b_0 ⨾ \square ⨾ b_1 ⨾ \square ⨾ b_2 \rangle \,|\, \langle c_0 ⨾ \square ⨾ c_1 \rangle \,|\, \langle d_0 ⨾ \square ⨾ d_1 \rangle$ if and only if $\langle a_0 ⨾ \square ⨾ \mathrm{id}_I ⨾ \square ⨾ a_1 \rangle = \langle b_0 ⨾ c_0 ⨾ \square ⨾ c_1 ⨾ b_1 ⨾ d_0 ⨾ \square ⨾ d_1 ⨾ b_2 \rangle$. □

Lemma D.6 (Produoidal splice, third laxator)**.** *We can construct a natural transformation,*

$$\varphi_2 \colon \mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; N \otimes N\right) \to \mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; N\right),$$

exclusively from compositions and Yoneda isomorphisms. This laxator is defined by $\varphi_2(\langle f_0 ⨾ \square ⨾ f_1 \rangle \,|\, h_0 \,|\, h_1) = f_0 ⨾ (h_0 \otimes h_1 ⨾ f_1)$.

Lemma D.7 (Produoidal splice, fourth laxator)**.** *We can construct a natural transformation,*

$$\varphi_0 \colon \mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; I\right) \to \mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; N\right),$$

exclusively from compositions and Yoneda isomorphisms. This laxator is defined by $\varphi_0 \langle a_0 ⨾ \square ⨾ a_1 \rangle = a_0 ⨾ a_1$.

Proposition D.8 (From Proposition 4.8)**.** *Monoidal splice gives a functor* $\mathcal{T}$: **Mon** → **Produo**.

Proof. Definition 4.6 defines the action on monoidal categories. We define the action on monoidal functors. Given a monoidal functor $F : \mathbb{C} \to \mathbb{D}$, define the produoidal functor $\mathcal{T}F : \mathcal{T}\mathbb{C} \to \mathcal{T}\mathbb{D}$ by

$$\begin{aligned}
&\begin{smallmatrix}A\\B\end{smallmatrix} \mapsto \begin{smallmatrix}FA\\FB\end{smallmatrix}\\
&\mathcal{T}F := F_{A,X} \times F_{Y,B} : \mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}, \begin{smallmatrix}X\\Y\end{smallmatrix}) \to \mathcal{T}\mathbb{D}(\begin{smallmatrix}FA\\FB\end{smallmatrix}, \begin{smallmatrix}FX\\FY\end{smallmatrix})\\
&\mathcal{T}F_{\triangleleft} := F_{A,X} \times F_{Y,X'} \times F_{Y',B} : \mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}, \begin{smallmatrix}X\\Y\end{smallmatrix} \triangleleft \begin{smallmatrix}X'\\Y'\end{smallmatrix}) \to \mathcal{T}\mathbb{D}(\begin{smallmatrix}FA\\FB\end{smallmatrix}, \begin{smallmatrix}FX\\FY\end{smallmatrix} \triangleleft \begin{smallmatrix}FX'\\FY'\end{smallmatrix})\\
&\mathcal{T}F_{\otimes} := F_{A,X\otimes Y} \times F_{X'\otimes Y',B} : \mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}, \begin{smallmatrix}X\\Y\end{smallmatrix} \otimes \begin{smallmatrix}X'\\Y'\end{smallmatrix}) \to \mathcal{T}\mathbb{D}(\begin{smallmatrix}FA\\FB\end{smallmatrix}, \begin{smallmatrix}FX\\FY\end{smallmatrix} \otimes \begin{smallmatrix}FX'\\FY'\end{smallmatrix})\\
&\mathcal{T}F_N := F_{A,B} : \mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}, N) \to \mathcal{T}\mathbb{D}(\begin{smallmatrix}FA\\FB\end{smallmatrix}, N)\\
&\mathcal{T}F_I := F_{A,I} \times F_{I,B} : \mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}, I) \to \mathcal{T}\mathbb{D}(\begin{smallmatrix}FA\\FB\end{smallmatrix}, I).
\end{aligned}$$

It follows from the produoidal structure on spliced monoidal arrows (Proposition D.3) that this preserves coherence maps. If $\mathbf{Id}_{\mathbb{C}} : \mathbb{C} \to \mathbb{C}$ is an identity functor, then it defines the identity $\mathbf{Id}_{\mathcal{T}\mathbb{C}}$, which has underlying functor the identity and identity natural transformations. If $G : \mathbb{B} \to \mathbb{C}$ is another monoidal functor, then $\mathcal{S}(G ⨾ F) = \mathcal{S}G ⨾ \mathcal{S}F$ follows from composition of monoidal functors. □

Theorem D.9 (From Theorem 4.9)**.** *There exists an adjunction between monoidal categories (and strict monoidal functors) and produoidal categories (and produoidal functors), where the monoidal contour is the left adjoint, and the produoidal splice category is the right adjoint.*

Proof. As in Theorem C.7, we again have that $\mathcal{D}\mathbb{B}$ is presented by generators and equations; so, to specify a strict monoidal functor $\mathcal{D}\mathbb{B} \to \mathbb{M}$, it is enough to specify images of the generators satisfying the equations. Let $(\mathbb{M}, \otimes_M, I_M)$ be a monoidal category. Then a strict monoidal functor $\mathcal{D}\mathbb{B} \to \mathbb{M}$ amounts to the following data.

- For each object $X \in \mathbb{B}_{\text{obj}}$, a pair of objects $X^L, X^R \in \mathbb{M}_{\text{obj}}$;
- for each element $f \in \mathbb{B}(X; N)$, a morphism $f_0 \in \mathbb{M}(X^L; X^R)$;
- for each unit $f \in \mathbb{B}(X; I)$, a choice of morphisms $f_0 \in \mathbb{M}(X^L; I_M)$, $g_0 \in \mathbb{M}(I_M; X^R)$;
- for each morphism $f \in \mathbb{B}(A; X)$, a choice of morphisms $f_0 \in \mathbb{M}(A^L; X^L)$ and $f_1 \in \mathbb{M}(X^R; A^R)$;

- for each sequential split $f \in \mathbb{B}(A; X \triangleleft Y)$, a choice of morphisms $f_0 \in \mathbb{M}(A^L; X^L), f_1 \in \mathbb{M}(X^L; X^R)$, and $f_2 \in \mathbb{M}(X^R, A^R)$;
- for each parallel split $f \in \mathbb{B}(A; X \otimes Y)$, a choice of morphisms $f_0 \in \mathbb{M}(A^L; X^L \otimes Y^L)$ and $f_1 \in \mathbb{M}(X^R \otimes Y^R; A^R)$.

Such that for each promonoidal structure

- $\alpha(a \fatsemi_1 b) = (c \fatsemi_2 d)$ in $\mathbb{B} \Rightarrow a_0 \fatsemi (b_0 \otimes \mathrm{id}) = c_0 \fatsemi (\mathrm{id} \otimes d_0)$ and $(b_1 \otimes \mathrm{id}) \fatsemi a_1 = (\mathrm{id} \otimes d_1) \fatsemi c_1$ in $\mathbb{M}$;
- $\lambda(a \fatsemi_1 b) = c = \rho(d \fatsemi_2 e)$ in $\mathbb{B} \Rightarrow a_0 \fatsemi (b_0 \otimes \mathrm{id}) = c_0 = d_0 \fatsemi (\mathrm{id} \otimes e_0)$ and $(b_1 \otimes \mathrm{id}) \fatsemi a_1 = c_1 = (\mathrm{id} \otimes e_1) \fatsemi d_1$ in $\mathbb{M}$;

and such that

- $\psi_2(a \mid b \mid c) = (d \mid e \mid f)$ in $\mathbb{B} \Rightarrow a_0 \fatsemi (b_0 \otimes c_0) = d_0 \fatsemi e_0, b_1 \otimes c_1 = e_1 \fatsemi d_1 \fatsemi f_0$ and $(b_2 \otimes c_2) \fatsemi a_1 = f_1 \fatsemi d_2$ in $\mathbb{M}$;
- $\psi_0(a) = (b \mid c \mid d)$ in $\mathbb{B} \Rightarrow a_0 = b_0 \fatsemi c_0$, $\mathrm{id} = c_1 \fatsemi b_1 \fatsemi d_0$, and $a_1 = d_1 \fatsemi b_2$ in $\mathbb{M}$;
- $\varphi_2(a \mid b \mid c) = d$ in $\mathbb{B} \Rightarrow a_0 \fatsemi (b_0 \otimes c_0) \fatsemi a_1 = d_0$ in $\mathbb{M}$;
- $\varphi_0(a) = b$ in $\mathbb{B} \Rightarrow a_0 \fatsemi a_1 = b_0$ in $\mathbb{M}$.

On the other hand, a produoidal functor $F : \mathbb{B} \to \mathcal{T}\mathbb{M}$, also amounts to the following data. For each

- $X \in \mathbb{B}_{\mathrm{obj}}$ an object $F(X) = (X^L, X^R) \in \mathcal{T}\mathbb{M}_{\mathrm{obj}}$;
- $f \in \mathbb{B}(X; N)$, an element $F(f) = f_0 \in \mathcal{T}\mathbb{M}(\genfrac{}{}{0pt}{}{X^L}{X^R}; N)$;
- $f \in \mathbb{B}(X; I)$, a unit $F(f) = \langle f \parallel g \rangle \in \mathcal{T}\mathbb{M}(\genfrac{}{}{0pt}{}{X^L}{X^R}; I_M)$
- $f \in \mathbb{B}(A; X)$, a spliced arrow $F(f) = \langle f_0 \fatsemi \square \fatsemi f_1 \rangle \in \mathcal{T}\mathbb{M}(\genfrac{}{}{0pt}{}{A}{B}, \genfrac{}{}{0pt}{}{X}{Y})$;
- $f \in \mathbb{B}(A; X \triangleleft Y)$, a spliced arrow $F(f) = \langle f_0 \fatsemi \square \fatsemi f_1 \fatsemi \square \fatsemi f_2 \rangle \in \mathcal{T}\mathbb{M}(\genfrac{}{}{0pt}{}{A^L}{A^R}, \genfrac{}{}{0pt}{}{X^L}{X^R} \triangleleft \genfrac{}{}{0pt}{}{Y^L}{Y^R})$;
- $f \in \mathbb{B}(A; X \otimes Y)$, a spliced monoidal arrow $F(f) = \langle f_0 \fatsemi \square \otimes \square \fatsemi f_1 \rangle \in \mathcal{T}\mathbb{M}(\genfrac{}{}{0pt}{}{A^L}{A^R}, \genfrac{}{}{0pt}{}{X^L}{X^R} \otimes \genfrac{}{}{0pt}{}{Y^L}{Y^R})$;

Such that for each promonoidal structure

- $\alpha(a \mid b) = (c \mid d)$ in $\mathbb{B} \Rightarrow \alpha(Fa \mid Fb) = (Fc \mid Fd)$ in $\mathcal{T}\mathbb{M}$;
- $\lambda(a \mid b) = c = \rho(d \mid e)$ in $\mathbb{B} \Rightarrow \lambda(Fa \mid Fb) = Fc = \rho(Fd \mid Fe)$ in $\mathcal{T}\mathbb{M}$;
  and such that
- $\psi_2(a \mid b \mid c) = (d \mid e \mid f)$ in $\mathbb{B} \Rightarrow \psi_2(Fa \mid Fb \mid Fc) = (Fd \mid Fe \mid Ff)$ in $\mathcal{T}\mathbb{M}$;
- $\psi_0(a) = (b \mid c \mid d)$ in $\mathbb{B} \Rightarrow \psi_0(Fa) = (Fa \mid Fc \mid Fd)$ in $\mathcal{T}\mathbb{M}$;
- $\varphi_2(a \mid b \mid c) = d$ in $\mathbb{B} \Rightarrow \varphi_2(Fa \mid Fb \mid Fc) = Fd$ in $\mathcal{T}\mathbb{M}$;
- $\varphi_0(a) = b$ in $\mathbb{B} \Rightarrow \varphi_0(Fa) = Fb$ in $\mathcal{T}\mathbb{M}$.

Each of these points is exactly equal by definition, which establishes the desired adjunction. $\square$

Appendix E
Normalization

E.1 Normalization

Theorem E.1 (From Theorem 5.2)**.** *Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a produoidal category. The profunctor $\mathcal{N}\mathbb{V}(\bullet;\bullet) = \mathbb{V}(\bullet;N\otimes\bullet\otimes N)$ forms a promonad. Moreover, the Kleisli category of this promonad is a normal produoidal category with the following sequential and parallel splits and units.*

$$\mathcal{N}\mathbb{V}(A;B) = \mathbb{V}(A;N\otimes B\otimes N);$$
$$\mathcal{N}\mathbb{V}(A;B\otimes C) = \mathbb{V}(A;N\otimes B\otimes N\otimes C\otimes N);$$
$$\mathcal{N}\mathbb{V}(A;B\triangleleft C) = \mathbb{V}(A;(N\otimes B\otimes N)\triangleleft(N\otimes C\otimes N));$$
$$\mathcal{N}\mathbb{V}(A;I) = \mathbb{V}(A;N);$$
$$\mathcal{N}\mathbb{V}(A;N) = \mathbb{V}(A;N).$$

Proof. We define the following multiplication and unit for the promonad, $\mathcal{N}\mathbb{V}$. They are constructed out of laxators of the produoidal category $\mathbb{V}$ and Yoneda isomorphisms; thus, they must be associative and unital by coherence. The unit is defined by

$$\begin{array}{lcl}
\mathbb{V}(A;B) & \cong & \text{(by unitality of } \mathbb{V}) \\
\mathbb{V}(A;I\otimes B\otimes I) & \to & \text{(by the laxators of } \mathbb{V}) \\
\mathbb{V}(A;N\otimes B\otimes N) & = & \text{(by definition)} \\
\mathcal{N}\mathbb{V}(A;B). & &
\end{array}$$

The multiplication is defined by,

$$\begin{array}{lcl}
\int^{B\in\mathbb{V}} \mathcal{N}\mathbb{V}(A;B)\times\mathcal{N}\mathbb{V}(B;C) & = & \text{(by definition)} \\
\int^{B\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes N)\times\mathbb{V}(B;N\otimes C\otimes N) & \cong & \text{(by Yoneda reduction)} \\
\mathbb{V}(A;N\otimes N\otimes C\otimes N\otimes N) & \to & \text{(by laxators of } \mathbb{V}) \\
\mathbb{V}(A;N\otimes C\otimes N) & = & \text{(by definition)} \\
\mathcal{N}\mathbb{V}(A;C). & &
\end{array}$$

Let us now construct the unitors and the associators. Again, they are constructed out of laxators of the produoidal category $\mathbb{V}$, the associators and unitors of $\mathbb{V}$, and Yoneda isomorphisms. We first consider the right unitor.

$$\begin{array}{lcl}
\int^{X\in\mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A;B\otimes X)\times\mathcal{N}\mathbb{V}(X;N) & = & \text{(by definition)} \\
\int^{X\in\mathcal{N}\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes N\otimes X\otimes N)\times\mathcal{N}\mathbb{V}(X;N) & \cong & \text{(by associativity of } \mathbb{V}) \\
\int^{X\in\mathcal{N}\mathbb{V},P\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes P)\times\mathbb{V}(P;N\otimes X\otimes N)\times\mathcal{N}\mathbb{V}(X;N) & = & \text{(by definition)} \\
\int^{X\in\mathcal{N}\mathbb{V},P\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes P)\times\mathcal{N}\mathbb{V}(P;X)\times\mathcal{N}\mathbb{V}(X;N) & \cong & \text{(by Yoneda reduction)} \\
\int^{P\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes P)\times\mathcal{N}\mathbb{V}(P;N) & = & \text{(by definition)} \\
\int^{P\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes P)\times\mathbb{V}(P;N) & \cong & \text{(by unitality)} \\
\int^{P\in\mathbb{V}} \mathbb{V}(A;N\otimes B\otimes N). & &
\end{array}$$

We now consider the left unitor.

$$\begin{aligned}
&\int^{X \in \mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A; X \otimes B) \times \mathcal{N}\mathbb{V}(X; N) && = && \text{(by definition)}\\
&\int^{X \in \mathcal{N}\mathbb{V}} \mathbb{V}(A; N \otimes X \otimes N \otimes B \otimes N) \times \mathcal{N}\mathbb{V}(X; N) && \cong && \text{(by associativity of } \mathbb{V}\text{)}\\
&\int^{X \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; P \otimes B \otimes N) \times \mathbb{V}(P; N \otimes X \otimes N) \times \mathcal{N}\mathbb{V}(X; N) && = && \text{(by definition)}\\
&\int^{X \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; P \otimes B \otimes N) \times \mathcal{N}\mathbb{V}(P; X) \times \mathcal{N}\mathbb{V}(X; N) && \cong && \text{(by Yoneda reduction)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; P \otimes B \otimes N) \times \mathcal{N}\mathbb{V}(P; N) && = && \text{(by definition)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; P \otimes B \otimes N) \times \mathbb{V}(P; N) && \cong && \text{(by unitality)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; N \otimes B \otimes N).
\end{aligned}$$

Finally, we consider the associator. We can do so in two steps, showing that both sides of the equation

$$\int^{X \in \mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A; B \otimes X) \times \mathcal{N}\mathbb{V}(X; C \otimes D) \cong \int^{Y \in \mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A; Y \otimes D) \times \mathcal{N}\mathbb{V}(Y; B \otimes C)$$

are isomorphic to $\mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N \otimes D \otimes N)$. The first side by

$$\begin{aligned}
&\int^{X \in \mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A; B \otimes X) \times \mathcal{N}\mathbb{V}(X; C \otimes D) && = && \text{(by definition)}\\
&\int^{X \in \mathcal{N}\mathbb{V}} \mathbb{V}(A; N \otimes B \otimes N \otimes X \otimes N) \times \mathcal{N}\mathbb{V}(X; C \otimes D) && \cong && \text{(by associativity)}\\
&\int^{X \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; N \otimes B \otimes P) \times \mathbb{V}(P; N \otimes X \otimes N) \times \mathcal{N}\mathbb{V}(X; C \otimes D) && = && \text{(by definition)}\\
&\int^{X \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; N \otimes B \otimes P) \times \mathcal{N}\mathbb{V}(P; X) \times \mathcal{N}\mathbb{V}(X; C \otimes D) && \cong && \text{(by Yoneda reduction)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; N \otimes B \otimes P) \times \mathcal{N}\mathbb{V}(P; C \otimes D) && = && \text{(by definition)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; N \otimes B \otimes P) \times \mathbb{V}(P; N \otimes C \otimes N \otimes D \otimes N) && \cong && \text{(by associativity)}\\
&\mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N \otimes D \otimes N),
\end{aligned}$$

and the second side by

$$\begin{aligned}
&\int^{Y \in \mathcal{N}\mathbb{V}} \mathcal{N}\mathbb{V}(A; Y \otimes D) \times \mathcal{N}\mathbb{V}(Y; B \otimes C) && = && \text{(by definition)}\\
&\int^{Y \in \mathcal{N}\mathbb{V}} \mathbb{V}(A; N \otimes Y \otimes N \otimes D \otimes N) \times \mathcal{N}\mathbb{V}(Y; B \otimes C) && \cong && \text{(by associativity)}\\
&\int^{Y \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; P \otimes D \otimes N) \times \mathbb{V}(P; N \otimes Y \otimes N) \times \mathcal{N}\mathbb{V}(Y; B \otimes C) && = && \text{(by definition)}\\
&\int^{Y \in \mathcal{N}\mathbb{V}, P \in \mathbb{V}} \mathbb{V}(A; P \otimes D \otimes N) \times \mathcal{N}\mathbb{V}(P; X) \times \mathcal{N}\mathbb{V}(X; B \otimes C) && \cong && \text{(by Yoneda reduction)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; P \otimes D \otimes N) \times \mathcal{N}\mathbb{V}(P; B \otimes C) && = && \text{(by definition)}\\
&\int^{P \in \mathbb{V}} \mathbb{V}(A; P \otimes D \otimes N) \times \mathbb{V}(P; N \otimes B \otimes N \otimes C \otimes N) && \cong && \text{(by associativity)}
\end{aligned}$$

$\mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N \otimes D \otimes N)$.

Precisely because they are constructed out of coherence morphisms for the base produoidal category $\mathbb{V}$, we know that these satisfy the pentagon and triangle equations and define a promonoidal category. The unitors and associators for the sequential promonoidal structure are defined similarly. Finally, we define the laxators of $\mathcal{N}\mathbb{V}$, making it into a produoidal category.

The first laxator,

$$\psi_2 : \mathcal{N}\mathbb{V}(A; (B_1 \triangleleft C_1) \otimes (B_2 \triangleleft C_2)) \longrightarrow \mathcal{N}\mathbb{V}(A; (B_1 \otimes B_2) \triangleleft (C_1 \otimes C_2)),$$

is defined by the following reasoning.

$$\begin{aligned}
&\mathcal{N}\mathbb{V}(A; (B_1 \triangleleft C_1) \otimes (B_2 \triangleleft C_2)) \\
&\quad = \quad \text{(by definition)} \\
&\mathbb{V}(A; N \otimes ((N \otimes B_1 \otimes N) \triangleleft (N \otimes C_1 \otimes N)) \otimes N \otimes ((N \otimes B_2 \otimes N) \triangleleft (N \otimes C_2 \otimes N)) \otimes N) \\
&\quad \to \quad \text{(by } \psi_2 \text{ of } \mathbb{V}) \\
&\mathbb{V}(A; ((N \otimes N \otimes B_1 \otimes N) \triangleleft (N \otimes N \otimes B_2 \otimes N)) \otimes ((N \otimes N \otimes B_2 \otimes N \otimes N) \triangleleft (N \otimes C_2 \otimes N \otimes N))) \\
&\quad \to \quad \text{(by } \psi_2 \text{ of } \mathbb{V}) \\
&\mathbb{V}(A; (N \otimes N \otimes B_1 \otimes N \otimes N \otimes N \otimes B_2 \otimes N \otimes N) \triangleleft (N \otimes N \otimes C_1 \otimes N \otimes N \otimes N \otimes C_2 \otimes N \otimes N)) \\
&\quad \to \quad \text{(by } \varphi_2 \text{ of } \mathbb{V}) \\
&\mathbb{V}(A; (N \otimes N \otimes B_1 \otimes N \otimes B_2 \otimes N \otimes N) \triangleleft (N \otimes N \otimes C_1 \otimes N \otimes C_2 \otimes N \otimes N)) \\
&\quad = \quad \text{(by definition)} \\
&\mathcal{N}\mathbb{V}(A; (B_1 \otimes B_2) \triangleleft (C_1 \otimes C_2)).
\end{aligned}$$

The remaining laxators are isomorphisms that arise from applications of unitality or just as identities.

$$\begin{aligned}
\psi_0 &: \mathcal{N}\mathbb{V}(A, I) \xrightarrow{\cong} \mathcal{N}\mathbb{V}(A; I \triangleleft I) \\
\varphi_2 &: \mathcal{N}\mathbb{V}(A; N \otimes N) \xrightarrow{\cong} \mathcal{N}\mathbb{V}(A; N) \\
\varphi_0 &: \mathcal{N}\mathbb{V}(A; I) \xrightarrow{id} \mathcal{N}\mathbb{V}(A; N)
\end{aligned}$$

This has shown that the resulting category is also a *normal* produoidal category. □

Proposition E.2. *Normalization extends to a endofunctor of produoidal categories* $\mathcal{N}$: **Produo** → **Produo**.

Proof. Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ and $\mathbb{W}_{\oslash,J,\blacktriangleleft,M}$ be produoidal categories. $\mathcal{N}$ sends $\mathbb{V}$ to its normalization $\mathcal{N}\mathbb{V}$. Let $(F, F_\otimes, F_I, F_\triangleleft, F_N) : \mathbb{V} \to \mathbb{W}$ be a produoidal functor. Then $\mathcal{N}F : \mathcal{N}\mathbb{V} \to \mathcal{N}\mathbb{W}$ has underlying functor defined by F on objects and on morphisms by

$$\begin{aligned}
&\mathbb{V}(A; N \otimes B \otimes N) && = \quad \text{(by definition)} \\
&\int^{X,Y\in\mathbb{V}} \mathbb{V}(A; X \otimes B \otimes Y) \times \mathbb{V}(X; N) \times \mathbb{V}(Y; N) && \to \quad \text{(induced by } F_\otimes, F_N) \\
&\int^{X,Y\in\mathbb{V}} \mathbb{W}(FA; FX \oslash FB \oslash FY) \times \mathbb{W}(FX; M) \times \mathbb{W}(FY; M) && \to \quad \text{(inclusion, universal prop. of coend)} \\
&\int^{P,Q\in\mathbb{W}} \mathbb{W}(FA; P \oslash FB \oslash Q) \times \mathbb{W}(P; M) \times \mathbb{W}(Q; M) && = \quad \text{(by definition)} \\
&\mathbb{W}(FA; M \oslash FB \oslash M).
\end{aligned}$$

$\mathcal{N}F_\otimes$ and $\mathcal{N}F_\triangleleft$ are defined similarly, and $\mathcal{N}F_N$ is F_N. We have $\mathcal{N}\mathbf{Id}_\mathbb{V} = \mathbf{Id}_{\mathcal{N}\mathbb{V}}$, since all the data of the left hand side is given by identity maps on $\mathcal{N}\mathbb{V}$, and if $G : \mathbb{U} \to \mathbb{V}$ is another produoidal functor, then $\mathcal{N}(G \mathbin{⨾} F) = \mathcal{N}G \mathbin{⨾} \mathcal{N}F$ follows from the naturality of the components of F and G. □

Theorem E.3 (From Theorem 5.3)**.** *The functor $\mathcal{N}$:* **Produo** $\to$ **Produo** *from Proposition E.2 is an idempotent monad.*

Proof. Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a produoidal category and let $\triangleleft_N, \otimes_N, N$ denote the sequential splits, parallel splits, and unit in its normalization $\mathcal{N}\mathbb{V}$.

The monad has unit η with component at $\mathbb{V}$ the following produoidal functor $\eta_{\mathbb{V}} : \mathbb{V} \to \mathcal{N}\mathbb{V}$. The underlying functor is the functor induced by the promonad [Rom22, Lemma 3.8]: it is identity on objects, and acts on morphisms by the unit of the promonad. The following components of the produoidal functor preserve laxators and coherence maps since they are constructed only from laxators and coherence maps.

$$
\begin{aligned}
&\eta_{\otimes} : \mathbb{V}(A; B \otimes C) \overset{\lambda,\rho}{\to} \mathbb{V}(A; I \otimes B \otimes I \otimes C \otimes I) \xrightarrow{\varphi_0} \mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N),\\
&\eta_I : \mathbb{V}(A; I) \xrightarrow{\varphi_0} \mathbb{V}(A; N),\\
&\eta_{\triangleleft} : \mathbb{V}(A; B \triangleleft C) \overset{\lambda,\rho}{\to} \mathbb{V}(A; (I \otimes B \otimes I) \triangleleft (I \otimes C \otimes I)) \xrightarrow{\varphi_0} \mathbb{V}(A; (N \otimes B \otimes N) \triangleleft (N \otimes C \otimes N)),\\
&\eta_N : \mathbb{V}(A; N) \overset{\mathrm{id}}{\to} \mathbb{V}(A; N).
\end{aligned}
$$

The monad has multiplication μ with component at $\mathbb{V}$ the following *isomorphism* $\mu_{\mathbb{V}} : \mathcal{N}\mathcal{N}\mathbb{V} \cong \mathcal{N}\mathbb{V}$ of produoidal categories (witnessing that the monad is idempotent). The underlying functor is identity on objects, and acts on morphisms by

$$\mathcal{N}\mathcal{N}\mathbb{V}(A; B) = \mathcal{N}\mathbb{V}(A; N \otimes_N B \otimes_N N) \overset{\lambda,\rho}{\cong} \mathcal{N}\mathbb{V}(A; B).$$

The following natural transformations make this a produoidal functor:

$$
\begin{aligned}
&\mu_{\otimes} : \mathcal{N}\mathcal{N}\mathbb{V}(A; B \otimes_{NN} C) = \mathcal{N}\mathbb{V}(A; N \otimes_N B \otimes_N N \otimes_N C \otimes_N N) \overset{\lambda,\rho}{\cong} \mathcal{N}\mathbb{V}(A; B \otimes_N C),\\
&\mu_N = \mu_I : \mathcal{N}\mathcal{N}\mathbb{V}(A; N) = \mathcal{N}\mathbb{V}(A; N),\\
&\mu_{\triangleleft} : \mathcal{N}\mathcal{N}\mathbb{V}(A; B \triangleleft_{NN} C) = \mathcal{N}\mathbb{V}(A; (N \otimes_N B \otimes_N N) \triangleleft_N (N \otimes_N C \otimes_N N)) \overset{\lambda,\rho}{\cong} \mathcal{N}\mathbb{V}(A; B \triangleleft_N C).
\end{aligned}
$$

.

Finally we verify the monad laws. $\eta_{\mathcal{N}\mathbb{V}} ⨟ \mu_{\mathbb{V}}$ is identity on objects and on morphisms applies left and right unitors followed by their inverses, thus has underlying functor equal to the identity. The components of the natural transformations are also identities, since the laxator φ_0 is an identity for $\mathcal{N}\mathbb{V}$, and they are otherwise composed of unitors followed by their inverses, and similarly for the other unit law (using the unitality coherence equations of Figure 36). $\mu_{\mathcal{N}\mathbb{V}} ⨟ \mu_{\mathbb{V}}$ and $\mathcal{N}\mu_{\mathbb{V}} ⨟ \mu_{\mathbb{V}}$ are identity on objects and amount to applying left and right unitors twice on morphisms, and similarly for their components. □

Lemma E.4. *A produoidal category $\mathbb{V}$ has exactly one algebra structure for the normalization monad when it is normal, and none otherwise.*

Proof. Let $(f_{\mathrm{map}}, f_{\otimes}, f_I, f_{\triangleleft}, f_N) \colon \mathcal{N}\mathbb{V} \to \mathbb{V}$ be an algebra. This means that the following commutative diagrams with the unit and multiplication of the normalization monad must commute.

$$\begin{tikzcd}
\mathbb{V} \arrow[r, "\eta"] \arrow[dr, "\mathrm{id}"'] & \mathcal{N}\mathbb{V} \arrow[d, "f"] \\
& \mathbb{V}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathcal{N}\mathcal{N}\mathbb{V} \arrow[r, "\mu"] \arrow[d, "\mathcal{N}f"'] & \mathcal{N}\mathbb{V} \arrow[d, "f"] \\
\mathcal{N}\mathbb{V} \arrow[r, "f"'] & \mathbb{V}
\end{tikzcd}$$

Now, consider how the laxator $\psi_0 \colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; N)$ is transported by these maps.

$$\begin{tikzcd}
 & \mathbb{V}(\bullet; N) \arrow[dr, "f_I"] \arrow[d, "\mathrm{id}"] & \\
\mathbb{V}(\bullet; I) \arrow[ur, "\eta_I"] \arrow[d, "\psi_0"'] \arrow[rr, bend right, "\mathrm{id}"] & \mathbb{V}(\bullet; N) \arrow[dr, "f_N"] & \mathbb{V}(\bullet; I) \arrow[d, "\psi_0"] \\
\mathbb{V}(\bullet; N) \arrow[ur, "\mathrm{id}"] \arrow[rr, bend right, "\mathrm{id}"'] & & \mathbb{V}(\bullet; N)
\end{tikzcd}$$

We conclude that $\eta_I = \psi_0$, but also that $f_N = \mathrm{id}$. As a consequence, ψ_0 is invertible and f_I must be its inverse. We have shown that the produoidal category $\mathbb{V}$ must be normal.

We will now show that this already determines all of the functor f. We know that $\eta_\otimes, \eta_\triangleleft, \eta_{\mathrm{map}}$ are isomorphisms because they are constructed from the unitors, associators, and the laxator ψ_0, which is an isomorphism in this case. This determines that $f_\otimes, f_\triangleleft, f_{\mathrm{map}}$ must be their inverses. By construction, these satisfy all coherence morphisms. □

Theorem E.5 (From Theorem 5.4)**.** *Normalization determines an adjunction between produoidal categories and normal produoidal categories,*

$$\mathcal{N} \colon \mathbf{Produo} \rightleftharpoons \mathbf{nProduo} \colon \mathcal{U}$$

That is, $\mathcal{N}\mathbb{V}$ is the free normal produoidal category over $\mathbb{V}$.

Proof. We know that the algebras for the normalization monad are exactly the normal produoidal categories (Lemma E.4). We also know that the normalization monad is idempotent (Theorem 5.3). This implies that the forgetful functor from its category of algebras is fully faithful, and thus, the algebra morphisms are exactly the produoidal functors. As a consequence, the canonical adjunction to the category of algebras of the monad is exactly an adjunction to the category of normal produoidal categories. □

E.2 Symmetric Normalization

Theorem E.6 (From Theorem 5.6)**.** *Let $\mathbb{V}_{\otimes,I,\triangleleft,N}$ be a symmetric produoidal category. The profunctor $\mathcal{N}_\sigma\mathbb{V}(\bullet;\bullet) = \mathbb{V}(\bullet; N \otimes \bullet)$ forms a promonad. Moreover, the Kleisli category of this promonad is a normal symmetric produoidal category with the following sequential and parallel splits and units.*

$$\begin{aligned}
\mathcal{N}_\sigma\mathbb{V}(A; B) &= \mathbb{V}(A; N \otimes B);\\
\mathcal{N}_\sigma\mathbb{V}(A; B \otimes_N C) &= \mathbb{V}(A; N \otimes B \otimes C);\\
\mathcal{N}_\sigma\mathbb{V}(A; B \triangleleft_N C) &= \mathbb{V}(A; (N \otimes B) \triangleleft (N \otimes C));\\
\mathcal{N}_\sigma\mathbb{V}(A; N) &= \mathbb{V}(A; N);\\
\mathcal{N}_\sigma\mathbb{V}(A; I) &= \mathbb{V}(A; N).
\end{aligned}$$

Proof. The unit and multiplication of the promonad are given in essentially the same way as in the proof of Theorem E.1. Likewise the associators, unitors and laxators of $\mathcal{N}_\sigma\mathbb{V}$ are given in essentially the same way, though one must use the fact that $\mathbb{V}$ is symmetric. We need additionally a symmetry natural isomorphism for $\mathcal{N}_\sigma\mathbb{V}$. Its components are defined by,

$$\begin{aligned}
&\mathcal{N}_\sigma\mathbb{V}(A; B \otimes C) && = \quad \text{(by definition)}\\
&\mathbb{V}(A; N \otimes B \otimes C) && \cong \quad \text{(by associativity)}\\
&\int^{X \in \mathbb{V}} \mathbb{V}(A; N \otimes X) \times \mathbb{V}(X; B \otimes C) && \cong \quad \text{(by symmetry of } \mathbb{V}\text{)}\\
&\int^{X \in \mathbb{V}} \mathbb{V}(A; N \otimes X) \times \mathbb{V}(X; C \otimes B) && \cong \quad \text{(by associativity)}\\
&\mathbb{V}(A; N \otimes C \otimes B) && = \quad \text{(by definition)}
\end{aligned}$$

$\mathcal{N}_\sigma\mathbb{V}(A; C \otimes B)$.

These satisfy hexagon and symmetry identities because these are satisfied by $\mathbb{V}$, and we only use symmetries and coherences of $\mathbb{V}$. Thus we have a normal symmetric produoidal category $\mathcal{N}_\sigma\mathbb{V}$. □

Definition E.7 (Symmetric produoidal functor)**.** A *symmetric produoidal functor* is a produoidal functor $F \colon \mathbb{V} \to \mathbb{W}$ that moreover preserves the symmetry, in that $F_\otimes \mathbin{⨟} \sigma_\mathbb{V} = \sigma_\mathbb{W} \mathbin{⨟} F_\otimes$. We denote by **SymProduo** the category of symmetric produoidal categories and symmetric produoidal functors.

Proposition E.8. *Symmetric normalization extends to a endofunctor of symmetric produoidal categories* $\mathcal{N}_\sigma$: **SymProduo** → **SymProduo**.

Proof. The construction is essentially the same as in Proposition E.2. The only thing left to check is that $\mathcal{N}_\sigma F$ there constructed preserves symmetries whenever F does (see Theorem E.6). This is because the symmetry of $\mathcal{N}_\sigma\mathbb{V}$ is constructed out of associativity and symmetries of $\mathbb{V}$, which $\mathcal{N}_\sigma F_\otimes$, constructed itself out of $F_\otimes$, associativity, and symmetries of $\mathbb{V}$, must preserve. □

Theorem E.9. *The functor* $\mathcal{N}_\sigma$: **SymProduo** → **SymProduo** *from Proposition E.2 is an idempotent monad.*

Proof. The construction is again essentially the same as in Theorem E.3. It is left to check that the unit and multiplication constructed in this way preserve the symmetries. Indeed, $\eta_\sigma \colon \mathbb{V} \to \mathcal{N}_\sigma\mathbb{V}$ is symmetric produoidal because $\eta_\otimes$ is constructed out of natural associators and laxators that commute with the symmetry. □

Lemma E.10. *A symmetric produoidal category* $\mathbb{V}$ *has exactly one algebra structure for the symmetric normalization monad when it is normal, and none otherwise.*

Proof. The proof essentially follows the same reasoning as Lemma E.4, replacing the construction with the symmetric version and the previous lemmas. □

Theorem E.11 (From Theorem 5.7)**.** *Symmetric normalization determines an adjunction between symmetric produoidal categories and normal symmetric produoidal categories,*

$$\mathcal{N}_\sigma \colon \textbf{SymProduo} \rightleftharpoons \textbf{nSymProduo} \colon \mathcal{U}$$

Where we define the category of normal symmetric produoidal categories, **nSymProduo***, to use as functors the symmetric produoidal functors, adquiring a full forgetful functor* $\mathcal{U}$*.*

That is, $\mathcal{N}_\sigma\mathbb{V}$ *is the free symmetric normal produoidal category over the symmetric produoidal category* $\mathbb{V}$*.*

Proof. The proof essentially follows Theorem E.5, now using the previous lemmas and Lemma E.10. □

E.3 Normalization of duoidals and normalization of produoidals

We conjecture that the normalization of a produoidal category could still be seen to arise from the normalization procedure for duoidal categories outlined by Garner and López Franco [GF16]. Every produoidal category $\mathbb{V}$ induces a closed duoidal structure on its presheaf category $\hat{\mathbb{V}} := [\mathbb{V}^{op}, \textbf{Set}]$: indeed, by a result of Day, any promonoidal structure induces a closed monoidal structure on the presheaf category [Day70b], [Day70a]; furthermore, one can confirm that the two closed monoidal structures on $\hat{\mathbb{V}}$ interact in such a way as to make the category duoidal (Theorem I.6).

Normalizing the duoidal $\hat{\mathbb{V}}$ yields the category of algebras $\mathrm{EM}(\mathcal{N}\mathbb{V})$ for the promonad $\mathcal{N}\mathbb{V}$ – or, equivalently, the category of algebras for the cocontinuous monad induced by $\mathcal{N}\mathbb{V}$ on $\hat{\mathbb{V}}$. $\mathrm{EM}(\mathcal{N}\mathbb{V})$ is now normal duoidal, and furthermore the closure of the tensors on $\hat{\mathbb{V}}$ carries across to make $\mathrm{EM}(\mathcal{N}\mathbb{V})$ also closed. Now, one notes that we have the following isomorphism $\mathrm{EM}(\mathcal{N}\mathbb{V}) \cong [\mathcal{N}\mathbb{V}^{op}, \textbf{Set}]$, that is, the category of algebras is the presheaf category of the Kleisli object $\mathcal{N}\mathbb{V}$ of the promonad in **Prof**. Therefore, the closed monoidal structures of $\mathrm{EM}(\mathcal{N}\mathbb{V})$ must correspond to promonoidal structures of $\mathcal{N}\mathbb{V}$ and these interact so as to make $\mathcal{N}\mathbb{V}$ produoidal.

Appendix F
Monoidal Context

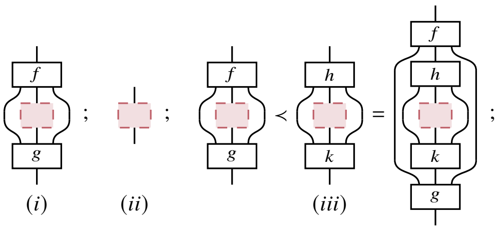


Fig. 29: Generic monoidal context (i), identity (ii) and composition (iii).

Remark F.1 (Algebra of monoidal contexts). We explicitly state all the operations that form the normal produoidal algebra of monoidal contexts. We do so using 1-dimensional notation for compactness, but we do believe the conceptual picture is clearer when they are translated into 2-dimensional string diagrams.

(Identity)

$(\mathrm{id}_A \mathbin{⨟} \blacksquare \mathbin{⨟} \mathrm{id}_B)$

(Composition)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g) \prec (h \mathbin{⨟} (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \mathbin{⨟} k) =$$
$$(f \mathbin{⨟} (\mathrm{id}_M \otimes h \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q\otimes N}) \mathbin{⨟} (\mathrm{id}_M \otimes k \otimes \mathrm{id}_N) \mathbin{⨟} g),$$

(Unit action)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g) \prec h = f \mathbin{⨟} (\mathrm{id}_M \otimes h \otimes \mathrm{id}_N) \mathbin{⨟} g,$$

(Seq. split first action)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \mathbin{⨟} h) \prec_1 (u \mathbin{⨟} (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \mathbin{⨟} v) =$$
$$f \mathbin{⨟} (\mathrm{id}_M \otimes u \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q\otimes N}) \mathbin{⨟} (\mathrm{id}_M \otimes v \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \mathbin{⨟} h,$$

(Seq. split second action)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \mathbin{⨟} h) \prec_2 (u \mathbin{⨟} (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \mathbin{⨟} v) =$$
$$f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes u \otimes \mathrm{id}_L) \mathbin{⨟} (\mathrm{id}_{K\otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q\otimes L}) \mathbin{⨟} (\mathrm{id}_K \otimes v \otimes \mathrm{id}_L) \mathbin{⨟} h,$$

(Seq. split third action)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g) \prec (u \mathbin{⨟} (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \mathbin{⨟} v \mathbin{⨟} (\mathrm{id}_R \otimes \blacksquare \otimes \mathrm{id}_S) \mathbin{⨟} w) =$$
$$f \mathbin{⨟} (\mathrm{id}_M \otimes u \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q\otimes N}) \mathbin{⨟} (\mathrm{id}_M \otimes v \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes R} \otimes \blacksquare \otimes \mathrm{id}_{S\otimes N})$$
$$\mathbin{⨟} (\mathrm{id}_M \otimes w \otimes \mathrm{id}_N) \mathbin{⨟} g,$$

(Seq. left associativity)

$$(f \mathbin{⨟} (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \mathbin{⨟} h) \prec_1^{\alpha} (u \mathbin{⨟} (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \mathbin{⨟} v \mathbin{⨟} (\mathrm{id}_R \otimes \blacksquare \otimes \mathrm{id}_S) \mathbin{⨟} w) =$$
$$f \mathbin{⨟} (\mathrm{id}_M \otimes u \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q\otimes N}) \mathbin{⨟} (\mathrm{id}_M \otimes v \otimes \mathrm{id}_N) \mathbin{⨟} (\mathrm{id}_{M\otimes R} \otimes \blacksquare \otimes \mathrm{id}_{S\otimes N})$$
$$\mathbin{⨟} (\mathrm{id}_M \otimes w \otimes \mathrm{id}_N) \mathbin{⨟} g \mathbin{⨟} (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \mathbin{⨟} h,$$

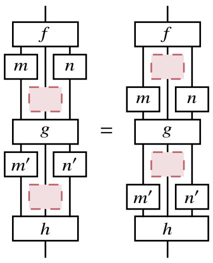


Fig. 30: Dinaturality of sequential splits of monoidal contexts.

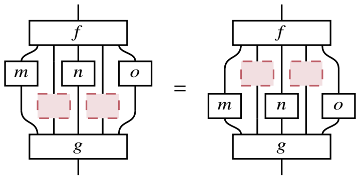


Fig. 31: Parallel splits for monoidal contexts.

(Seq. right associativity)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi h) \prec^{\alpha}_{2} (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \fatsemi v \fatsemi (\mathrm{id}_R \otimes \blacksquare \otimes \mathrm{id}_S) \fatsemi w) =$$
$$f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes u \otimes \mathrm{id}_L) \fatsemi (\mathrm{id}_{K \otimes P} \otimes \blacksquare \otimes \mathrm{id}_{R \otimes L}) \fatsemi (\mathrm{id}_K \otimes v \otimes \mathrm{id}_L)$$
$$\fatsemi (\mathrm{id}_{K \otimes R} \otimes \blacksquare \otimes \mathrm{id}_{S \otimes L}) \fatsemi (\mathrm{id}_L \otimes w \otimes \mathrm{id}_L) \fatsemi h,$$

(Seq. left unitor)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi h) \prec^{\lambda} u =$$
$$f \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi h,$$

(Seq. right unitor)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi h) \prec^{\rho} u =$$
$$f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes u \otimes \mathrm{id}_L) \fatsemi h,$$

(Par. split first action)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec_1 (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \fatsemi v) =$$
$$f \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_{N \otimes X' \otimes O}) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \mathrm{id}_{Q \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes v \otimes \mathrm{id}_{N \otimes Y' \otimes O}) \fatsemi g,$$

(Par. split second action)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec_2 (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q) \fatsemi v) =$$
$$f \fatsemi (\mathrm{id}_{M \otimes X \otimes N} \otimes u \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes P} \otimes \blacksquare \otimes \mathrm{id}_{O \otimes Q}) \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes v \otimes \mathrm{id}_O) \fatsemi g,$$

(Par. split third action)

$$(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) \prec (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_R) \fatsemi v) =$$
$$f \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes N}) \fatsemi (\mathrm{id}_M \otimes w \otimes \mathrm{id}_N) \fatsemi g,$$

(Par. left associativity)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec_1^{\alpha} (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_R) \fatsemi v) = \\
&\quad f \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_{N \otimes X \otimes O}) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \\
&\qquad \fatsemi (\mathrm{id}_M \otimes v \otimes \mathrm{id}_{N \otimes Y \otimes O}) \fatsemi g,
\end{aligned}$$

(Par. right associativity)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec_2^{\alpha} (u \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_R) \fatsemi v) = \\
&\quad f \fatsemi (\mathrm{id}_{M \otimes X \otimes N} \otimes u \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes O}) \\
&\qquad \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes v \otimes \mathrm{id}_O) \fatsemi g,
\end{aligned}$$

(Par. left unitor)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec^{\lambda} u = \\
&\quad f \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_{N \otimes X' \otimes O}) \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g = \\
&\quad f \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes u \otimes \mathrm{id}_{N \otimes X' \otimes O}) \fatsemi g,
\end{aligned}$$

(Par. right unitor)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \prec^{\rho} v = \\
&\quad f \fatsemi (\mathrm{id}_{M \otimes X \otimes N} \otimes v \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes Y' \otimes O}) \fatsemi g = \\
&\quad f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes Y' \otimes O}) \fatsemi (\mathrm{id}_{M \otimes X \otimes N} \otimes v \otimes \mathrm{id}_O) \fatsemi g,
\end{aligned}$$

(Laxator, left side)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi g) \\
&\quad \prec_1^{\psi} (j_0 \fatsemi (\mathrm{id}_U \otimes \blacksquare \otimes \mathrm{id}_V) \fatsemi j_1 \fatsemi (\mathrm{id}_{U'} \otimes \blacksquare \otimes \mathrm{id}_{V'}) \fatsemi j_2) \\
&\quad \prec_2^{\psi} (k_0 \fatsemi (\mathrm{id}_W \otimes \blacksquare \otimes \mathrm{id}_T) \fatsemi k_1 \fatsemi (\mathrm{id}_{W'} \otimes \blacksquare \otimes \mathrm{id}_{T'}) \fatsemi k_2) = \\
&\quad f \fatsemi (\mathrm{id}_M \otimes j_0 \otimes \mathrm{id}_N \otimes k_0 \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_{M \otimes U} \otimes \blacksquare \otimes \mathrm{id}_{V \otimes N \otimes U'} \otimes \blacksquare \otimes \mathrm{id}_{V' \otimes O}) \\
&\qquad \fatsemi (\mathrm{id}_M \otimes j_1 \otimes \mathrm{id}_N \otimes k_1 \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_{M \otimes W} \otimes \blacksquare \otimes \mathrm{id}_{T \otimes N \otimes W'} \otimes \blacksquare \otimes \mathrm{id}_{T' \otimes O}) \\
&\qquad \fatsemi (\mathrm{id}_M \otimes j_2 \otimes \mathrm{id}_N \otimes k_2 \otimes \mathrm{id}_O) \fatsemi g,
\end{aligned}$$

(Laxator, right side)

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi h) \\
&\quad \prec_1^{\psi} (j_0 \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_R) \fatsemi j_1) \\
&\quad \prec_2^{\psi} (k_0 \fatsemi (\mathrm{id}_{P'} \otimes \blacksquare \otimes \mathrm{id}_{Q'} \otimes \blacksquare \otimes \mathrm{id}_{R'}) \fatsemi k_1) = \\
&\quad f \fatsemi (\mathrm{id}_M \otimes j_0 \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes N}) \\
&\qquad \fatsemi (\mathrm{id}_M \otimes j_1 \otimes \mathrm{id}_N) \fatsemi g \fatsemi (\mathrm{id}_K \otimes k_0 \otimes \mathrm{id}_L) \fatsemi (\mathrm{id}_{K \otimes P'} \otimes \blacksquare \otimes \mathrm{id}_{Q'} \otimes \blacksquare \otimes \mathrm{id}_{R' \otimes L}) \\
&\qquad \fatsemi (\mathrm{id}_K \otimes k_1 \otimes \mathrm{id}_L) \fatsemi h.
\end{aligned}$$

Remark F.2. In the following derivations, we understand that an isolated ($\blacksquare$) actually means ($\mathrm{id}_I \otimes \blacksquare \otimes \mathrm{id}_I$).

Proposition F.3 (From Proposition 6.2)**.** *Monoidal contexts form a category. Composition of monoidal contexts is associative and unital.*

Proof. We first check that the composition of monoidal contexts is associative.

$$((f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) \prec (f' \fatsemi (\mathrm{id}_{M'} \otimes \blacksquare \otimes \mathrm{id}_{N'}) \fatsemi g')) \prec (f'' \fatsemi (\mathrm{id}_{M''} \otimes \blacksquare \otimes \mathrm{id}_{N''}) \fatsemi g'') \quad =$$

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes f' \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_{M\otimes M'} \otimes \blacksquare \otimes \mathrm{id}_{N'\otimes N}) \fatsemi (\mathrm{id}_M \otimes g' \otimes \mathrm{id}_N) \fatsemi g) \prec \\
&\qquad (f'' \fatsemi (\mathrm{id}_{M''} \otimes \blacksquare \otimes \mathrm{id}_{N''}) \fatsemi g'') && = \\
&f \fatsemi (\mathrm{id}_M \otimes f' \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_{M\otimes M'} \otimes f'' \otimes \mathrm{id}_{N\otimes N'}) \\
&\qquad \fatsemi (\mathrm{id}_{M\otimes M'\otimes M''} \otimes \blacksquare \otimes \mathrm{id}_{N''\otimes N'\otimes N}) \fatsemi (\mathrm{id}_{M\otimes M'} \otimes g'' \otimes \mathrm{id}_{N'\otimes N}) \fatsemi (\mathrm{id}_N \otimes g' \otimes \mathrm{id}_N) \fatsemi g && = \\
&f \fatsemi (\mathrm{id}_M \otimes (f' \fatsemi (\mathrm{id}_{M'} \otimes f'' \otimes \mathrm{id}_{N'})) \otimes \mathrm{id}_N) \\
&\qquad \fatsemi (\mathrm{id}_{M\otimes M'\otimes M''} \otimes \blacksquare \otimes \mathrm{id}_{N''\otimes N'\otimes N}) \fatsemi (\mathrm{id}_M \otimes ((\mathrm{id}_{M'} \otimes g'' \otimes \mathrm{id}_{N'}) \fatsemi g') \otimes \mathrm{id}_N) \fatsemi g && = \\
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) \prec (f' \fatsemi (\mathrm{id}_{M'} \otimes f'' \otimes \mathrm{id}_{N'}) \fatsemi \\
&\qquad (\mathrm{id}_{M'\otimes M''} \otimes \blacksquare \otimes \mathrm{id}_{N''\otimes N'}) \fatsemi (\mathrm{id}_{M'} \otimes g'' \otimes \mathrm{id}_{N'}) \fatsemi g') && = \\
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) \prec ((f' \fatsemi (\mathrm{id}_{M'} \otimes \blacksquare \otimes \mathrm{id}_{N'}) \fatsemi g') \prec (f'' \fatsemi (\mathrm{id}_{M''} \otimes \blacksquare \otimes \mathrm{id}_{N''}) \fatsemi g''))
\end{aligned}$$

We now check left unitality of the identities,

$$\begin{aligned}
&(f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) \prec (\mathrm{id}_X \fatsemi (\mathrm{id}_I \otimes \blacksquare \otimes \mathrm{id}_I) \fatsemi \mathrm{id}_Y) && = \\
&\qquad (f \fatsemi (\mathrm{id}_M \otimes \mathrm{id}_X \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_M \otimes \mathrm{id}_X \otimes \mathrm{id}_N) \fatsemi g) && = \\
&\qquad (f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g),
\end{aligned}$$

and right unitality,

$$\begin{aligned}
&(\mathrm{id}_A \fatsemi (\mathrm{id}_I \otimes \blacksquare \otimes \mathrm{id}_I) \fatsemi \mathrm{id}_B) \prec (f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g) && = \\
&\qquad (\mathrm{id}_A \fatsemi (\mathrm{id}_I \otimes f \otimes \mathrm{id}_I) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi (\mathrm{id}_I \otimes g \otimes \mathrm{id}_I) \fatsemi \mathrm{id}_B) && = \\
&\qquad (f \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi g).
\end{aligned}$$

This concludes the proof. □

Proposition F.4 (From Proposition 6.5)**.** *The category of monoidal contexts forms a normal produoidal category with its units, sequential and parallel splits.*

Proof. Lemmas F.5 to F.7 construct the associators and unitors for the sequential promonoidal structure, and Lemmas F.8 to F.10 define the associators and unitors for the parallel promonoidal structure. As they are all constructed with Yoneda isomorphisms, they must satisfy the coherence equations. Lemma F.11 defines the laxators, again using only Yoneda isomorphisms and composition in $\mathbb{C}$. For concision, our proofs freely elide the tensor product of objects, writing XY for $X \otimes Y$. □

Lemma F.5 (Monoidal contexts sequential associator)**.** *We construct a natural isomorphism*

$$(\prec_2^{\alpha}) : \int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \triangleleft \tbinom{U}{V}\right) \times \mathcal{M}\mathbb{C}\left(\tbinom{U}{V} ; \tbinom{X'}{Y'} \triangleleft \tbinom{X''}{Y''}\right) \cong \int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{U}{V} \triangleleft \tbinom{X''}{Y''}\right) \times \mathcal{M}\mathbb{C}\left(\tbinom{U}{V} ; \tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) : (\prec_1^{\alpha}),$$

satisfying the coherence equations of produoidal categories. This isomorphism is defined on representatives of the equivalence class as

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_2) \prec_2^{\alpha} (g_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_2) = \\
&((h_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi h_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi h_2) \mid (k_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi k_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi k_2))
\end{aligned}$$

if and only if

$$\begin{aligned}
&f_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_1 \fatsemi (\mathrm{id} \otimes g_0 \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g_1 \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g_2 \otimes \mathrm{id}) \fatsemi f_2 = \\
&h_0 \fatsemi (\mathrm{id} \otimes k_0 \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes k_1 \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes k_2 \otimes \mathrm{id}) \fatsemi h_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi h_2.
\end{aligned}$$

Proof. Firstly, we construct an isomorphism between the left hand side and a set of quadruples of morphisms. This isomorphism sends the pair

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_2) \mid (g_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_2) \\
\text{to}\quad &(f_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_1 \fatsemi (\mathrm{id} \otimes g_0 \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_1 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi (\mathrm{id} \otimes g_2 \otimes \mathrm{id}) \fatsemi f_2).
\end{aligned}$$

The isomorphism is constructed by the following coend derivation.

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{U\\V}\right) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; \substack{X'\\Y'} \triangleleft \substack{X''\\Y''}\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, M,N,O,P \in \mathbb{C}} \mathbb{C}(A; MXN) \times \mathbb{C}(MYN; OUP) \times \mathbb{C}(OVP; B) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; \substack{X'\\Y'} \triangleleft \substack{X''\\Y''}\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, M,N,O,P,Q,R \in \mathbb{C}} \mathbb{C}(A; MXN) \times \mathcal{M}\mathbb{C}\left(\substack{MYN\\B} ; \substack{U\\V}\right) \times \mathbb{C}(U; OX'P) \times \mathbb{C}(OY'P; QX''R) \times \mathbb{C}(QY''R; V) && \overset{y_2}{\cong} \\
&\int^{M,N,O,P,Q,R \in \mathbb{C}} \mathbb{C}(A; MXN) \times \mathbb{C}(MYN; OX'P) \times \mathbb{C}(OY'P; QX''R) \times \mathbb{C}(QY''R; B).
\end{aligned}$$

Now we construct an isomorphism between the right hand side and the same set of quadruples of morphisms. This isomorphism sends the pair

$$\begin{aligned}
&(h_0 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ h_1 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ h_2) \mid (k_0 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ k_1 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ k_2)) \\
\text{to} \quad &(h_0 ⨟ (\mathrm{id} \otimes k_0 \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ k_1 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ (\mathrm{id} \otimes k_2 \otimes \mathrm{id}) ⨟ h_1 ⨟ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨟ h_2).
\end{aligned}$$

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{U\\V} \triangleleft \substack{X''\\Y''}\right) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, M,N,O,P \in \mathbb{C}} \mathbb{C}(A; MUN) \times \mathbb{C}(MVN; OX''P) \times \mathbb{C}(OY''P; B) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, M,N,O,P,Q,R \in \mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\OX''P} ; \substack{U\\V}\right) \times \mathbb{C}(OY''P; B) \times \mathbb{C}(U; MXN) \times \mathbb{C}(MYN; QX'R) \times \mathbb{C}(QY'R; V) && \overset{y_2}{\cong} \\
&\int^{M,N,O,P,Q,R \in \mathbb{C}} \mathbb{C}(A; MXN) \times \mathbb{C}(MYN; QX'R) \times \mathbb{C}(QY'R; OX''P) \times \mathbb{C}(OY''P; B).
\end{aligned}$$

Composing both isomorphisms, we obtain the desired associator. Since it is composed exclusively from Yoneda isomorphisms, it must satisfy the coherence equations of produoidal categories (Definition I.5). ◻

Lemma F.6 (Monoidal contexts sequential left unitor)**.** *We construct a natural isomorphism*

$$(\prec^{\lambda}) : \int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{U\\V} \triangleleft \substack{X\\Y}\right) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; N\right) \cong \mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y}\right),$$

satisfying the coherence equations of produoidal categories. This isomorphism is defined on representatives of the equivalence class as

$$\begin{aligned}
&(f_0 ⨟ (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) ⨟ f_1 ⨟ (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) ⨟ f_2) \prec^{\lambda} g \qquad = \\
&f_0 ⨟ (\mathrm{id}_M \otimes g \otimes \mathrm{id}_N) ⨟ f_1 ⨟ (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) ⨟ f_2.
\end{aligned}$$

Proof. We need to prove that this function is well-defined and does indeed induce an isomorphism after quotienting. We show this by constructing the isomorphism using coend calculus.

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{U\\V} \triangleleft \substack{X\\Y}\right) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; N\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, P,Q,R,S \in \mathbb{C}} \mathbb{C}(A; PUQ) \times \mathbb{C}(PVQ; RXS) \times \mathbb{C}(RYS; B) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; N\right) && \overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}, R,S \in \mathbb{C}} \mathcal{M}\mathbb{C}\left(\substack{A\\RXS} ; \substack{U\\V}\right) \times \mathbb{C}(RYS; B) \times \mathcal{M}\mathbb{C}\left(\substack{U\\V} ; N\right) && \overset{y_2}{\cong} \\
&\int^{R,S \in \mathbb{C}} \mathbb{C}(A; RXS) \times \mathbb{C}(RYS; B) && \overset{\text{def}}{=} \\
&\mathcal{M}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y}\right).
\end{aligned}$$

Since it is composed exclusively from Yoneda isomorphisms, it must satisfy the coherence equations of produoidal categories (Definition I.5). □

Lemma F.7 (Monoidal contexts sequential right unitor)**.** *We construct a natural isomorphism*

$$(\prec^{\rho}) : \int^{\substack{U\\V} \in \mathcal{MC}} \mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{U\\V}\right) \times \mathcal{MC}\left(\substack{U\\V} ; N\right) \cong \mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y}\right)$$

satisfying the coherence equations of produoidal categories. This isomorphism is defined on representatives of the equivalence class as

$$\begin{aligned}&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi f_1 \fatsemi (\mathrm{id}_K \otimes \blacksquare \otimes \mathrm{id}_L) \fatsemi f_2) \prec^{\rho} g &&= \\ &f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N) \fatsemi f_1 \fatsemi (\mathrm{id}_K \otimes g \otimes \mathrm{id}_L) \fatsemi f_2.\end{aligned}$$

Proof. As above, we do this by coend calculus:

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{MC}} \mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{U\\V}\right) \times \mathcal{MC}\left(\substack{U\\V} ; N\right) &&\overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{MC}, P,Q,R,S \in \mathbb{C}} \mathbb{C}(A; PXQ) \times \mathbb{C}(PYQ; RUS) \times \mathbb{C}(RVS; B) \times \mathcal{MC}\left(\substack{U\\V} ; N\right) &&\overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{MC}, P,Q,R,S \in \mathbb{C}} \mathbb{C}(A; PXQ) \times \mathcal{MC}\left(\substack{PYQ\\B} ; \substack{U\\V}\right) \times \mathcal{MC}\left(\substack{U\\V} ; N\right) &&\overset{y_2}{\cong} \\
&\int^{R,S \in \mathbb{C}} \mathbb{C}(A; RXS) \times \mathbb{C}(RYS; B) &&\overset{\text{def}}{=} \\
&\mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y}\right).
\end{aligned}$$

Since it is composed exclusively from Yoneda isomorphisms, it must satisfy the coherence equations of produoidal categories (Definition I.5). □

Lemma F.8 (Monoidal contexts parallel associator)**.** *We construct a natural isomorphism*

$$(\prec_2^{\alpha}) : \int^{\substack{U\\V} \in \mathcal{MC}} \mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y} \otimes \substack{U\\V}\right) \times \mathcal{MC}\left(\substack{U\\V} ; \substack{X'\\Y'} \otimes \substack{X''\\Y''}\right) \cong \int^{\substack{U\\V} \in \mathcal{MC}} \mathcal{MC}\left(\substack{A\\B} ; \substack{U\\V} \otimes \substack{X''\\Y''}\right) \times \mathcal{MC}\left(\substack{U\\V} ; \substack{X\\Y} \otimes \substack{X'\\Y'}\right) : (\prec_1^{\alpha})$$

exclusively from Yoneda isomorphisms. This isomorphism is defined on representatives of the equivalence class as

$$\begin{aligned}&(f_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi f_1) \prec_1^{\alpha} (g_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi g_1) &&= \\ &(h_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi h_1) \mid (j_0 \fatsemi (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) \fatsemi j_1)\end{aligned}$$

if and only if

$$\begin{aligned}&f_0 \fatsemi (\mathrm{id}_M \otimes g_0 \otimes \mathrm{id}_{N \otimes X \otimes O}) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes g_1 \otimes \mathrm{id}_{N \otimes Y \otimes O}) \fatsemi f_1 = \\ &h_0 \fatsemi (\mathrm{id}_{M \otimes X \otimes N} \otimes j_0 \otimes \mathrm{id}_O) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes P} \otimes \blacksquare \otimes \mathrm{id}_Q \otimes \blacksquare \otimes \mathrm{id}_{R \otimes O}) \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes j_1 \otimes \mathrm{id}_O) \fatsemi h_1,\end{aligned}$$

Proof. The left hand side is isomorphic to the following set,

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{MC}} \mathcal{MC}\left(\substack{A\\B} ; \substack{X\\Y} \otimes \substack{U\\V}\right) \times \mathcal{MC}\left(\substack{U\\V} ; \substack{X'\\Y'} \otimes \substack{X''\\Y''}\right) &&\overset{\text{def}}{=} \\
&\int^{\substack{U\\V} \in \mathcal{MC}, M,N,O \in \mathbb{C}} \mathbb{C}(A; M \otimes X \otimes N \otimes U \otimes O) \times \mathbb{C}(M \otimes Y \otimes N \otimes V \otimes O; B) \times \mathcal{MC}\left(\substack{U\\V} ; \substack{X'\\Y'} \otimes \substack{X''\\Y''}\right) &&\overset{y_2}{\cong} \\
&\int^{\substack{U\\V} \in \mathcal{MC}, M,N,O,P,Q \in \mathbb{C}} \mathbb{C}(A; M \otimes X \otimes P) \times \mathbb{C}(P; N \otimes U \otimes O) \times \mathbb{C}(M \otimes Y \otimes Q; B) \\
&\qquad \times \mathbb{C}(N \otimes V \otimes O; Q) \times \mathcal{MC}\left(\substack{U\\V} ; \substack{X'\\Y'} \otimes \substack{X''\\Y''}\right) &&\overset{\text{def}}{=}
\end{aligned}$$

$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, M, M', N', O', P, Q \in \mathbb{C}} \mathbb{C}(A; M \otimes X \otimes P) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}P\\Q\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix}\right) \times \mathbb{C}(M \otimes Y \otimes Q; B) \times \mathbb{C}(U; M' \otimes X' \otimes N' \otimes X'' \otimes O')$$
$$\times\, \mathbb{C}(M' \otimes Y' \otimes N' \otimes Y'' \otimes O'; V) \qquad \overset{y_2}{\cong}$$
$$\int^{M, M', N', O' \in \mathbb{C}} \mathbb{C}(A; M \otimes X \otimes M' \otimes X' \otimes N' \otimes X'' \otimes O') \times \mathbb{C}(M \otimes Y \otimes M' \otimes Y' \otimes N' \otimes Y'' \otimes O'; B).$$

In the same way, the right hand side is isomorphic to the following set,

$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix} \otimes \begin{smallmatrix}X''\\Y''\end{smallmatrix}\right) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}U\\V\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix} \otimes \begin{smallmatrix}X'\\Y'\end{smallmatrix}\right) \qquad \overset{\text{def}}{=}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, M, N, O \in \mathbb{C}} \mathbb{C}(A; M \otimes U \otimes N \otimes X'' \otimes O) \times \mathbb{C}(M \otimes V \otimes N \otimes Y'' \otimes O; B) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}U\\V\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix} \otimes \begin{smallmatrix}X'\\Y'\end{smallmatrix}\right) \qquad \overset{y_1}{\cong}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, M, N, O, P, Q \in \mathbb{C}} \mathbb{C}(P; M \otimes U \otimes N) \times \mathbb{C}(A; P \otimes X'' \otimes O) \times \mathbb{C}(M \otimes V \otimes N; Q) \qquad \overset{\text{def}}{=}$$
$$\times\, \mathbb{C}(Q \otimes Y'' \otimes O; B) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}U\\V\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix} \otimes \begin{smallmatrix}X'\\Y'\end{smallmatrix}\right) \qquad \overset{y_1}{\cong}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, M', N', O', O, P, Q \in \mathbb{C}} \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}P\\Q\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix}\right) \times \mathbb{C}(A; P \otimes X'' \otimes O) \times \mathbb{C}(Q \otimes Y'' \otimes O; B) \qquad \overset{\text{def}}{=}$$
$$\times\, \mathbb{C}(U; M' \otimes X \otimes N' \otimes X' \otimes O') \times \mathbb{C}(M' \otimes Y \otimes N' \otimes Y' \otimes O'; V) \qquad \overset{y_1}{\cong}$$
$$\int^{M', N', O', O, P, Q \in \mathbb{C}} \mathbb{C}(A; P \otimes X'' \otimes O) \times \mathbb{C}(Q \otimes Y'' \otimes O; B)$$
$$\times\, \mathbb{C}(P; M' \otimes X \otimes N' \otimes X' \otimes O') \times \mathbb{C}(M' \otimes Y \otimes N' \otimes Y' \otimes O'; Q) \qquad \overset{y_1}{\cong}$$
$$\int^{M', N', O', O \in \mathbb{C}} \mathbb{C}(A; M' \otimes X \otimes N' \otimes X' \otimes O' \otimes X'' \otimes O) \times \mathbb{C}(M' \otimes Y \otimes N' \otimes Y' \otimes O' \otimes Y'' \otimes O; B).$$

Composing both isomorphisms, we obtain the desired associator. □

Lemma F.9 (Monoidal contexts parallel left unitor)**.** *We construct a natural isomorphism*

$$(\prec^{\lambda}) : \int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix} \otimes \begin{smallmatrix}X\\Y\end{smallmatrix}\right) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}U\\V\end{smallmatrix}; N\right) \cong \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix}\right)$$

exclusively from Yoneda isomorphisms. This isomorphism is defined by

$$(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_H) \fatsemi f_1) \prec^{\lambda} g = f_0 \fatsemi (\mathrm{id}_M \otimes g \otimes \mathrm{id}_{N \otimes X' \otimes O}) \fatsemi (\mathrm{id}_{M \otimes Y \otimes N} \otimes \blacksquare \otimes \mathrm{id}_O) \fatsemi f_1.$$

Proof.

$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix} \otimes \begin{smallmatrix}X\\Y\end{smallmatrix}\right) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}U\\V\end{smallmatrix}; N\right) \qquad \overset{\text{def}}{=}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, P, Q, R \in \mathbb{C}} \mathbb{C}(A; P \otimes U \otimes Q \otimes X \otimes R) \times \mathbb{C}(P \otimes V \otimes Q \otimes Y \otimes R; B) \times \mathbb{C}(U; V) \qquad \overset{y_1}{\cong}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, P, Q, R, S, T \in \mathbb{C}} \mathbb{C}(A; S \otimes X \otimes R) \times \mathbb{C}(S; P \otimes U \otimes Q) \times \mathbb{C}(P \otimes V \otimes Q; T)$$
$$\times\, \mathbb{C}(T \otimes Y \otimes R; B) \times \mathbb{C}(U; V) \qquad \overset{\text{def}}{=}$$
$$\int^{\begin{smallmatrix}U\\V\end{smallmatrix}\in\mathcal{M}\mathbb{C}, R, S, T \in \mathbb{C}} \mathbb{C}(A; S \otimes X \otimes R) \times \mathcal{M}\mathbb{C}\left(\begin{smallmatrix}S\\T\end{smallmatrix}; \begin{smallmatrix}U\\V\end{smallmatrix}\right) \times \mathbb{C}(T \otimes Y \otimes R; B) \times \mathbb{C}(U; V) \qquad \overset{y_1}{\cong}$$
$$\int^{S, R \in \mathbb{C}} \mathbb{C}(A; S \otimes X \otimes R) \times \mathbb{C}(S \otimes Y \otimes R; B) \qquad \overset{\text{def}}{=}$$
$$\mathcal{M}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix}\right).$$

□

Lemma F.10 (Monoidal contexts parallel right unitor)**.** *We construct a natural isomorphism*

$$(\prec^{\rho}) : \int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{U}{V}\right) \times \mathcal{M}\mathbb{C}\left(\tbinom{U}{V} ; N\right) \cong \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y}\right)$$

exclusively from Yoneda isomorphisms. This isomorphism is defined by

$$(f_0 ⨾ (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_N \otimes \blacksquare \otimes \mathrm{id}_H) ⨾ f_1) \prec^{\rho} g = f_0 ⨾ (\mathrm{id}_M \otimes \blacksquare \otimes \mathrm{id}_{N \otimes Y' \otimes O}) ⨾ (\mathrm{id}_{M \otimes X \otimes N} \otimes g \otimes \mathrm{id}_O) ⨾ f_1.$$

Proof. We construct the isomorphism by the following coend derivation,

$$\begin{aligned}
&\int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{U}{V}\right) \times \mathcal{M}\mathbb{C}\left(\tbinom{U}{V} ; N\right) && \overset{\text{def}}{=} \\
&\int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}, P, Q, R \in \mathbb{C}} \mathbb{C}(A; P \otimes X \otimes Q \otimes U \otimes R) \times \mathbb{C}(P \otimes Y \otimes Q \otimes V \otimes R; B) \times \mathbb{C}(U; V) && \overset{y_1}{\cong} \\
&\int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}, P, Q, R, S, T \in \mathbb{C}} \mathbb{C}(A; P \otimes X \otimes S) \times \mathbb{C}(S; Q \otimes U \otimes R) \times \mathbb{C}(Q \otimes V \otimes R; T) \\
&\qquad \times \mathbb{C}(P \otimes Y \otimes T; B) \times \mathbb{C}(U; V) && \overset{\text{def}}{=} \\
&\int^{\binom{U}{V} \in \mathcal{M}\mathbb{C}, P, S, T \in \mathbb{C}} \mathbb{C}(A; P \otimes X \otimes S) \times \mathcal{M}\mathbb{C}\left(\tbinom{S}{T} ; \tbinom{U}{V}\right) \times \mathbb{C}(P \otimes Y \otimes T; B) \times \mathbb{C}(U; V) && \overset{y_1}{\cong} \\
&\int^{P, T \in \mathbb{C}} \mathbb{C}(A; P \otimes X \otimes T) \times \mathbb{C}(P \otimes Y \otimes T; B) && \overset{\text{def}}{=} \\
&\mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y}\right).
\end{aligned}$$

This concludes the proof. □

Lemma F.11 (Monoidal contexts laxators)**.** *We construct the following morphisms*

$$\begin{aligned}
\psi_2 &: \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \left(\tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) \otimes \left(\tbinom{U}{V} \triangleleft \tbinom{U'}{V'}\right)\right) \to \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; \left(\tbinom{X}{Y} \otimes \tbinom{U}{V}\right) \triangleleft \left(\tbinom{X'}{Y'} \otimes \tbinom{U'}{V'}\right)\right) \\
\psi_0 &: \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; I\right) \to \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; I \triangleleft I\right) \\
\varphi_2 &: \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; N \otimes N\right) \to \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; N\right) \\
\varphi_0 &: \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; I\right) \to \mathcal{M}\mathbb{C}\left(\tbinom{A}{B} ; N\right).
\end{aligned}$$

exclusively from composition in $\mathbb{C}$ and Yoneda isomorphisms. The laxator ψ_2 is defined by stating that the following equation holds

$$\begin{aligned}
&(f_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ f_1) \\
&\qquad \prec^{\psi}_{1} (g_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ g_1 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ g_2) \\
&\qquad \prec^{\psi}_{2} (h_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ h_1 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ h_2) = \\
&(j_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ j_1 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ j_2) \mid \\
&\qquad (k_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ k_1) \mid \\
&\qquad (l_0 ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ l_1)
\end{aligned}$$

if and only if

$$\begin{aligned}
&f_0 ⨾ (\mathrm{id} \otimes g_0 \otimes \mathrm{id} \otimes h_0 \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes g_1 \otimes \mathrm{id} \otimes h_1 \otimes \mathrm{id}) ⨾ \\
&\qquad (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes g_2 \otimes \mathrm{id} \otimes h_2 \otimes \mathrm{id}) ⨾ f_1 = \\
&j_0 ⨾ (\mathrm{id} \otimes k_0 \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes k_1 \otimes \mathrm{id}) ⨾ j_1 ⨾ (\mathrm{id} \otimes l_0 \otimes \mathrm{id}) ⨾ \\
&\qquad (\mathrm{id} \otimes \blacksquare \otimes \mathrm{id} \otimes \blacksquare \otimes \mathrm{id}) ⨾ (\mathrm{id} \otimes l_1 \otimes \mathrm{id}) ⨾ j_2.
\end{aligned}$$

Furthermore, since $\mathcal{M}\mathbb{C}$ is normal, ψ_0, φ_2, and φ_0 are isomorphisms.

Proof. Consider the right hand side of ψ_2. It is isomorphic to the following

$$\begin{aligned}
&\int^{\tbinom{P}{Q},\tbinom{P'}{Q'}\in\mathcal{M}\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{B};\tbinom{P}{Q}\triangleleft\tbinom{P'}{Q'}\right)\times\mathcal{M}\mathbb{C}\left(\tbinom{P}{Q};\tbinom{X}{Y}\otimes\tbinom{U}{V}\right)\times\mathcal{M}\mathbb{C}\left(\tbinom{P'}{Q'};\tbinom{X'}{Y'}\otimes\tbinom{U'}{V'}\right) && \overset{\text{def}}{=}\\
&\int^{\tbinom{P}{Q},\tbinom{P'}{Q'}\in\mathcal{M}\mathbb{C},M,N,O\in\mathbb{C}} \mathbb{C}(A;M\otimes P\otimes N)\times\mathbb{C}(M\otimes Q\otimes N;M'\otimes P'\otimes N')\times\mathbb{C}(M'\otimes Q'\otimes N';B)\\
&\qquad\times\mathcal{M}\mathbb{C}\left(\tbinom{P}{Q};\tbinom{X}{Y}\otimes\tbinom{U}{V}\right)\times\mathcal{M}\mathbb{C}\left(\tbinom{P'}{Q'};\tbinom{X'}{Y'}\otimes\tbinom{U'}{V'}\right) && \overset{\text{def}}{=}\\
&\int^{\tbinom{P}{Q},\tbinom{P'}{Q'}\in\mathcal{M}\mathbb{C},M,N,O\in\mathbb{C}} \mathbb{C}(A;M\otimes P\otimes N)\times\mathcal{M}\mathbb{C}\left(\tbinom{M\otimes Q\otimes N}{B};\tbinom{P'}{Q'}\right)\times\mathcal{M}\mathbb{C}\left(\tbinom{P}{Q};\tbinom{X}{Y}\otimes\tbinom{U}{V}\right)\times\mathcal{M}\mathbb{C}\left(\tbinom{P'}{Q'};\tbinom{X'}{Y'}\otimes\tbinom{U'}{V'}\right) && \overset{y_1}{\cong}\\
&\int^{\tbinom{P}{Q}\in\mathcal{M}\mathbb{C},M,N,O,C,D,E,F,G,H\in\mathbb{C}} \mathbb{C}(A;M\otimes P\otimes N)\times\mathbb{C}(P;C\otimes X\otimes D\otimes U\otimes E)\times\mathbb{C}(C\otimes Y\otimes D\otimes V\otimes E;Q)\times\\
&\qquad\mathbb{C}(M\otimes Q\otimes N;F\otimes X'\otimes G\otimes U'\otimes H)\times\mathbb{C}(F\otimes Y'\otimes G\otimes V'\otimes H;B) && \overset{\text{def}}{=}\\
&\int^{\tbinom{P}{Q},\in\mathcal{M}\mathbb{C},C,D,E,F,G,H\in\mathbb{C}} \mathcal{M}\mathbb{C}\left(\tbinom{A}{F\otimes X'\otimes G\otimes U''\otimes H};\tbinom{P}{Q}\right)\times\mathbb{C}(P;C\otimes X\otimes D\otimes U\otimes E)\\
&\qquad\times\mathbb{C}(C\otimes Y\otimes D\otimes V\otimes E;Q)\times\mathbb{C}(F\otimes Y'\otimes G\otimes V'\otimes H;B) && \overset{y_1}{\cong}\\
&\int^{C,D,E,F,G,H\in\mathbb{C}} \mathbb{C}(A;C\otimes X\otimes D\otimes U\otimes E)\times\mathbb{C}(C\otimes Y\otimes D\otimes V\otimes E;F\otimes X'\otimes G\otimes U'\otimes H)\\
&\qquad\times\mathbb{C}(F\otimes Y'\otimes G\otimes V'\otimes H;B).
\end{aligned}$$

This isomorphism sends an element $(j_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi j_1\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi j_2 \mid k_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi k_1 \mid l_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi l_1)$ to $\langle j_0\fatsemi(\mathrm{id}\otimes k_0\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes k_1\otimes\mathrm{id})\fatsemi j_1\fatsemi(\mathrm{id}\otimes l_0\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes l_1\otimes\mathrm{id})\fatsemi j_2\rangle$. Define a map from the left hand side of ψ_2 to this set, sending a triple

$$\begin{aligned}
&(f_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi f_1 \mid\\
&g_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi g_1\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi g_2 \mid\\
&h_0\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi h_1\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi h_2)\\
&\qquad\mapsto\\
&f_0\fatsemi(\mathrm{id}\otimes g_0\otimes\mathrm{id}\otimes h_0\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes g_1\otimes\mathrm{id}\otimes h_1\otimes\mathrm{id})\fatsemi\\
&\qquad(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi(\mathrm{id}\otimes g_2\otimes\mathrm{id}\otimes h_2\otimes\mathrm{id})\fatsemi f_1.
\end{aligned}$$

Now composing this map with the isomorphism yields the desired morphism ψ_2. The remaining laxators ψ_0, φ_2, and φ_0 are isomorphisms that arise from applications of unitality or just as identities. □

Theorem F.12 (From Theorem 6.6)**.** *Monoidal contexts are the free normalization of the cofree produoidal category over a category.*

Proof. We already know that the normalization procedure yields the free normalization over a produoidal category. It is only left to note that this is exactly the category we have explicitly constructed in this section.

This amounts to proving that the produoidal category of monoidal contexts is precisely the normalization of the produoidal category of spliced arrows. We do so for morphisms, the rest of the proof is similar.

$$\begin{aligned}
&\mathcal{NS}\mathbb{C}\left(\tbinom{A}{B};\tbinom{X}{Y}\right) && \overset{\text{def}}{=}\\
&\mathcal{S}\mathbb{C}\left(\tbinom{A}{B};N\otimes\tbinom{X}{Y}\otimes N\right) && \overset{\text{def}}{=}\\
&\int^{\tbinom{U}{V},\tbinom{U'}{V'};\in\mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B};\tbinom{U}{V}\otimes\tbinom{X}{Y}\otimes\tbinom{U'}{V'}\right)\times\mathcal{S}\mathbb{C}\left(\tbinom{U}{V};N\right)\times\mathcal{S}\mathbb{C}\left(\tbinom{U'}{V'};N\right) && \overset{\text{def}}{=}\\
&\int^{\tbinom{U}{V},\tbinom{U'}{V'}\in\mathcal{S}\mathbb{C}} \mathbb{C}(A;U\otimes X\otimes U')\times\mathbb{C}(V\otimes Y\otimes V';B)\times\mathbb{C}(U;V)\times\mathbb{C}(U';V') && \overset{\text{def}}{=}
\end{aligned}$$

$$\int^{U,V,U',V' \in \mathbb{C}} \mathbb{C}(A; U \otimes X \otimes U') \times \mathbb{C}(V \otimes Y \otimes V'; B) \times \mathbb{C}(U; V) \times \mathbb{C}(U'; V') \overset{y_1}{\cong}$$

$$\int^{U,U' \in \mathbb{C}} \mathbb{C}(A; U \otimes X \otimes U') \times \mathbb{C}(U \otimes Y \otimes U'; B) \overset{\text{def}}{=}$$

$$\mathcal{M}\mathbb{C}\binom{A; X}{B; Y}$$

The rest of the profunctors follow a similar reasoning. □

Appendix G
Monoidal Lenses

Proposition G.1 (From Proposition 7.2)**.** *Monoidal lenses form a normal symmetric produoidal category with the following morphisms, units, sequential and parallel splits.*

$$\begin{aligned}
\mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y}\right) &= \mathbb{C}(A; \bullet \otimes X) \diamond \mathbb{C}(\bullet \otimes Y; B);\\
\mathcal{L}\mathbb{C}\left(\substack{A\\B} ; N\right) &= \mathbb{C}(A; B);\\
\mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) &= \mathbb{C}(A; \bullet^1 \otimes X) \ \diamond \mathbb{C}(\bullet^1 \otimes Y; \bullet^2 \otimes X') \diamond \mathbb{C}(\bullet^2 \otimes Y'; B);\\
\mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y} \otimes \substack{X'\\Y'}\right) &= \mathbb{C}(A; \bullet^1 \otimes X \otimes X') \diamond \mathbb{C}(\bullet^1 \otimes Y \otimes Y'; B).
\end{aligned}$$

Proof. Lemmas G.2 and G.3 construct the associators, and Lemmas G.4 and G.5 define the unitors. Lemma G.6 constructs the symmetry. As they are all constructed with Yoneda isomorphisms and symmetries, they must satisfy the coherence equations. Finally, the laxators are constructed in much the same way as in Lemma F.11. □

Lemma G.2 (Monoidal lenses sequential associator)**.** *We construct a natural isomorphism*

$$(\prec_2^{\alpha}) : \int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{U\\V}\right) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X'\\Y'} \triangleleft \substack{X''\\Y''}\right) \cong \int^{\substack{U\\V} \in \mathcal{M}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{U\\V} \triangleleft \substack{X''\\Y''}\right) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) : (\prec_1^{\alpha})$$

exclusively from Yoneda isomorphisms.

Proof. Out of Yoneda reductions, we construct an isomorphism between the left hand side and a set of quadruples of morphisms.

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{X\\Y} \triangleleft \substack{U\\V}\right) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X'\\Y'} \triangleleft \substack{X''\\Y''}\right) && \overset{\text{def}}{=}\\
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}, P, Q \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathbb{C}(P \otimes Y; Q \otimes U) \times \mathbb{C}(Q \otimes V; B) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X'\\Y'} \triangleleft \substack{X''\\Y''}\right) && \overset{\text{def}}{=}\\
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}, P, Q, R \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathcal{L}\mathbb{C}\left(\substack{P \otimes Y\\B} ; \substack{U\\V}\right) \times \mathbb{C}(U; Q \otimes X') \times \mathbb{C}(Q \otimes Y'; R \otimes X'') \times \mathbb{C}(R \otimes Y''; V) && \overset{y_2}{\cong}\\
&\int^{P, Q, R \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathbb{C}(P \otimes Y; Q \otimes X') \times \mathbb{C}(Q \otimes Y'; R \otimes X'') \times \mathbb{C}(R \otimes Y''; B).
\end{aligned}$$

Out of Yoneda reductions, we construct an isomorphism between the right hand side and the same set of quadruples of morphisms.

$$\begin{aligned}
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\substack{A\\B} ; \substack{U\\V} \triangleleft \substack{X''\\Y''}\right) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) && \overset{\text{def}}{=}\\
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}, P, Q \in \mathbb{C}} \mathbb{C}(A; Q \otimes U) \times \mathbb{C}(Q \otimes V; P \otimes X'') \times \mathbb{C}(P \otimes Y''; B) \times \mathcal{L}\mathbb{C}\left(\substack{U\\V} ; \substack{X\\Y} \triangleleft \substack{X'\\Y'}\right) && \overset{\text{def}}{=}\\
&\int^{\substack{U\\V} \in \mathcal{L}\mathbb{C}, P, Q, R \in \mathbb{C}} \mathcal{L}\mathbb{C}\left(\substack{A\\P \otimes X''} ; \substack{U\\V}\right) \times \mathbb{C}(P \otimes Y''; B) \times \mathbb{C}(U; Q \otimes X) \times \mathbb{C}(Q \otimes Y; R \otimes X') \times \mathbb{C}(R \otimes Y'; V) && \overset{y_2}{\cong}\\
&\int^{P, Q, R \in \mathbb{C}} \mathbb{C}(A; Q \otimes X) \times \mathbb{C}(Q \otimes Y; R \otimes X') \times \mathbb{C}(R \otimes Y'; P \otimes X'') \times \mathbb{C}(P \otimes Y''; B).
\end{aligned}$$

Composing both isomorphisms, we obtain the desired associator. It gets defined by the following operations,

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes \blacksquare) \fatsemi f_2) \prec_1^{\alpha} (g_0 \fatsemi (\mathrm{id}_P \otimes \blacksquare) \fatsemi g_1 \fatsemi (\mathrm{id}_Q \otimes \blacksquare) \fatsemi g_2) &&=\\
&f_0 \fatsemi (\mathrm{id}_M \otimes g_0) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare) \fatsemi (\mathrm{id}_M \otimes g_1) \fatsemi (\mathrm{id}_{M \otimes Q} \otimes \blacksquare) \fatsemi (\mathrm{id}_M \otimes g_2) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes \blacksquare) \fatsemi f_2.\\
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes \blacksquare) \fatsemi f_2) \prec_2^{\alpha} (h_0 \fatsemi (\mathrm{id}_P \otimes \blacksquare) \fatsemi h_1 \fatsemi (\mathrm{id}_Q \otimes \blacksquare) \fatsemi h_2) &&=\\
&f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes h_0) \fatsemi (\mathrm{id}_{N \otimes P} \fatsemi \blacksquare) \fatsemi (\mathrm{id}_N \otimes h_1) \fatsemi (\mathrm{id}_{N \otimes Q} \otimes \blacksquare) \fatsemi (\mathrm{id}_N \otimes h_2) \fatsemi f_2.
\end{aligned}$$

□

Lemma G.3 (Monoidal lenses parallel associator)**.** *We construct a natural isomorphism*

$$(\prec_2^\alpha) : \int^{{U \atop V} \in \mathcal{M}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \otimes {U \atop V}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X' \atop Y'} \otimes {X'' \atop Y''}\right) \cong \int^{{U \atop V} \in \mathcal{M}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {U \atop V} \otimes {X'' \atop Y''}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X \atop Y} \otimes {X' \atop Y'}\right) : (\prec_1^\alpha)$$

exclusively from Yoneda isomorphisms.

Proof. The left hand side is isomorphic to:

$$\begin{aligned}
&\int^{{U \atop V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \otimes {U \atop V}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X' \atop Y'} \otimes {X'' \atop Y''}\right) && = \quad \text{(by representability)}\\
&\int^{{U \atop V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \otimes {U \atop V}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X' \otimes X'' \atop Y' \otimes Y''}\right) && \cong \quad \text{(by Yoneda reduction)}\\
&\mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \otimes {X' \otimes X'' \atop Y' \otimes Y''}\right) && \cong \quad \text{(by representability)}\\
&\mathcal{L}\mathbb{C}\left({A \atop B} ; {X \otimes X' \otimes X'' \atop Y \otimes Y' \otimes Y''}\right),
\end{aligned}$$

and the right hand side is isomorphic to the same:

$$\begin{aligned}
&\int^{{U \atop V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {U \atop V} \otimes {X'' \atop Y''}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X \atop Y} \otimes {X' \atop Y'}\right) && = \quad \text{(by representability)}\\
&\int^{{U \atop V} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {U \atop V} \otimes {X'' \atop Y''}\right) \times \mathcal{L}\mathbb{C}\left({U \atop V} ; {X \otimes X' \atop Y \otimes Y'}\right) && \cong \quad \text{(by Yoneda reduction)}\\
&\mathcal{L}\mathbb{C}\left({A \atop B} ; {X \otimes X' \atop Y \otimes Y'} \otimes {X'' \atop Y''}\right) && \cong \quad \text{(by representability)}\\
&\mathcal{L}\mathbb{C}\left({A \atop B} ; {X \otimes X' \otimes X'' \atop Y \otimes Y' \otimes Y''}\right).
\end{aligned}$$

Composing both isomorphisms, we obtain the desired associator,

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \blacksquare) \fatsemi f_1) \prec_1^\alpha (g_0 \fatsemi (\mathrm{id}_P \otimes \blacksquare \otimes \blacksquare) \fatsemi g_1) && =\\
&f_0 \fatsemi (\mathrm{id}_M \otimes g_0 \otimes \mathrm{id}_{X''}) \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \blacksquare \otimes \blacksquare) \fatsemi (\mathrm{id}_M \otimes g_1 \otimes \mathrm{id}_{Y''}) \fatsemi f_1.\\
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \blacksquare) \fatsemi f_1) \prec_2^\alpha (h_0 \fatsemi (\mathrm{id}_Q \otimes \blacksquare \otimes \blacksquare) \fatsemi h_1) && =\\
&f_0 \fatsemi \sigma \fatsemi (\mathrm{id}_M \otimes h_0 \otimes \mathrm{id}_X) \fatsemi \sigma \fatsemi (\mathrm{id}_{M \otimes P} \otimes \blacksquare \otimes \blacksquare \otimes \blacksquare) \fatsemi \sigma \fatsemi (\mathrm{id}_M \otimes h_1 \otimes \mathrm{id}_Y) \fatsemi \sigma \fatsemi f_1.
\end{aligned}$$

This concludes the proof. □

Lemma G.4 (Monoidal lenses sequential right unitor)**.** *We construct a natural isomorphism*

$$(\prec^\rho) : \int^{{X' \atop Y'} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \lhd {X' \atop Y'}\right) \times \mathcal{L}\mathbb{C}\left({X' \atop Y'} ; N\right) \cong \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y}\right)$$

exclusively from Yoneda isomorphisms.

Proof. We construct the isomorphism with the following coend calculus derivation.

$$\begin{aligned}
&\int^{{X' \atop Y'} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left({A \atop B} ; {X \atop Y} \lhd {X' \atop Y'}\right) \times \mathcal{L}\mathbb{C}\left({X' \atop Y'} ; N\right) && \overset{\text{def}}{=}\\
&\int^{{X' \atop Y'} \in \mathcal{L}\mathbb{C}, P, Q \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathbb{C}(P \otimes Y; Q \otimes X') \times \mathbb{C}(Q \otimes Y'; B) \times \mathbb{C}(X'; Y') && \overset{\text{def}}{=}\\
&\int^{{X' \atop Y'} \in \mathcal{L}\mathbb{C}, P \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathcal{L}\mathbb{C}\left({P \otimes Y \atop B} ; {X' \atop Y'}\right) \times \mathbb{C}(X'; Y') && \overset{y_2}{\cong}\\
&\int^{P \in \mathbb{C}} \mathbb{C}(A; P \otimes X) \times \mathbb{C}(P \otimes Y; B).
\end{aligned}$$

We obtain the following right unitor.

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes \blacksquare) \fatsemi f_2) \prec^{\rho} g && = \\
&f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi f_1 \fatsemi (\mathrm{id}_N \otimes g) \fatsemi f_2.
\end{aligned}$$

The left unitor is defined similarly. □

Lemma G.5 (Monoidal lenses parallel right unitor)**.** *We construct a natural isomorphism*

$$(\prec^{\rho}) : \int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \times \mathcal{L}\mathbb{C}\left(\tbinom{X'}{Y'} ; N\right) \cong \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y}\right)$$

exclusively from Yoneda isomorphisms and symmetry of $\mathbb{C}$*.*

Proof. We construct the isomorphism with the following coend calculus derivations.

$$\begin{aligned}
&\int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}} \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \times \mathcal{L}\mathbb{C}\left(\tbinom{X'}{Y'} ; N\right) \\
&= \qquad \text{(by definition)} \\
&\int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}, P \in \mathbb{C}} \mathbb{C}(A; P \otimes X \otimes X') \times \mathbb{C}(P \otimes Y \otimes Y'; B) \times \mathbb{C}(X'; Y') \\
&\cong \qquad \text{(by symmetry of } \mathbb{C}) \\
&\int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}, P \in \mathbb{C}} \mathbb{C}(A; P \otimes X' \otimes X) \times \mathbb{C}(P \otimes Y' \otimes Y; B) \times \mathbb{C}(X'; Y') \\
&\cong \qquad \text{(by Yoneda reduction)} \\
&\int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}, P,Q,R \in \mathbb{C}} \mathbb{C}(A; Q \otimes X) \times \mathbb{C}(Q; P \otimes X') \times \mathbb{C}(P \otimes Y'; R) \times \mathbb{C}(R \otimes Y; B) \times C(X'; Y') \\
&= \qquad \text{(by definition)} \\
&\int^{\binom{X'}{Y'} \in \mathcal{L}\mathbb{C}, P,Q,R \in \mathbb{C}} \mathbb{C}(A; Q \otimes X) \times \mathcal{L}\mathbb{C}\left(\tbinom{Q}{R} ; \tbinom{X'}{Y'}\right) \times \mathbb{C}(R \otimes Y; B) \times C(X'; Y') \\
&\cong \qquad \text{(by Yoneda reduction)} \\
&\int^{Q \in \mathbb{C}} \mathbb{C}(A; Q \otimes X) \times \mathbb{C}(Q \otimes Y; B).
\end{aligned}$$

We obtain the following right unitor.

$$\begin{aligned}
&(f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \blacksquare) \fatsemi f_1) \prec^{\rho} g && = \\
&f_0 \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \blacksquare) \fatsemi (\mathrm{id}_M \otimes (\sigma \fatsemi (g \otimes \mathrm{id}_X) \fatsemi \sigma)) \fatsemi f_1 && = \\
&f_0 \fatsemi (\mathrm{id}_M \otimes (\sigma \fatsemi (g \otimes \mathrm{id}_X) \fatsemi \sigma)) \fatsemi (\mathrm{id}_M \otimes \blacksquare \otimes \blacksquare) \fatsemi f_1.
\end{aligned}$$

The left unitor is defined similarly. □

Lemma G.6 (Monoidal lenses symmetry)**.** *We construct the symmetries* $\mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \cong \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X'}{Y'} \otimes \tbinom{X}{Y}\right)$.

Proof. These follow from the symmetries of $\mathbb{C}$ and representability of $\otimes$ for monoidal lenses.

$$\mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) \cong \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X \otimes X'}{Y \otimes Y'}\right) \cong \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X' \otimes X}{Y' \otimes Y}\right) \cong \mathcal{L}\mathbb{C}\left(\tbinom{A}{B} ; \tbinom{X'}{Y'} \otimes \tbinom{X}{Y}\right).$$

This concludes the proof. □

Proposition G.7 (From Proposition 7.6)**.** *Let* $(\mathbb{C}, \otimes, I)$ *be a symmetric monoidal category. There exist monoidal functors* $(\boldsymbol{!}) \colon \mathbb{C} \to \mathcal{L}\mathbb{C}$ *and* $(\boldsymbol{?}) \colon \mathbb{C}^{op} \to \mathcal{L}\mathbb{C}$*.*

Proof. This proof appears with a different language in the work of Riley [Ril18, Proposition 2.0.14]. In fact, there, the combined identity-on-objects functor $(\boldsymbol{!} \times \boldsymbol{?}) \colon \mathbb{C} \times \mathbb{C}^{op} \to \mathcal{L}\mathbb{C}$ is shown to be monoidal. In our case, we can define $\boldsymbol{!}f = (f \fatsemi \blacksquare \fatsemi \mathrm{id}_I)$ and $\boldsymbol{?}g = (\mathrm{id}_I \fatsemi \blacksquare \fatsemi g)$, and then check that compositions and tensoring of morphisms are compatible with composition and tensoring of monoidal lenses, this is straightforward. Moreover, as we comment in the text, we can see that, by definition, $\mathbf{!}(A \otimes B) = \binom{A\otimes B}{I} = \binom{A}{I} \otimes \binom{B}{I} = \mathbf{!}A \otimes \mathbf{!}B$ and $\mathbf{?}(A \otimes B) = \binom{I}{A\otimes B} = \binom{I}{A} \otimes \binom{I}{B} = \mathbf{?}A \otimes \mathbf{?}B$. □

Proposition G.8 (From Proposition 7.8)**.** *Let* $(\mathbb{C}, \times, 1)$ *be a cartesian monoidal category. Its produoidal category of lenses is given by the following profunctors.*

$$\begin{aligned}
\mathbf{Lens}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right) &= \mathbb{C}(A; X) \times \mathbb{C}(A \times Y; B). \\
\mathbf{Lens}\left(\tbinom{A}{B}; \tbinom{X}{Y} \triangleleft \tbinom{X'}{Y'}\right) &= \mathbb{C}(A; X) \times \mathbb{C}(A \times Y; X') \times \mathbb{C}(A \times Y \times Y'; B). \\
\mathbf{Lens}\left(\tbinom{A}{B}; \tbinom{X}{Y} \otimes \tbinom{X'}{Y'}\right) &= \mathbb{C}(A; X \times X') \times \mathbb{C}(A \times Y \times Y'; B). \\
\mathbf{Lens}\left(\tbinom{A}{B}\right) &= \mathbb{C}(A; B).
\end{aligned}$$

Proof. We employ coend calculus. The derivation of the morphisms of cartesian lenses is very well-known [Ril18], [CEG+20]; we derive the sequential and parallel splits. Indeed, the sequential split reduces as

$$\begin{aligned}
& \int^{M,N} \mathbb{C}(A; M \times X) \times \mathbb{C}(M \times Y; N \times X') \times \mathbb{C}(N \times Y'; B) \\
\cong \quad & \text{(Universal property of the product)} \\
& \int^{M,N} \mathbb{C}(A; M) \times \mathbb{C}(A; X) \times \mathbb{C}(M \times Y; N) \times \mathbb{C}(M \times Y; X') \times \mathbb{C}(N \times Y'; B) \\
\cong \quad & \text{(by Yoneda reduction)} \\
& \mathbb{C}(A; X) \times \mathbb{C}(A \times Y; X') \times \mathbb{C}(A \times Y \times Y'; B).
\end{aligned}$$

And the parallel split reduces as

$$\begin{aligned}
& \int^{M} \mathbb{C}(A; M \times X \times X') \times \mathbb{C}(M \times Y \times Y'; B) \\
\cong \quad & \text{(Universal property of the product)} \\
& \int^{M} \mathbb{C}(A; M) \times \mathbb{C}(A; X \times X') \times \mathbb{C}(M \times Y \times Y'; B) \\
\cong \quad & \text{(by Yoneda reduction)} \\
& \mathbb{C}(A; X \times X') \times \mathbb{C}(A \times Y \times Y'; B).
\end{aligned}$$

The unit is just the same as in the general monoidal case. □

Theorem G.9 (From Theorem 7.3)**.** *Monoidal lenses are the free symmetric normalization of the cofree symmetric produoidal category over a monoidal category.*

Proof. We have already proven that the symmetric normalization procedure yields the free symmetric normalization over a symmetric produoidal category (Theorem 5.7).

The rest of the proof amounts to show that the normal symmetric produoidal category of monoidal lenses is precisely the symmetric normalization of the produoidal category of spliced arrows. We do so for morphisms, the rest of the proof is similar.

$$\begin{aligned}
& \mathcal{N}_\sigma \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{X}{Y}\right) && \overset{\text{def}}{=} \\
& \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; N \otimes \tbinom{X}{Y}\right) && \overset{\text{def}}{=} \\
& \int^{\binom{U}{V} \in \mathcal{S}\mathbb{C}} \mathcal{S}\mathbb{C}\left(\tbinom{A}{B}; \tbinom{U}{V} \otimes \tbinom{X}{Y}\right) \times \mathcal{S}\mathbb{C}\left(\tbinom{U}{V}; N\right) && \overset{\text{def}}{=} \\
& \int^{U,V \in \mathbb{C}} \mathbb{C}(A; U \otimes X) \times \mathbb{C}(V \otimes Y; B) \times \mathbb{C}(U; V) && \overset{y_1}{\cong} \\
& \int^{U \in \mathbb{C}} \mathbb{C}(A; U \otimes X) \times \mathbb{C}(U \otimes Y; B) && \overset{\text{def}}{=}
\end{aligned}$$

$$\mathcal{L}\mathbb{C}\left(\substack{A\\B};\substack{X\\Y}\right)$$

The rest of the profunctors follow a similar reasoning. $\square$

Appendix H
Further Work

Theorem H.1 (From Proposition 8.1)**.** *Let $\mathbb{V}$ be a normal and $\otimes$-symmetric produoidal category with coends over $\mathbb{V}$ commuting with finite connected limits. Then, $[\mathbb{V}^{\mathrm{op}}, \mathbf{Set}]$ is a dependence category in the sense of Shapiro and Spivak [SS22].*

Proof. Whenever $\mathbb{V}$ is produoidal, $[\mathbb{V}^{\mathrm{op}}, \mathbf{Set}]$, its category of presheaves is duoidal, with the structure given by convolution (Theorem I.6).

At the same time, $[\mathbb{V}^{\mathrm{op}}, \mathbf{Set}]$ is a locally cartesian closed category will all limits because it is a presheaf category. Whenever finite connected limits are preserved by $\otimes, \triangleleft$, we obtain a dependence category [SS22, Theorem 4.8]. This means we only need the following isomorphism,

$$\begin{aligned}
&\int^{U,V} \mathbb{V}(X; U \otimes V) \times \lim_i P_i(U) \times \lim_j Q_j(V) \\
\cong \quad & \text{(Commutation of limits)} \\
&\int^{U,V} \lim_{i,j} \mathbb{V}(X; U \otimes V) \times P_i(U) \times Q_j(V) \\
\cong \quad & \text{(Coends commute with finite connected limits)} \\
&\lim_{i,j} \int^{U,V} \mathbb{V}(X; U \otimes V) \times P_i(U) \times Q_j(V)
\end{aligned}$$

Where we use our hypothesis on the last step. We conjecture this can be extended to an arbitrary $\mathbb{V}$ with minor constraints. □

Appendix I
Duoidal and Produoidal Categories

By the Eckmann-Hilton argument, each time we have two monoids $(*, \circ)$ such that one is a monoid homomorphism over the other, $(a \circ b) * (c \circ d) = (a * c) \circ (b * d)$, we know that both monoids coincide into a single commutative monoid.

However, an extra dimension helps us side-step the Eckmann-Hilton argument. If, instead of equalities or isomorphisms, we use directed morphisms, both monoids (which now may become 2-monoids) do not necessarily coincide, and the resulting structure is that of a duoidal category.

Definition I.1 (Duoidal category)**.** A *duoidal category* [AM10] is a category $\mathbb{C}$ with two monoidal structures, $(\mathbb{C}, \otimes, I, \alpha, \lambda, \rho)$ and $(\mathbb{C}, \triangleleft, N, \beta, \kappa, \nu)$ such that the latter distribute over the former. In other words, it is endowed with a duoidal tensor, $(\triangleleft) \colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, together with natural distributors

$$\psi_2 \colon (X \triangleleft Z) \otimes (Y \triangleleft W) \to (X \otimes Y) \triangleleft (Z \otimes W), \qquad \psi_0 \colon I \to I \triangleleft I, \qquad \varphi_2 \colon N \otimes N \to N, \quad \text{and} \quad \varphi_0 \colon I \to N,$$

satisfying the following coherence equations (Figures 32 to 36).

Remark I.2. In other words, the duoidal tensor and unit are lax monoidal functors for the first monoidal structure, which means that the laxators must satisfy the following equations.

1) $(\psi_2 \otimes id) ⨾ \psi_2 ⨾ (\alpha \triangleleft \alpha) = \alpha ⨾ (id \otimes \psi_2) ⨾ \psi_2$, for the associator;
2) $(\psi_0 \otimes id) ⨾ \psi_2 ⨾ (\lambda \triangleleft \lambda) = \lambda$, for the left unitor; and
3) $(id \otimes \psi_0) ⨾ \psi_2 ⨾ (\rho \triangleleft \rho) = \rho$, for the right unitor;
4) $\alpha ⨾ (id \otimes \varphi_2) ⨾ \varphi_2 = (\varphi_2 \otimes id) ⨾ \varphi_2$, for the associator;
5) $(\varphi_0 \otimes id) ⨾ \varphi_2 = \lambda$, for the left unitor; and
6) $(id \otimes \varphi_0) ⨾ \varphi_2 = \rho$, for the right unitor.

Theorem I.3 (Coherence, [AM10])**.** *Any two parallel morphisms constructed out of the coherence isomorphisms and laxators of a duoidal category coincide.*

$$\begin{tikzcd}
((A \triangleleft B) \otimes (C \triangleleft D)) \otimes (E \triangleleft F) \arrow[r, "\alpha"] \arrow[d, "\psi_2 \otimes id"'] & (A \triangleleft B) \otimes ((C \triangleleft D) \otimes (E \triangleleft F)) \arrow[d, "id \otimes \psi_2"] \\
((A \otimes C) \triangleleft (B \otimes D)) \otimes (E \triangleleft F) \arrow[d, "\psi_2"'] & (A \triangleleft B) \otimes ((C \otimes E) \triangleleft (D \otimes F)) \arrow[d, "\psi_2"] \\
((A \otimes C) \otimes E) \triangleleft ((B \otimes D) \otimes F) \arrow[r, "\alpha \triangleleft \alpha"] & (A \otimes (C \otimes E)) \triangleleft (B \otimes (D \otimes F))
\end{tikzcd}$$

$$\begin{tikzcd}
((A \triangleleft B) \triangleleft C) \otimes ((D \triangleleft E) \triangleleft F) \arrow[r, "\beta \otimes \beta"] \arrow[d, "\psi_2"'] & (A \triangleleft (B \triangleleft C)) \otimes (D \triangleleft (E \triangleleft F)) \arrow[d, "\psi_2"] \\
((A \triangleleft B) \otimes (D \triangleleft E)) \triangleleft (C \otimes F) \arrow[d, "\psi_2 \otimes id"'] & (A \otimes D) \triangleleft ((B \triangleleft C) \otimes (E \triangleleft F)) \arrow[d, "id \otimes \psi_2"] \\
((A \otimes D) \triangleleft (B \otimes E)) \triangleleft (C \otimes F) \arrow[r, "\beta"] & (A \otimes D) \triangleleft ((B \otimes E) \triangleleft (C \otimes F))
\end{tikzcd}$$

Fig. 32: Coherence diagrams for associativity of a duoidal category.

$$\begin{tikzcd}
I \otimes (A \triangleleft B) \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\lambda"'] & (I \triangleleft I) \otimes (A \triangleleft B) \arrow[d, "\psi_2"] & (A \triangleleft B) \otimes I \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\rho"'] & (A \triangleleft B) \otimes (I \triangleleft I) \arrow[d, "\psi_2"] \\
A \triangleleft B & (I \otimes A) \triangleleft (I \otimes B) \arrow[l, "\lambda \triangleleft \lambda"] & A \triangleleft B & (A \otimes I) \triangleleft (B \otimes I) \arrow[l, "\rho \triangleleft \rho"]
\end{tikzcd}$$

Fig. 33: Coherence diagrams for $\otimes$-unitality of a duoidal category.

$$\begin{tikzcd}
N \triangleleft (A \otimes B) \arrow[d, "\kappa"'] & (N \otimes N) \triangleleft (A \otimes B) \arrow[l, "\varphi_2 \triangleleft id"'] \arrow[d, "\psi_2"] & (A \otimes B) \triangleleft N \arrow[d, "\nu"'] & (A \otimes B) \triangleleft (N \otimes N) \arrow[l, "id \triangleleft \varphi_2"'] \arrow[d, "\psi_2"] \\
A \otimes B & (N \triangleleft A) \otimes (N \triangleleft B) \arrow[l, "\kappa \otimes \kappa"] & A \otimes B & (A \triangleleft N) \otimes (B \triangleleft N) \arrow[l, "\nu \otimes \nu"]
\end{tikzcd}$$

Fig. 34: Coherence diagrams for $\triangleleft$-unitality of a duoidal category.

$$\begin{tikzcd}
(N \otimes N) \otimes N \arrow[rr, "\alpha"] \arrow[d, "\varphi_2 \otimes id"'] & & N \otimes (N \otimes N) \arrow[d, "id \otimes \varphi_2"] \\
N \otimes N \arrow[r, "\varphi_2"'] & N & N \otimes N \arrow[l, "\varphi_2"]
\end{tikzcd}
\qquad
\begin{tikzcd}
I \triangleleft I \arrow[d, "\psi_0 \otimes id"'] & I \arrow[l, "\psi_0"'] \arrow[r, "\psi_0"] & I \triangleleft I \arrow[d, "id \otimes \psi_0"] \\
(I \triangleleft I) \triangleleft I \arrow[rr, "\beta"'] & & I \triangleleft (I \triangleleft I)
\end{tikzcd}$$

Fig. 35: Associativity and coassociativity for N and I in a duoidal category.

$$\begin{tikzcd}
N \otimes I \arrow[r, "\rho"] \arrow[d, "id \otimes \varphi_0"'] & N \\
N \otimes N \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\quad
\begin{tikzcd}
I \otimes N \arrow[r, "\lambda"] \arrow[d, "\varphi_0 \otimes id"'] & N \\
N \otimes N \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\quad
\begin{tikzcd}
I \triangleleft N \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\nu"'] & I \triangleleft I \\
I \arrow[ur, "\psi_0"'] &
\end{tikzcd}
\quad
\begin{tikzcd}
N \triangleleft I \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\kappa"'] & I \triangleleft I \\
I \arrow[ur, "\psi_0"'] &
\end{tikzcd}$$

Fig. 36: Unitality and counitality for N and I in a duoidal category.

I.1 Normalization of duoidal categories

Garner and López Franco [GF16] introduce a procedure for normalizing a sufficiently well-behaved duoidal category, based in the construction of a new duoidal category of *bimodules*. In this text, we introduce a normalization procedure for an arbitrary produoidal category. For completeness, let us recall first the original procedure [GF16].

Let M be a bimonoid in the duoidal category $(\mathbb{V}, \otimes, I, \triangleleft, N)$, with maps $e \colon I \to M$ and $m \colon M \otimes M \to M$; and with maps $u \colon M \to N$ and $d \colon M \to M \triangleleft M$. Consider now the category of $M^{\otimes}$-bimodules. This category has a monoidal structure lifted from $(\mathbb{V}, \triangleleft, N)$:

1) the unit, N, has a bimodule structure with

$$M \otimes N \otimes M \overset{u \otimes \mathrm{id} \otimes u}{\longrightarrow} N \otimes N \otimes N \longrightarrow N;$$

2) the sequencing of two $M^{\otimes}$-bimodules is a $M^{\otimes}$-bimodule with

$$\begin{aligned} & M \otimes (A \triangleleft B) \otimes M \\ & \quad \to (M \triangleleft M) \otimes (A \triangleleft B) \otimes (M \triangleleft M) \\ & \quad \to (M \otimes A \otimes M) \triangleleft (M \otimes B \otimes M) \to A \triangleleft B. \end{aligned}$$

Moreover, whenever $\mathbb{V}$ admits reflexive coequalizers preserved by $(\otimes)$, the category of $M^{\otimes}$-bimodules is monoidal with the tensor of bimodules: the coequalizer

$$A \otimes M \otimes B \rightrightarrows A \otimes B \twoheadrightarrow A \otimes_M B.$$

In this case $(\mathbf{Bimod}^{\otimes}_M, \otimes_M, M, \triangleleft, N)$ is a duoidal category.

Theorem I.4 (Normalization of a duoidal category)**.** *Let* $(\mathbb{V}, \otimes, I, \triangleleft, N)$ *be a duoidal category with reflexive coequalizers preserved by* $(\otimes)$*. The category of* N*-bimodules is then a normal duoidal category,*

$$\mathcal{N}(\mathbb{V}) = (\mathbf{Bimod}^{\otimes}_N, \otimes_N, N, \triangleleft, N).$$

We call this category the normalization *[GF16] of the duoidal category* $\mathbb{V}$*.*

I.2 Produoidal Categories

Definition I.5 (Produoidal category, from Definition 4.2)**.** A *produoidal category* is a category $\mathbb{V}$ endowed with two promonoidal structures,

$$\begin{aligned} \mathbb{V}(\bullet; \bullet \otimes \bullet) &\colon \mathbb{V} \times \mathbb{V} \bullet\!\!-\!\!\circ \mathbb{V}, \text{ and } \mathbb{V}(\bullet; I) \colon 1 \bullet\!\!-\!\!\circ \mathbb{V}, \\ \mathbb{V}(\bullet; \bullet \triangleleft \bullet) &\colon \mathbb{V} \times \mathbb{V} \bullet\!\!-\!\!\circ \mathbb{V}, \text{ and } \mathbb{V}(\bullet; N) \colon 1 \bullet\!\!-\!\!\circ \mathbb{V}, \end{aligned}$$

such that one laxly distributes over the other. This is to say that it is endowed with the following natural *laxators*,

$$\begin{aligned} \psi_2 &\colon \mathbb{V}(\bullet; (X \triangleleft Y) \otimes (Z \triangleleft W)) \to \mathbb{V}(\bullet; (X \otimes Z) \triangleleft (Y \otimes W)), \\ \psi_0 &\colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; I \triangleleft I), \\ \varphi_2 &\colon \mathbb{V}(\bullet; N \otimes N) \to \mathbb{V}(\bullet; N), \\ \varphi_0 &\colon \mathbb{V}(\bullet; I) \to \mathbb{V}(\bullet; N). \end{aligned}$$

Laxators, together with unitors and associators must satisfy the coherence conditions in the following diagrams (Figures 37 to 41).

$$\begin{tikzcd}
\mathbb{V}(\bullet, ((A \triangleleft B) \otimes (C \triangleleft D)) \otimes (E \triangleleft F)) \arrow[r, "\alpha"] \arrow[d, "\psi_2 \otimes id"'] & \mathbb{V}(\bullet, (A \triangleleft B) \otimes ((C \triangleleft D) \otimes (E \triangleleft F))) \arrow[d, "id \otimes \psi_2"] \\
\mathbb{V}(\bullet, ((A \otimes C) \triangleleft (B \otimes D)) \otimes (E \triangleleft F)) \arrow[d, "\psi_2"'] & \mathbb{V}(\bullet, (A \triangleleft B) \otimes ((C \otimes E) \triangleleft (D \otimes F))) \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, ((A \otimes C) \otimes E) \triangleleft ((B \otimes D) \otimes F)) \arrow[r, "\alpha \triangleleft \alpha"] & \mathbb{V}(\bullet, (A \otimes (C \otimes E)) \triangleleft (B \otimes (D \otimes F)))
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbb{V}(\bullet, ((A \triangleleft B) \triangleleft C) \otimes ((D \triangleleft E) \triangleleft F)) \arrow[r, "\beta \otimes \beta"] \arrow[d, "\psi_2"'] & \mathbb{V}(\bullet, (A \triangleleft (B \triangleleft C)) \otimes (D \triangleleft (E \triangleleft F))) \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, ((A \triangleleft B) \otimes (D \triangleleft E)) \triangleleft (C \otimes F)) \arrow[d, "\psi_2 \otimes id"'] & \mathbb{V}(\bullet, (A \otimes D) \triangleleft ((B \triangleleft C) \otimes (E \triangleleft F))) \arrow[d, "id \otimes \psi_2"] \\
\mathbb{V}(\bullet, ((A \otimes D) \triangleleft (B \otimes E)) \triangleleft (C \otimes F)) \arrow[r, "\beta"] & \mathbb{V}(\bullet, (A \otimes D) \triangleleft ((B \otimes E) \triangleleft (C \otimes F)))
\end{tikzcd}$$

Fig. 37: Coherence diagrams for associativity of a produoidal category.

$$\begin{tikzcd}
\mathbb{V}(\bullet, I \otimes (A \triangleleft B)) \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\lambda"'] & \mathbb{V}(\bullet, (I \triangleleft I) \otimes (A \triangleleft B)) \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, A \triangleleft B) & \mathbb{V}(\bullet, (I \otimes A) \triangleleft (I \otimes B)) \arrow[l, "\lambda \triangleleft \lambda"']
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{V}(\bullet, (A \triangleleft B) \otimes I) \arrow[r, "\psi_0 \otimes id"] \arrow[d, "\rho"'] & \mathbb{V}(\bullet, (A \triangleleft B) \otimes (I \triangleleft I)) \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, A \triangleleft B) & \mathbb{V}(\bullet, (A \otimes I) \triangleleft (B \otimes I)) \arrow[l, "\rho \triangleleft \rho"']
\end{tikzcd}$$

Fig. 38: Coherence diagrams for ⊗-unitality of a produoidal category.

$$\begin{tikzcd}
\mathbb{V}(\bullet, N \triangleleft (A \otimes B)) \arrow[d, "\kappa"'] & \mathbb{V}(\bullet, (N \otimes N) \triangleleft (A \otimes B)) \arrow[l, "\varphi_2 \triangleleft id"'] \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, A \otimes B) & \mathbb{V}(\bullet, (N \triangleleft A) \otimes (N \triangleleft B)) \arrow[l, "\kappa \otimes \kappa"']
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbb{V}(\bullet, (A \otimes B) \triangleleft N) \arrow[d, "\nu"'] & \mathbb{V}(\bullet, (A \otimes B) \triangleleft (N \otimes N)) \arrow[l, "id \triangleleft \varphi_2"'] \arrow[d, "\psi_2"] \\
\mathbb{V}(\bullet, A \otimes B) & \mathbb{V}(\bullet, (A \triangleleft N) \otimes (B \triangleleft N)) \arrow[l, "\nu \otimes \nu"']
\end{tikzcd}$$

Fig. 39: Coherence diagrams for ◁-unitality of a produoidal category.

$$\begin{tikzcd}
\mathbb{V}(\bullet, (N \otimes N) \otimes N) \arrow[rr, "\alpha"] \arrow[d, "\varphi_2 \otimes id"'] & & \mathbb{V}(\bullet, N \otimes (N \otimes N)) \arrow[d, "id \otimes \varphi_2"] \\
\mathbb{V}(\bullet, N \otimes N) \arrow[r, "\varphi_2"'] & \mathbb{V}(\bullet, N) & \mathbb{V}(\bullet, N \otimes N) \arrow[l, "\varphi_2"]
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbb{V}(\bullet, I \triangleleft I) \arrow[d, "\psi_0 \otimes id"'] & \mathbb{V}(\bullet, I) \arrow[l, "\psi_0"'] \arrow[r, "\psi_0"] & \mathbb{V}(\bullet, I \triangleleft I) \arrow[d, "id \otimes \psi_0"] \\
\mathbb{V}(\bullet, (I \triangleleft I) \triangleleft I) \arrow[rr, "\beta"'] & & \mathbb{V}(\bullet, I \triangleleft (I \triangleleft I))
\end{tikzcd}$$

Fig. 40: Associativity and coassociativity for N and I in a produoidal category.

$$\begin{tikzcd}
\mathbb{V}(\bullet, N \otimes I) \arrow[r, "\rho"] \arrow[d, "id \otimes \varphi_0"'] & \mathbb{V}(\bullet, N) \\
\mathbb{V}(\bullet, N \otimes N) \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{V}(\bullet, I \otimes N) \arrow[r, "\lambda"] \arrow[d, "\varphi_0 \otimes id"'] & \mathbb{V}(\bullet, N) \\
\mathbb{V}(\bullet, N \otimes N) \arrow[ur, "\varphi_2"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{V}(\bullet, I \triangleleft N) \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\nu"'] & \mathbb{V}(\bullet, I \triangleleft I) \\
\mathbb{V}(\bullet, I) \arrow[ur, "\psi_0"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
\mathbb{V}(\bullet, N \triangleleft I) \arrow[r, "id \otimes \varphi_0"] \arrow[d, "\kappa"'] & \mathbb{V}(\bullet, I \triangleleft I) \\
\mathbb{V}(\bullet, I) \arrow[ur, "\psi_0"'] &
\end{tikzcd}$$

Fig. 41: Unitality and counitality for N and I in a produoidal category.

I.3 Produoidals induce duoidals

Theorem I.6. *Let $\mathbb{V}$ be a produoidal category, then its category of presheaves,* $[\mathbb{V}^{\mathrm{op}}, \mathbf{Set}]$*, is duoidal with the structure given by convolution [BS13].*

Proof. Let P and Q be presheaves in $\mathbb{V}$. We define the following tensor products on presheaves by convolution of the tensor products in $\mathbb{V}$.

$$(P \otimes Q)(A) = \int^{U,V} \mathrm{hom}(A, U \otimes V) \times P(U) \times Q(V),$$

$$(P \triangleleft Q)(A) = \int^{U,V} \mathrm{hom}(A, U \triangleleft V) \times P(U) \times Q(V).$$

These tensor products can be shown in a straightforward way to form a duoidal category, inheriting the laxators from those of $\mathbb{V}$. □

Appendix J
Tambara modules

Definition J.1 (Tambara module, [PS07]). Let $(\mathbb{A}, \otimes, I)$ be a strict monoidal category. A *Tambara module* is a profunctor $T \colon \mathbb{A}^{\mathrm{op}} \times \mathbb{A} \to \mathbf{Set}$ endowed with natural transformations

$$t_l^M : T(X;Y) \to T(M \otimes X, M \otimes Y),$$
$$t_r^M : T(X;Y) \to T(X \otimes M, Y \otimes M),$$

that are natural in both X and Y, but also dinatural on M. These must moreover satisfy the following axioms:

- $t_l^I = id$ and $t_r^I = id$, unitality;
- $t_l^M \mathbin{⨟} t_l^N = t_l^{N \otimes M}$ and $t_r^M \mathbin{⨟} t_r^N = t_l^{M \otimes N}$, multiplicativity;
- $t_l^M \mathbin{⨟} t_r^N = t_r^N \mathbin{⨟} t_l^M$, and compatibility.

Tambara modules are the algebras of a monad. We start by noting that the hom profunctor is a monoid with respect to Day convolution. This makes the following functor a monad on endoprofunctors, the so-called Pastro-Street monad [PS07],

$$\Phi(P) = hom \circledast P \circledast hom;$$

where $\Phi \colon [\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}] \to [\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}]$.

Theorem J.2. *The algebras of the Pastro-Street monad, the Φ-algebras, are precisely Tambara modules [PS07]. As a consequence, the* free Tambara module *over a profunctor $H \colon \mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbf{Set}$ is $\Phi(H)$.*

Example J.3. Consider the profunctor $\text{ꝁ}(A;B) \colon \mathbb{A}^{\mathrm{op}} \times \mathbb{A} \to \mathbf{Set}$ that produes a hole of types A and B. That is, let $\text{ꝁ}(A;B) = \mathrm{hom}(\bullet, A) \times \mathrm{hom}(B, \bullet)$. The free Tambara module over it is the monoidal context with a hole of type A and B,

$$\Phi(\text{ꝁ}{}^A_B) = \int^{M,N} \mathrm{hom}(\bullet, M \otimes A \otimes N) \times \mathrm{hom}(M \otimes B \otimes N, \bullet).$$

J.1 Normalization of profunctors

Let $(\mathbb{C}, \otimes, I)$ be a monoidal category. The category of endoprofunctors $\mathbb{C}^{\mathrm{op}} \times \mathbb{C} \to \mathbf{Set}$ is then duoidal with composition $(\triangleleft)$ and Day convolution $(\circledast)$.

$$(\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}, \circledast, I, \triangleleft, \mathrm{hom}).$$

Moreover, we can also construct its normalization: the category of endoprofunctors, $[\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}]$, has reflexive coequalisers; thus, we are in the conditions of Theorem I.4. The normal duoidal category of $\mathrm{hom}^{\circledast}$-bimodules has been traditionally called the category of *Tambara modules*.

$$\mathcal{N}(\mathbb{C}^{\mathrm{op}} \times \mathbb{C}, \mathbf{Set}, \circledast, I, \triangleleft, \mathrm{hom}) = (\mathbf{Tamb}, \circledast_{\mathrm{hom}}, \mathrm{hom}, \triangleleft, \mathrm{hom}).$$

Theorem J.4. *The category of Tambara modules is a normal duoidal category and, in fact, it is the normalization of the duoidal category of endoprofunctors.*

Appendix K
Monoidal Categories

K.1 Monoidal categories.

Endowed with the notion of isomorphism, we can now relax our definition of theory of processes by substituting strict equalities by isomorphism.

Definition K.1. A {symmetric} monoidal category [Mac78] $(\mathbb{C}, \otimes, I)$ is a tuple

$$(\mathbb{C}_{\text{obj}}, \mathbb{C}_{\text{mor}}, (\fatsemi), \text{id}, (\otimes)_{\text{obj}}, (\otimes)_{\text{mor}}, I, \alpha, \lambda, \rho, \{\sigma\}),$$

specifying a set of objects, or resource types, $\mathbb{C}_{\text{obj}}$; a set of morphisms, or processes, $\mathbb{C}_{\text{mor}}$; a composition operation; a family of identity morphisms; a tensor operation on objects and morphisms; a unit object and families of associator, left unitor, right unitor {and swapping morphisms}.

The families of associator, left unitor and right unitor morphisms have the following types.

$$\begin{aligned}\alpha_{A,B,C} &\colon (A \otimes B) \otimes C \to A \otimes (B \otimes C),\\ \lambda_A &\colon I \otimes A \to A,\\ \rho_A &\colon A \otimes I \to A.\end{aligned}$$

They must satisfy the following non-strict versions of the axioms.

$$A \otimes (B \otimes C) \cong (A \otimes B) \otimes C, \tag{1}$$
$$A \otimes I \cong A \cong I \otimes A, \tag{2}$$
$$(f \fatsemi g) \fatsemi h = f \fatsemi (g \fatsemi h), \tag{3}$$
$$\text{id}_B \fatsemi f = f = f \fatsemi \text{id}_B, \tag{4}$$
$$(f \otimes (g \otimes h)) \fatsemi \alpha = \alpha \fatsemi ((f \otimes g) \otimes h), \tag{5}$$
$$(f \otimes \text{id}_I) \fatsemi \rho = \rho \fatsemi f, \tag{6}$$
$$(f \otimes g) \fatsemi (h \otimes k) = (f \fatsemi h) \otimes (g \fatsemi k), \tag{7}$$
$$\sigma_{A,B\otimes C} \fatsemi \alpha = \alpha \fatsemi (\sigma_{A,B} \fatsemi \text{id}_C) \fatsemi (\text{id}_B \otimes \sigma_{A,C}), \tag{8}$$
$$\sigma_{A,B\otimes C} \fatsemi \alpha = \alpha \fatsemi (\sigma_{A,B} \fatsemi \text{id}_C) \fatsemi (\text{id}_B \otimes \sigma_{A,C}), \tag{9}$$
$$\sigma_{A,A'} \fatsemi (g \otimes f) = (f \otimes g) \fatsemi \sigma_{B,B'}, \tag{10}$$
$$\sigma_{A,B} \fatsemi \sigma_{B,A} = \text{id}_{A\otimes B}. \tag{11}$$

{Additionally}, they must satisfy the following axioms, whenever they are formally well-typed.

$$\alpha \fatsemi \alpha = (\alpha \otimes \text{id}) \fatsemi \alpha \fatsemi (\text{id} \otimes \alpha), \tag{12}$$
$$\rho = \alpha \fatsemi (\text{id} \otimes \lambda), \tag{13}$$
$$\alpha \fatsemi \sigma \fatsemi \alpha = (\sigma \otimes \text{id}) \fatsemi \alpha \fatsemi (\text{id} \otimes \sigma). \tag{14}$$

String diagrams [JS91] are a sound and complete syntax for monoidal categories.

Construction K.2. *Let $\mathbb{C}$ be a monoidal category. Its strictification,* **Strict**$(\mathbb{C})$*, is a monoidal category where*

- *objects are cliques: for each list of objects of $\mathbb{C}$, say, $[A_0, \dots, A_n] \in$* **List**$(\mathbb{C})$*, we form the clique containing all possible parenthesizations and coherence isomorphisms between them;*
- *morphisms are clique morphisms: a morphism between any two components of the clique, which determines a morphism between all of them.*

The tensor product is concatenation, which makes it a strict monoidal category.

Remark K.3. There is a strong monoidal functor $\mathbb{C} \to$ **Strict**$(\mathbb{C})$, this makes an object A into an object $[A]$; this is fully-faithful but, moreover, it is essentially surjective, giving a monoidal equivalence.

Theorem K.4. *Every monoidal category is monoidally equivalent to its strictification.*

5. Open Diagrams via Coend Calculus

Mario Román

Applied Category Theory (ACT, 2020)

Abstract: Morphisms in a monoidal category are usually interpreted as processes, and graphically depicted as square boxes. In practice, we are faced with the problem of interpreting what non-square boxes ought to represent in terms of the monoidal category and, more importantly, how should they be composed. Examples of this situation include lenses or learners. We propose a description of these non-square boxes, which we call open diagrams, using the monoidal bicategory of profunctors. A graphical coend calculus can then be used to reason about open diagrams and their compositions.

Declaration: *Hereby I declare that my contribution to this manuscript was to: write the manuscript as a single-author, identify the research problem and conduct the research.*

OPEN DIAGRAMS VIA COEND CALCULUS

MARIO ROMÁN

Abstract. Morphisms in a monoidal category are usually interpreted as *processes*, and graphically depicted as square boxes. In practice, we are faced with the problem of interpreting what *non-square boxes* ought to represent in terms of the monoidal category and, more importantly, how should they be composed. Examples of this situation include *lenses* or *learners*. We propose a description of these non-square boxes, which we call *open diagrams*, using the monoidal bicategory of profunctors. A graphical coend calculus can then be used to reason about open diagrams and their compositions.

1. Introduction

1.1. **Open Diagrams.** Morphisms in monoidal categories are interpreted as processes with inputs and outputs and generally represented by square boxes. This interpretation, however, raises the question of how to represent a process that does not consume all the inputs at the same time or a process that does not produce all the outputs at the same time. For instance, consider a process that consumes an input, produces an output, then consumes a second input and ends producing an output. Graphically, we have a clear idea of how this process should be represented, even if it is not a morphism in the category.

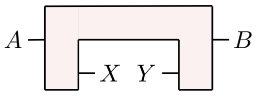


Figure 1. A process with a non-standard shape. The input A is taken at the beginning, then the output X is produced, strictly after that, the input Y is taken; finally, the output B is produced.

Reasoning graphically, it seems clear, for instance, that we should be able to *plug* a morphism connecting the first output to the second input inside this process and get back an actual morphism of the category.

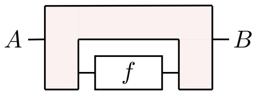


Figure 2. It is possible to plug a morphism $f\colon X \to Y$ inside the previous process (Figure 1), and, importantly, get back a morphism $A \to B$.

The particular shape depicted above has been extensively studied by [Ril18] under the name of (monoidal) *optic*; it can be also called a *monoidal lens*; and it has applications in bidirectional data accessing [PGW17, BG18, Kme18] or compositional game theory [GHWZ18]. A multi-legged generalization has appeared also in quantum circuit design

Date: May 10, 2020.

[CDP08] and quantum causality [KU17] as a notational convention, see [Rom20]. It can be shown that boxes of that particular shape should correspond to elements of a suitable *coend* (Figure 3, see also §1.2 and [Mil17, Ril18]). The intuition for this coend representation is to first consider a tuple of morphisms, and then quotient out by the equivalence relation generated by sliding morphisms along connected wires.

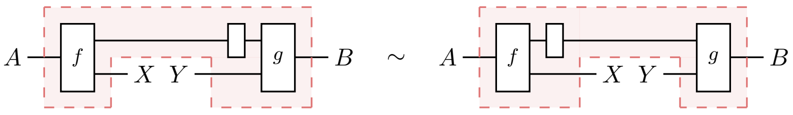


FIGURE 3. A box of this shape is meant to represent a pair of morphisms in a monoidal category quotiented out by "sliding a morphism" over the upper wire.

It has remained unclear, however, how this process should be carried in full generality and if it was in solid ground. Are we being formal when we use these *open* or *incomplete* diagrams? What happens with all the other possible shapes that one would want to consider in a monoidal category? In principle, they are not usual squares. For instance, the second of the shapes in Figure 4 has three inputs and two outputs, but the first input cannot affect the last output; and the last input cannot affect the first output.[1]

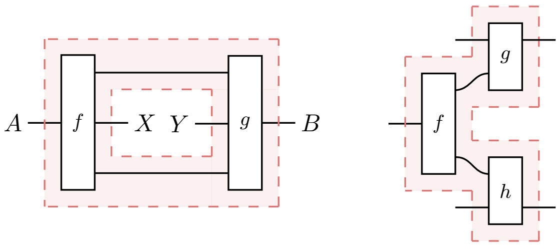


FIGURE 4. Some other shapes for boxes in a monoidal category.

This text presents the idea that incomplete diagrams should be interpreted as valid diagrams in the monoidal bicategory of profunctors; and that compositions of incomplete diagrams correspond to reductions that employ the monoidal bicategory structure. At the same time, this gives a graphical presentation of *coend calculus*.

1.2. **Coend calculus.** Coends are particular cases of colimits and *coend calculus* is a practical formalism that uses Yoneda reductions to describe isomorphisms between them. Their dual counterparts are *ends*, and formalisms for both interact nicely in a *(Co)End calculus* [Lor19].

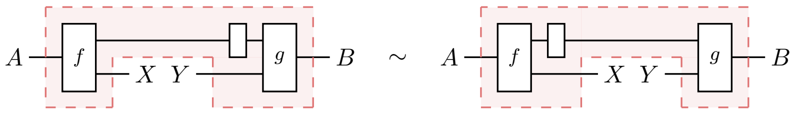
Definition 1.1. The **coend** $\int^{X\in\mathbf{C}} P(X,X)$ of a profunctor $P\colon \mathbf{C}^{op}\times\mathbf{C}\to\mathbf{Set}$ is the coequalizer of the action of morphisms on both arguments of the profunctor.

$$\int^{X\in\mathbf{C}} P(X,X) \cong \mathrm{coeq}\left(\bigsqcup_{f\colon B\to A} P(A,B) \rightrightarrows \bigsqcup_{X\in\mathbf{C}} P(X,X) \right).$$

An element of the coend is an equivalence class of pairs $[X, x\in P(X,X)]$ under the equivalence relation generated by $[X, P(f,-)(z)] \sim [Y, P(-,f)(z)]$ for each $f : Y\to X$.

[1]This particular shape comes from a question by Nathaniel Virgo on `categorytheory.zulipchat.com`.

Our main idea is to use these equivalence relations to deal with the quotienting arising in non-square monoidal boxes.



$$\int^{M} \mathbf{C}(A, M \otimes X) \times \mathbf{C}(M \otimes Y, B).$$

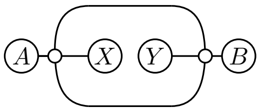
FIGURE 5. We can go back to the previous example (Figure 3) to check how it coincides with the quotienting arising from the dinaturality of a coend.

1.3. **Contributions.** Our first contribution is a graphical calculus of *shapes* of open diagrams (§2), with semantics on the monoidal bicategory of profunctors, and with an emphasis on representing monoidal structures. We show how to compose and simplify shapes (§3). Our second contribution is a graphical calculus with *open diagrams,* in terms of the category of pointed profunctors, and hinting at a pseudofunctorial analogue of *functor boxes* [Mel06] (§4).

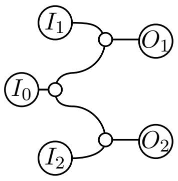
As examples, we recast the multiple ways of composing *monoidal lenses* and other coend constructions on the literature on optics (§2.3). We study categories with feedback (§2.4) and *learners* (§8.4).

2. Shapes of Open Diagrams

In the same sense that morphisms sharing the same domain and codomain are collected into an hom-set; open diagrams sharing the same *shape* will be collected into a set. Our first step is to provide a graphical calculus for these shapes and, at the same time, an interpretation that assigns a set to each shape (Figure 6).



$$\cong \int^{M,N} \mathbf{C}(A, M \otimes X \otimes N) \times \mathbf{C}(M \otimes Y \otimes N, B),$$



$$\cong \int^{M,N} \mathbf{C}(I_0, M \otimes N) \times \mathbf{C}(I_1 \otimes M, O_1) \times \mathbf{C}(N \otimes I_2, O_2).$$

FIGURE 6. The shapes of Figure 4, interpreted as sets.

2.1. **Inputs, outputs, junctions and forks.** Shapes will be interpreted in **Prof**, the monoidal bicategory of profunctors. Its 0-cells are small categories $(\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots)$; its 1-cells from $\mathbf{A}$ to $\mathbf{B}$ are profunctors $\mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$; and its 2-cells are natural transformations (see [Lor19, §5]). Two profunctors $P \colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$ and $Q \colon \mathbf{B}^{op} \times \mathbf{C} \to \mathbf{Set}$ compose into a profunctor $(P \diamond Q) \colon \mathbf{A}^{op} \times \mathbf{C} \to \mathbf{Set}$ given by

$$(P \diamond Q)(A, C) \coloneqq \int^{B \in \mathbf{B}} P(A, B) \times Q(B, C).$$

The monoidal product is the cartesian product of categories, two profunctors $P_1 \colon \mathbf{A}_1^{op} \times \mathbf{B}_1 \to \mathbf{Set}$ and $P_2 \colon \mathbf{A}_2^{op} \times \mathbf{B}_2 \to \mathbf{Set}$ can be joint into the profunctor $(P_1 \otimes P_2) \colon (\mathbf{A}_1 \times \mathbf{A}_2)^{op} \times (\mathbf{B}_1 \times \mathbf{B}_2) \to \mathbf{Set}$ defined by

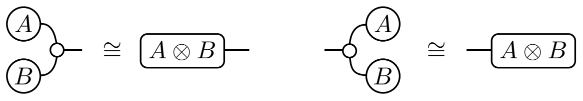
$$(P_1 \otimes P_2)(A_1, A_2, B_1, B_2) \coloneqq P_1(A_1, B_1) \times P_2(A_2, B_2).$$

The string diagrammatic calculus for monoidal bicategories has been studied by Bartlett [Bar14] expanding on a strictification result by Schommer-Pries [SP11]. It is similar to the graphical calculus of monoidal categories, with the caveat that deformations correspond to invertible 2-cells instead of equalities. For instance, arrows between diagrams on this text will denote natural transformations, which are 2-cells of the bicategorical structure of profunctors.

Definition 2.1 (Input and output ports)**.** Every object $A \in \mathbf{C}$ determines two profunctors $(Ⓐ\!-) \coloneqq \mathbf{C}(A, -) \colon \mathbf{1}^{op} \times \mathbf{C} \to \mathbf{Set}$ and $(-\!Ⓐ) \coloneqq \mathbf{C}(-, A) \colon \mathbf{C}^{op} \times \mathbf{1} \to \mathbf{Set}$ via its contravariant and covariant Yoneda embeddings.

Definition 2.2 (Junctions and forks)**.** Every monoidal category $\mathbf{C}$ has a canonical pseudomonoid structure on the monoidal bicategory $\mathbf{Prof}$ given by $(\succ\!\circ\!-) \coloneqq \mathbf{C}(- \otimes -, -)$ and $(\circ\!-) \coloneqq \mathbf{C}(I, -)$, and also a canonical pseudocomonoid structure given by $(-\!\circ\!\prec) \coloneqq \mathbf{C}(-, - \otimes -)$ and $(-\!\circ) \coloneqq \mathbf{C}(-, I)$.

Proposition 2.3. *By definition,* $(Ⓘ\!-) \cong (\circ\!-)$ *and* $(-\!Ⓘ) \cong (-\!\circ)$*; moreover,*



In general, Yoneda embeddings are pseudofunctorial (see Proposition 8.3).

2.2. **Copying and discarding.** Shapes define sets in terms of coends, making them less practical for direct manipulation. However, shapes can be reduced to more familiar descriptions in some particular cases. For instance, if $\mathbf{C}$ is cartesian monoidal, the shape of Figure 7 reduces to a pair of morphisms $\mathbf{C}(I_0 \times I_1, O_1)$ and $\mathbf{C}(I_0 \times I_2, O_2)$. This justifies our previous intuition, back in Figure 4, that the input I_1 should not be able to affect O_2, while the input I_2 should not be able to affect O_1.

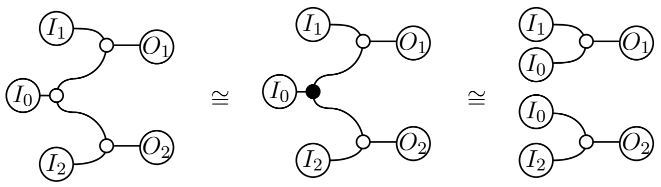


FIGURE 7. Simplifying a diagram.

Our second step is to justify some reductions like these in the cases of cartesian, cocartesian and symmetric monoidal categories. Every object of the category of profunctors has already a canonical pseudocomonoid structure lifted from $\mathbf{Cat}$ which is given by $(-\!\bullet\!\prec) \coloneqq \mathbf{C}(-^0, -^1) \times \mathbf{C}(-^0, -^2)$ and $(-\!\bullet) \coloneqq 1$, and also a pseudomonoid structure given by $(\succ\!\bullet\!-) \coloneqq \mathbf{C}(-^1, -^0) \times \mathbf{C}(-^2, -^0)$, and $(\bullet\!-) \coloneqq 1$. These two structures "copy and discard" representable and corepresentable functors, respectively (see Proposition 8.5).

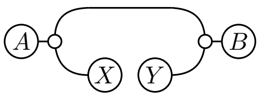
Proposition 2.4 (Cartesian and cocartesian)**.** *A monoidal category is cartesian if and only if* $(-\!\circ) \cong (-\!\bullet)$ *and* $(-\!\circ\!\prec) \cong (-\!\bullet\!\prec)$*, i.e. the monoidal structure coincides with the canonical one. Dually, a monoidal category is cocartesian if and only if* $(\succ\!\bullet\!-) \cong (\succ\!\circ\!-)$ *and* $(\circ\!-) \cong (\bullet\!-)$*.*

Proof. The natural isomorphism $\mathbf{C}(X, Y \otimes Z) \cong \mathbf{C}(X, Y) \times \mathbf{C}(X, Z)$ is precisely the universal property of the product; a similar reasoning holds for initial objects, terminal objects and coproducts. $\square$

Proposition 2.5 (Symmetric monoidal)**.** *If a monoidal category* $\mathbf{C}$ *is* symmetric *then its symmetric pseudomonoid structure can be lifted from* $\mathbf{Cat}$ *to* $\mathbf{Prof}$*. We have* $\sigma\colon ( ) \cong ( )$ *and* $\sigma^*\colon ( ) \cong ( )$*, dual 2-cells in the bicategory* $\mathbf{Prof}$ *that commute with unitors and associators (see also Proposition 8.4).*

2.3. **Example: Lenses.** Profunctor optics and lenses have been extensively studied in functional programming [Kme18, Mil17, PGW17, BG18] for bidirectional data accessing. The theory of optics uses coend calculus both to describe how optics compose and how to reduce them in sufficiently well-behaved cases to tuples of morphisms. Categories of monoidal optics and the informal interpretation of optics as *diagrams with holes* have been studied in depth [Ril18]. We will study lenses from the perspective of the graphical calculus of $\mathbf{Prof}$. This presents a new way of describing reductions with coend calculus that also formalizes the intuition of lenses as *diagrams with holes.*

Definition 2.6. A monoidal lens [Mil17, PGW17, Ril18, "Optic" in Definition 2.0.1] from $A, B \in \mathbf{C}$ to $X, Y \in \mathbf{C}$ is an element of the following set.

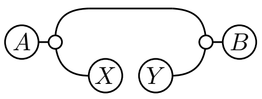


$$= \int^{M} \mathbf{C}(A, M \otimes X) \times \mathbf{C}(M \otimes Y, B)$$

For applications [FJ19, GHWZ18], the most popular case of monoidal lenses is that of cartesian lenses.

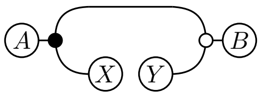
Proposition 2.7. *In a cartesian category* $\mathbf{C}$*, a lens* $(A, B) \to (X, Y)$ *is given by a pair of morphisms* $\mathbf{C}(A, X)$ *and* $\mathbf{C}(A \times Y, B)$*. In a cocartesian category, lenses are called* prisms *[Kme18] and they are given by a pair of morphisms* $\mathbf{C}(S, A + T)$ *and* $\mathbf{C}(B, T)$*.*

Proof. We write the proof for lenses, the proof for prisms is dual and can be obtained by mirroring the diagrams. The coend derivation can be found, for instance, in [Mil17].



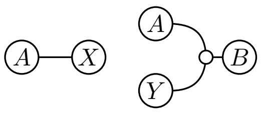
$$\int^{M} \mathbf{C}(A, M \times X) \times \mathbf{C}(M \times Y, B)$$

$\cong$ $\{( ) \cong ( )\}$ $\qquad$ $\cong$ {Universal property of the product}



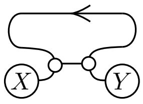
$$\int^{M} \mathbf{C}(A, M) \times \mathbf{C}(A, X) \times \mathbf{C}(M \times Y, B)$$

$\cong$ {Copy} $\qquad$ $\cong$ {Yoneda lemma}



$$\mathbf{C}(A, X) \times \mathbf{C}(A \times Y, B)$$ $\square$

2.4. **Example: Feedback.** Shapes do not need to be limited to a single category. For instance, we can make use of the opposite category to introduce feedback, in the sense of the *categories with feedback* of [KSW02]. Wires in the opposite category will be marked with an arrow to distinguish them.



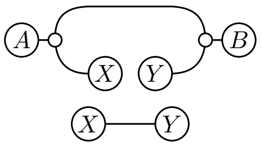
$$= \int^{M \in \mathbf{C}} \mathbf{C}(M \otimes X, M \otimes Y).$$

FIGURE 8. A shape with feedback, interpreted as a set.

Proposition 2.8 (see [Sta13])**.** *Profunctors form a compact closed bicategory. The dual of a category is its opposite category.*

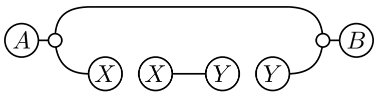
3. COMPOSING AND REDUCING SHAPES

We have been focusing on the invertible transformations between shapes, but arguably the most interesting case is that of non-invertible transformations. Our next step is to describe rules for composing and reducing diagrams that translate to valid coend calculus reductions. For instance, as we saw in the introduction (Figure 2), a lens $(A, B) \to (X, Y)$ can be composed with a morphism $X \to Y$ to obtain a morphism $A \to B$.



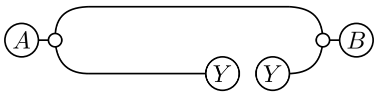
$$\left(\int^{M} \mathbf{C}(A, M \otimes X) \times \mathbf{C}(M \otimes Y, B)\right) \times \mathbf{C}(X, Y)$$

$\cong$ {Isotopy} $\cong$ {Continuity}



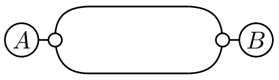
$$\int^{M} \mathbf{C}(A, M \otimes X) \times \mathbf{C}(X, Y) \times \mathbf{C}(M \otimes Y, B)$$

$\to$ $\{\varepsilon_X\}$ $\to$ {Composition along X}



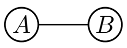
$$\int^{M} \mathbf{C}(A, M \otimes Y) \times \mathbf{C}(M \otimes Y, B)$$

$\to$ $\{\varepsilon_Y\}$ $\to$ {Composition along Y}



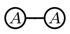
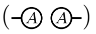
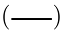
$$\int^{M,N} \mathbf{C}(A, M \otimes N) \times \mathbf{C}(M \otimes N, B)$$

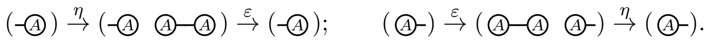
$\to$ $\{\varepsilon_\otimes\}$ $\to$ {Composition along $M \otimes N$}

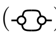
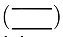
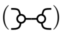


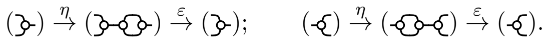
$$\mathbf{C}(A, B)$$

FIGURE 9. Composing a lens with a morphism, formalizing Figure 2.

Definition 3.1 (Joining and splitting wires)**.** Identities and composition define natural transformations $\eta_A \colon (\quad) \to$ () and $\varepsilon_A \colon$ () $\to$ (). They determine an adjunction, as the following transformations are identities.



In the same vein, junctions and forks have natural transformations $\varepsilon_\otimes \colon$ () $\to$ (——) and $\eta_\otimes \colon$ () $\to$ (). They determine an adjunction, as the following transformations are identities.



3.1. **Example: Categories of Optics.** Two lenses of types $(A, B) \to (X, Y)$ and $(X, Y) \to (U, V)$ can be composed with each other to form a category of optics [Ril18]. There is, however, another way of composing two lenses. When the base category is symmetric, a lens $(A, Y) \to (X, V)$ can be composed with a lens $(X, B) \to (U, Y)$ into a lens $(A, B) \to (U, Y)$. We will observe that, even if **Prof** is symmetric, the reduction explicitly uses symmetry on the base category **C**.

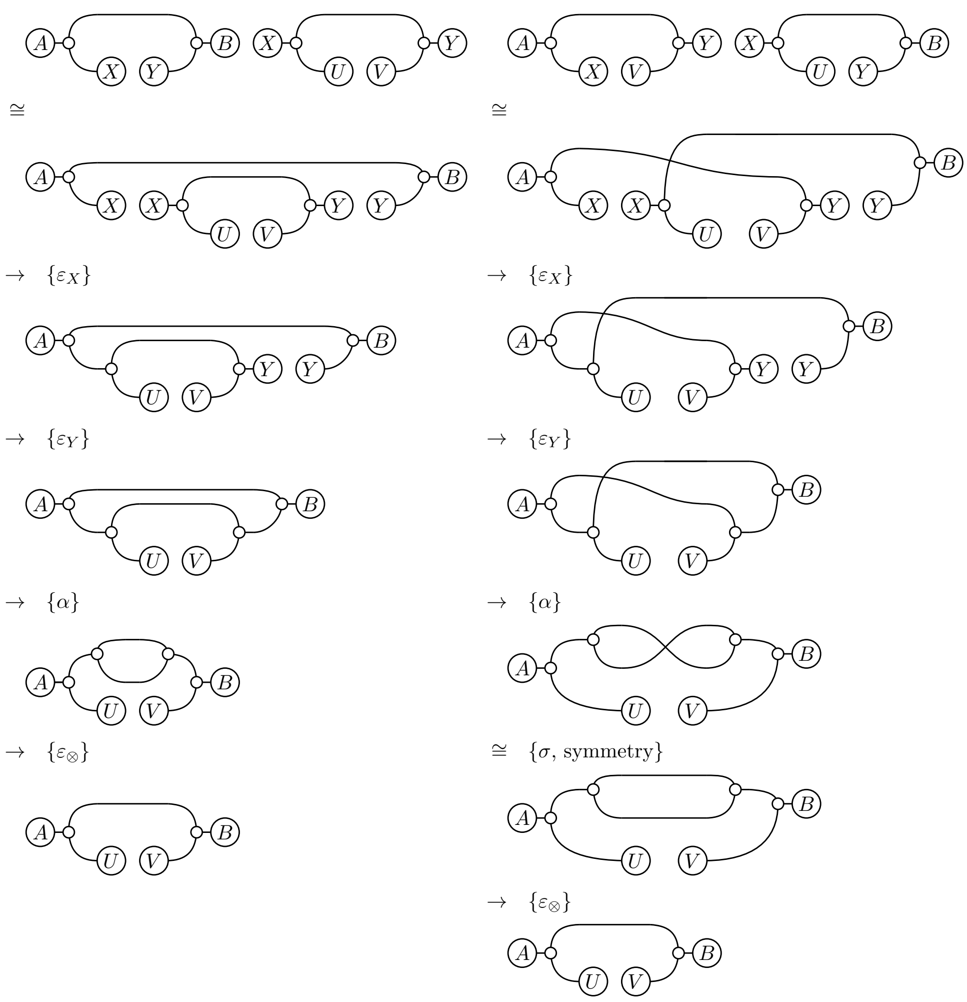
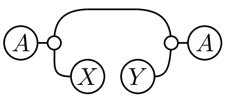


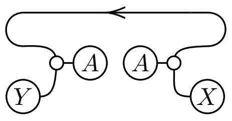
FIGURE 10. In parallel, two possible compositions of optics.

3.2. **Example: from Lenses to Dynamical Systems.** In [SSV16, Definition 2.3.1], a discrete dynamical system, a Moore machine, is characterized to have the same data as a lens $(A, A) \to (X, Y)$. The following derivation is a conceptual justification of this coincidence: a lens with suitable types can be made into a morphism of the free category with feedback [KSW02], subsuming particular cases such as *Moore machines*.



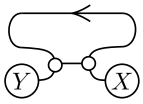
$$\int^{M} \mathbf{C}(A, M \otimes X) \times \mathbf{C}(M \otimes Y, A)$$

$\cong$ {Isotopy} $\qquad \cong$ {Commutativity of $(\times)$}



$$\int^{M} \mathbf{C}(M \otimes Y, A) \times \mathbf{C}(A, M \otimes X)$$

$\to$ $\{\varepsilon_A\}$ $\qquad \to$ {Composition along A}



$$\int^{M} \mathbf{C}(M \otimes Y, M \otimes X)$$

FIGURE 11. From lenses to dynamical systems.

4. Open Diagrams

Our final step is to justify how to obtain the diagrams that originally motivated this text (*open diagrams*) by "looking inside" the shapes. So far, the element of a set described by a shape could be only expressed as a derivation of the shape from the empty diagram. In this section, we show diagrams that summarize these derivations and that represent specific elements of the shape.

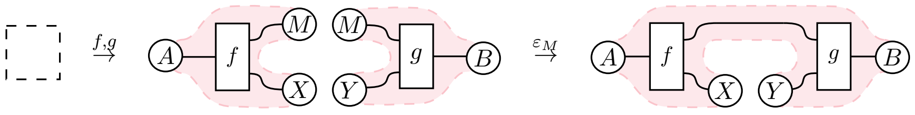


FIGURE 12. Open diagrams represent specific elements.

4.1. **Open Diagrams.** Open diagrams will be interpreted in $\mathbf{Prof}_*$, the symmetric monoidal bicategory of pointed profunctors. Its 0-cells are categories with a chosen object; its 1-cells from $(\mathbf{A}, X)$ to $(\mathbf{B}, Y)$ are profunctors $P \colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$ with a chosen point $p \in P(X, Y)$; and its 2-cells are natural transformations preserving that chosen point.

Proposition 4.1. *Reductions on shapes can be lifted to reductions on open diagrams.*

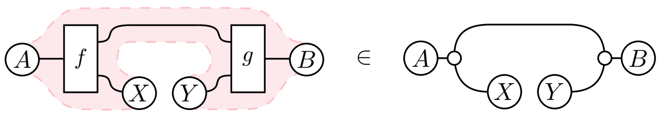
Proof. There exists a pseudofunctor $U \colon \mathbf{Prof}_* \to \mathbf{Prof}$ that forgets about the specific point. It holds that $a \in A$ for every element $(A, a) \in \mathbf{Prof}_*((\mathbf{1}, 1), (\mathbf{1}, 1))$. Natural transformations $\alpha \colon P \to Q$ can be lifted to $\alpha_* \colon (P, p) \to (Q, \alpha(p))$ in a unique way, determining a discrete opfibration $\mathbf{Prof}_*(\mathbf{A}, \mathbf{B}) \to \mathbf{Prof}(\mathbf{A}, \mathbf{B})$ for every pair of categories $\mathbf{A}$ and $\mathbf{B}$. □

Proposition 4.2. *Diagrams on the base category can be lifted to open diagrams.*

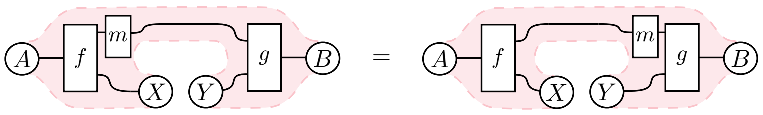
Proof. Let $\mathbf{C}$ be a small category. There exists a pseudofunctor $\mathbf{C} \to \mathbf{Prof}_*$ sending every object $A \in \mathbf{C}$ to the 0-cell pair $(\mathbf{C}, A)$ and every morphism $f \in \mathbf{C}(A, B)$ to the 1-cell pair $(\hom_{\mathbf{C}}, f)$. Moreover, when $(\mathbf{C}, \otimes, I)$ is monoidal, the pseudofunctor is lax and oplax monoidal (weak pseudofunctor in [MV18]), with oplaxators being left adjoint to laxators (see §8.3). This can be called an *op-ajax monoidal pseudofunctor*, following the notion of *ajax monoidal functor* from [FS18]. □

The graphical calculus for open diagrams can then be interpreted as the graphical calculus of pointed profunctors enhanced with a pseudofunctorial box, in the same vein as the functor boxes of [Mel06]. Similar *"internal diagrams"* have been described before by [BDSPV15] (and summarized in [Hu19]) as a "graphical mnemonic notation".

4.2. **Example: Categories of Optics.** The lens $\langle g, f\rangle \colon (A,B) \to (X,Y)$ is depicted as the following open diagram.

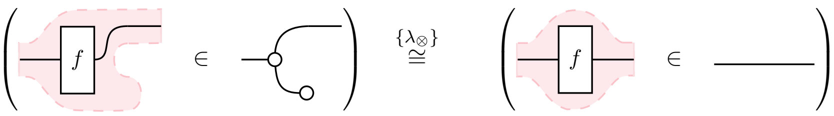


The quotienting that makes $\langle g, (m \otimes \mathrm{id}_X) \circ f\rangle = \langle g \circ (m \otimes \mathrm{id}_Y), f\rangle$ is explicit in this graphical calculus. The following two diagrams are equal in the category $\mathbf{Prof}_*$: they represent the same set and the same element within it.

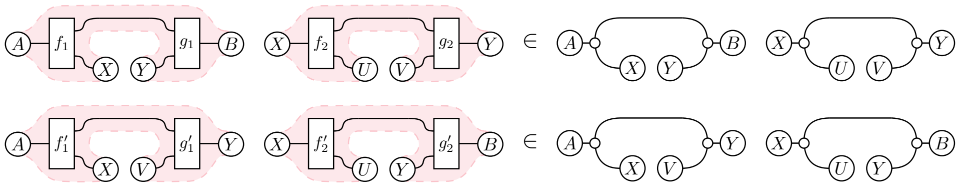


Note that we cannot speak of equality between open diagrams with different shapes, for they belong to different sets. We could however speak of equality between two open diagrams such that the shape of the first can be deformed into the shape of the second. The deformation determines an isomorphism between the sets defined by the shapes. Equality of elements on isomorphic sets is understood to be equality after applying the isomorphism.

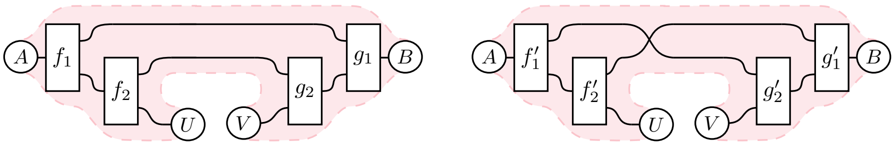
For instance, the following two elements are equal under the deformation given by counitality of the pseudocomonoid structure.



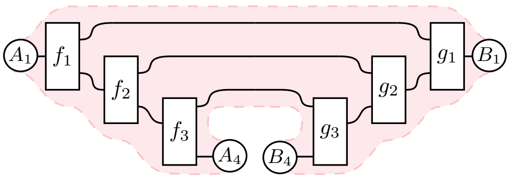
We will use open diagrams to justify that both compositions from Example 3.1 determine a category. Consider two pairs of lenses of suitable types.

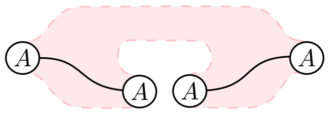


We can use Proposition 4.1 to lift the two compositions in Example 3.1 to two deformations of open diagrams that send the two pairs of lenses to the following two open diagrams, respectively.

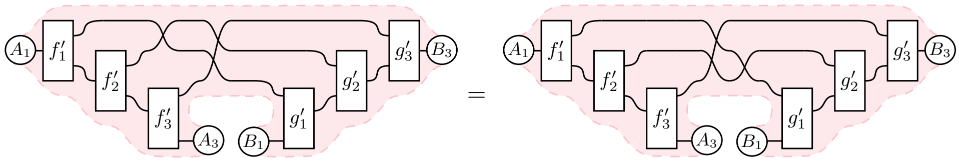


Let us show that a category can be defined from the first composition. Consider three lenses o_i for $i = 1, 2, 3$. We have two ways of composing them, as $o_1 \circ (o_2 \circ o_3)$ or $(o_1 \circ o_2) \circ o_3$, but they both give rise to the same final diagram, thanks to associativity of the base monoidal category. The identity is the diagram on the right.





For the second composition, checking associativity amounts to the following equality. The identity is the same as in the previous case.



The graphical calculus is hiding at the same time the details of two structures. The first is the quotient relation given by the coend in the monoidal bicategory of profunctors; the second is the coherence of the base monoidal category inside the pseudofunctorial box.

5. Related and Further Work

The graphical calculus for profunctors can be seen as a direction in which the graphical calculus for the cartesian bicategory of relations [BPS17, FS18] can be categorified. A notion of *cartesian bicategory* generalizing relations is discussed in [CKWW08]. For a slightly different future direction, we could try to relate this work to many of the interesting applications of *compact closed bicategories* (see [Sta13]); such as *resistor networks*, *double-entry bookkeeping* [KSW08] or *higher linear algebra* [KV94].

Certain shapes open diagrams have been described in the literature. Specifically, finite *combs* were used as notation by [CDP08, KU17, Ril18]; the relation with lenses is described in [Rom20]. Previous graphical calculi for lenses and optics [Hed17, Boi20] have elegantly captured some aspects of optics by working on the Kleisli or Eilenberg-Moore categories of the Pastro-Street monoidal monad [PS08]. The present approach diverges from previous formalisms by using the monoidal bicategory structure of profunctors. It is more general than considering combs, as it can express arbitrary shapes in non-symmetric monoidal categories. In any case, it enables us to reason about categories of optics themselves; the results on optics of [CEG$^+$20] can be greatly simplified in this calculus. We believe that it is closer to, and it provides a formal explanation to the *diagrams with holes* of [Ril18, Definition 2.0.1], which were missing from previous approaches.

Most of our first part can be repeated for arbitrary monoidal bicategories such as enriched profunctors or spans. Multiple approaches to open systems (decorated cospans [Fon15], structured cospans [BC19]) could be related in this way to open diagrams, but we have not explored this possibility yet. Another potential direction is to repeat this reasoning for the case of double categories and obtain a "tile" version of these diagrams (see [Mye16, HS19]).

6. Conclusions

We have presented a way to study and compose *processes* in monoidal categories that do not necessarily have the usual shape of a square box without losing the benefits of the usual language of monoidal categories. Direct applications seem to be circuit design, see

[CDP08], or the theory of optics [CEG+20]. This technique is justified by the formalism of coend calculus [Lor19] and string diagrams for monoidal bicategories [Bar14]. We also argue that the graphical representation of coend calculus is helpful to its understanding: contrasting with usual presentations of coends that are usually centered around the Yoneda reductions, the graphical approach seems to put more weight in the non-reversible transformations while making most applications of Yoneda lemma transparent. Regarding open diagrams, we can think of many other applications that have not been described in this text: we could speak of multiple categories at the same time and combine open diagrams of any of them using functors and adjunctions. This work has opened many paths that we aim to further explore.

We have been working in the symmetric monoidal bicategory of profunctors for simplicity, but the same results extend to the symmetric monoidal bicategory of $\mathcal{V}$-profunctors for $\mathcal{V}$ a Bénabou cosmos [Lor19, §5]. We can even consider arbitrary monoidal bicategories and drop the requirements for symmetry, copying or discarding. Finally, there is an important shortcoming to this approach that we leave as further work: the present graphical calculus is an extremely good tool for *coend calculus*, but it remains to see if it is so for *(co)end calculus*. In other words, *ends* "enter the picture" only as natural transformations (see [Wil10]), and this can feel limiting even if, after applying Yoneda embeddings, it usually suffices for most applications. As it happens with diagrammatic presentations of regular logic [BPS17, FS18], the existential quantifier plays a more prominent role. Diagrammatic approaches to obtaining the universal quantifier in a situation like this go back to Peirce and are described by [HS20].

7. Acknowledgements

The author thanks seminars, discussion with, questions and/or comments by Paweł Sobociński, Edward Morehouse, Fosco Loregian, Elena Di Lavore, Jens Seeber, Jules Hedges, Nathaniel Virgo, and the whole Compositional Systems and Methods group at Tallinn University of Technology. Mario Román was supported by the European Union through the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

References

[Bar14] Bruce Bartlett. Quasistrict Symmetric Monoidal 2-Categories Via Wire Diagrams, 2014.

[BC19] John C. Baez and Kenny Courser. Structured cospans, 2019.

[BDSPV15] Bruce Bartlett, Christopher L Douglas, Christopher J Schommer-Pries, and Jamie Vicary. Modular Categories as Representations of the 3-Dimensional Bordism 2-category. *arXiv preprint arXiv:1509.06811*, 2015.

[BG18] Guillaume Boisseau and Jeremy Gibbons. What You Needa Know About Yoneda: Profunctor Optics and the Yoneda Lemma (Functional Pearl). *PACMPL*, 2(ICFP):84:1–84:27, 2018.

[Boi20] Guillaume Boisseau. String Diagrams for Optics. *arXiv preprint arXiv:2002.11480*, 2020.

[Bor94] Francis Borceux. *Handbook of categorical algebra: volume 1, Basic category theory*, volume 1. Cambridge University Press, 1994.

[BPS17] Filippo Bonchi, Dusko Pavlovic, and Paweł Sobociński. Functorial semantics for relational theories. *CoRR*, abs/1711.08699, 2017.

[CDP08] G. Chiribella, G. M. D'Ariano, and P. Perinotti. Quantum Circuits Architecture. *Physical Review Letters*, 101(6), Aug 2008.

[CEG+20] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, and Mario Román. Profunctor optics, a categorical update. *arXiv preprint arXiv:1501.02503*, 2020.

[CK17] Bob Coecke and Aleks Kissinger. *Picturing quantum processes.* Cambridge University Press, 2017.

[CKWW08] Aurelio Carboni, G Max Kelly, Robert FC Walters, and Richard J Wood. Cartesian bicategories ii. *Theory and Applications of Categories*, 19(6):93–124, 2008.

[FJ19] Brendan Fong and Michael Johnson. Lenses and Learners. *CoRR*, abs/1903.03671, 2019.

[Fon15] Brendan Fong. Decorated Cospans, 2015.

[FS18] Brendan Fong and David I. Spivak. Graphical regular logic. *CoRR*, abs/1812.05765, 2018.

[FST19] Brendan Fong, David Spivak, and Rémy Tuyéras. Backprop as Functor: A compositional perspective on supervised learning. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE, 2019.

[GHWZ18] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481, 2018.

[Hed17] Jules Hedges. Coherence for lenses and open games. *CoRR*, abs/1704.02230, 2017.

[HS19] Linde Wester Hansen and Michael Shulman. Constructing symmetric monoidal bicategories functorially. *arXiv preprint arXiv:1910.09240*, 2019.

[HS20] Nathan Haydon and Paweł Sobociński. Compositional diagrammatic first-order logic. *In Peer Review*, 2020.

[Hu19] Nick Hu. External traced monoidal categories. Master's thesis, University of Oxford, 2019.

[Kme18] Edward Kmett. lens library, version 4.16. Hackage https://hackage.haskell.org/package/lens-4.16, 2012–2018.

[KSW02] Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *ITA*, 36(2):181–194, 2002.

[KSW08] Piergiulio Katis, N. Sabadini, and R. F. C. Walters. On partita doppia, 2008.

[KU17] Aleks Kissinger and Sander Uijlen. A categorical semantics for causal structure. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE, 2017.

[KV94] M. M. Kapranov and V. A. Voevodsky. *2-categories and Zamolodchikov tetrahedra equations*, volume 56 of *Proc. Sympos. Pure Math.*, page 177–259. Amer. Math. Soc., Providence, RI, 1994.

[Lor19] Fosco Loregian. Coend calculus. *arXiv preprint arXiv:1501.02503*, 2019.

[Mel06] Paul-André Melliès. Functorial boxes in string diagrams. In *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, pages 1–30, 2006.

[Mil17] Bartosz Milewski. Profunctor optics: the categorical view. https://bartoszmilewski.com/2017/07/07/profunctor-optics-the-categorical-view/, 2017.

[MV18] Joe Moeller and Christina Vasilakopoulou. Monoidal grothendieck construction. *arXiv preprint arXiv:1809.00727*, 2018.

[Mye16] David Jaz Myers. String diagrams for double categories and equipments. *arXiv preprint 1612.02762*, 2016.

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor Optics: Modular Data Accessors. *Programming Journal*, 1(2):7, 2017.

[PS08] Craig Pastro and Ross Street. Doubles for monoidal categories. *Theory and applications of categories*, 21(4):61–75, 2008.

[Ril18] Mitchell Riley. Categories of Optics. *arXiv preprint arXiv:1809.00738*, 2018.

[Rom20] Mario Román. Comb Diagrams for Discrete-Time Feedback. *arXiv preprint arXiv:2003.06214*, 2020.

[SP11] Christopher J. Schommer-Pries. The classification of two-dimensional extended topological field theories, 2011.

[SSV16] Patrick Schultz, David I. Spivak, and Christina Vasilakopoulou. Dynamical Systems and Sheaves, 2016.

[Sta13] Michael Stay. Compact closed bicategories. *arXiv preprint arXiv:1301.1053*, 2013.

[Wil10] Simon Willerton. Two 2-Traces. Slides from a talk, http://www.simonwillerton.staff.shef.ac.uk/ftp/TwoTracesBeamerTalk.pdf, 2010.

8. Appendix

8.1. **The Monoidal Bicategory of Profunctors.**

Definition 8.1. There exists a symmetric monoidal bicategory **Prof** having as 0-cells the (small) categories $\mathbf{A}, \mathbf{B}, \mathbf{C}, \dots$; as 1-cells from $\mathbf{A}$ to $\mathbf{B}$, the profunctors $\mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$; as 2-cells, the natural transformations; and as tensor product, the cartesian product of categories [Lor19]. Two profunctors $P \colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$ and $Q \colon \mathbf{B}^{op} \times \mathbf{C} \to \mathbf{Set}$ compose into the profunctor $(P \diamond Q) \colon \mathbf{A}^{op} \times \mathbf{C} \to \mathbf{Set}$ defined by

$$(P \diamond Q)(A, C) \coloneqq \int^{B \in \mathbf{B}} P(A, B) \times Q(B, C).$$

The unit of composition in the category $\mathbf{A}$ is the hom-profunctor $\hom_{\mathbf{A}} \colon \mathbf{A}^{op} \times \mathbf{A} \to \mathbf{Set}$. Strong unitality, $(\text{——}\boxed{P}\text{-}) \cong (\text{-}\boxed{P}\text{-}) \cong (\text{-}\boxed{P}\text{——})$, is given by the Yoneda isomorphisms.

$$(\hom_{\mathbf{A}} \diamond P)(A, B) \coloneqq \int^{A' \in \mathbf{A}} \hom_{\mathbf{A}}(A, A') \times P(A', B) \cong P(A, B)$$
$$(P \diamond \hom_{\mathbf{B}})(A, B) \coloneqq \int^{B' \in \mathbf{B}} P(A, B') \times \hom_{\mathbf{B}}(B', B) \cong P(A, B).$$

Strong associativity $(\text{-}\boxed{P_1}\text{-}\boxed{P_2}\text{-}) \diamond (\text{-}\boxed{P_3}\text{-}) \cong (\text{-}\boxed{P_1}\text{-}) \diamond (\text{-}\boxed{P_2}\text{-}\boxed{P_3}\text{-})$ follows from continuity and associativity of the cartesian product of sets. The invertible 2-cells that realise unitality and associativity satisfy the pentagon and triangular equations.

The monoidal product of two profunctors $P_1 \colon \mathbf{A}_1^{op} \times \mathbf{B}_1 \to \mathbf{Set}$ and $P_2 \colon \mathbf{A}_2^{op} \times \mathbf{B}_2 \to \mathbf{Set}$ is the profunctor $(P_1 \otimes P_2) \colon (\mathbf{A}_1 \times \mathbf{A}_2)^{op} \times (\mathbf{B}_1 \times \mathbf{B}_2) \to \mathbf{Set}$ defined by

$$(P_1 \otimes P_2)(A_1, A_2, B_1, B_2) \coloneqq P_1(A_1, B_1) \times P_2(A_2, B_2).$$

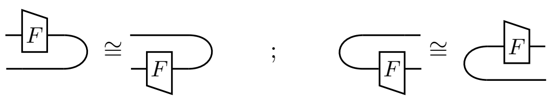
The unit of the monoidal structure is the terminal category. Unitality and associativity follow from those on sets. In the case of profunctors, unitors and associator are not only equivalences but isomorphisms of categories, with strictly commuting pentagons and triangles.

An alternative approach is to construct this symmetric monoidal bicategory from the double symmetric bicategory of profunctors, see [HS19].

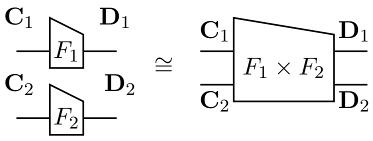
Every category has a dual, its opposite category. There are profunctors $(\mathbf{A}^{op} \times \mathbf{A}) \times \mathbf{1} \to \mathbf{Set}$ and $\mathbf{1}^{op} \times (\mathbf{A}^{op} \times \mathbf{A})^{op} \to \mathbf{Set}$ given by variations of the hom-profunctors; these are represented by caps and cups. Profunctors circulate through the caps and cups as expected thanks to the Yoneda lemma. See [Sta13] for the description as a compact closed bicategory.

Definition 8.2 (Yoneda Embedding of Functors)**.** Let $F \colon \mathbf{C} \to \mathbf{D}$ be a functor. It can be embedded as a profunctor $(\text{-}\boxed{F}\text{-}) \colon \mathbf{C}^{op} \times \mathbf{D} \to \mathbf{Set}$ or as a profunctor $(\text{-}\boxed{F}\text{-}) \colon \mathbf{D}^{op} \times \mathbf{C} \to \mathbf{Set}$. Moreover, every functor has an opposite, so it can also be embedded as a profunctor $(\text{-}\boxed{F}\text{-}) \colon (\mathbf{D}^{op})^{op} \times \mathbf{C}^{op} \to \mathbf{Set}$ or as a profunctor $(\text{-}\boxed{F}\text{-}) \colon (\mathbf{C}^{op})^{op} \times \mathbf{D}^{op} \to \mathbf{Set}$. In particular, $F \dashv G$ precisely when $(\text{-}\boxed{F}\text{-}) \cong (\text{-}\boxed{G}\text{-})$.

The suggestive shape of the boxes (from [CK17]) is matched by their semantics. Every category has a dual (namely, its opposite category) and functors circulate as expected through the cups and the caps that represent dualities.



Proposition 8.3. *Both Yoneda embeddings are strong monoidal pseudofunctors* **Cat** $\to$ **Prof**, *fully faithful on the 2-cells. Pseudofunctoriality gives* $(\text{-}\boxed{F}\text{-}\boxed{G}\text{-}) \cong (\text{-}\boxed{G \circ F}\text{-})$ *and its counterpart. Monoidality gives the following isomorphism and its mirrored counterpart.*



Proposition 8.4 (Functors are Left Adjoints). *In the category of profunctors, functors are left adjoints, in the sense that there exist morphisms* $\eta_F\colon (\text{——}) \to (\text{-}\boxed{F}\text{-}\boxed{F}\text{-})$ *and* $\varepsilon_F\colon (\text{-}\boxed{F}\text{-}\boxed{F}\text{-}) \to (\text{——})$ *and they verify the zig-zag identities. Moreover, every natural transformation commutes with these dualities in the sense that the following are two commutative squares.*[2]

$$\begin{tikzcd}
(\text{——}) \arrow[r, "\eta_F"] \arrow[d, "\eta_G"'] & (\text{-}\boxed{F}\text{-}\boxed{F}\text{-}) \arrow[d, "\alpha"] \\
(\text{-}\boxed{G}\text{-}\boxed{G}\text{-}) \arrow[r, "\alpha"] & (\text{-}\boxed{F}\text{-}\boxed{G}\text{-})
\end{tikzcd}
\qquad
\begin{tikzcd}
(\text{-}\boxed{F}\text{-}\boxed{G}\text{-}) \arrow[r, "\alpha"] \arrow[d, "\alpha"] & (\text{-}\boxed{F}\text{-}\boxed{F}\text{-}) \arrow[d, "\varepsilon_F"] \\
(\text{-}\boxed{G}\text{-}\boxed{G}\text{-}) \arrow[r, "\varepsilon_G"] & (\text{——})
\end{tikzcd}$$

A partial converse holds: a left adjoint profunctor is representable when its codomain is Cauchy complete; see [Bor94].

Proposition 8.5. *Every object* **A** *of the category of profunctors has already a canonical pseudocomonoid structure lifted from* **Cat** *and given by* $(\text{-}\!\prec) := \mathbf{A}(-^0, -^1) \times \mathbf{A}(-^0, -^2)$ *and* $(\text{—}\bullet) := 1$*; but also a pseudomonoid structure given by* $(\succ\!\text{-}) := \mathbf{A}(-^1, -^0) \times \mathbf{A}(-^2, -^0)$*, and* $(\bullet\text{—}) := 1$*. These structures* copy *and* discard *representable and corepresentable functors, respectively; but they also laxly copy and discard arbitrary profunctors.*

Proof. This is a consequence of the fact the diagonal and discard functors $(\Delta)\colon \mathbf{A} \to \mathbf{A} \times \mathbf{A}$ and $(!)\colon \mathbf{A} \to \mathbf{1}$ copy and discard functors in **Cat**. Pseudofunctoriality of both Yoneda embeddings sends them to the profunctors we are describing in **Prof**.

On the other hand, arbitrary profunctors are laxly copied and discarded. For instance, the following coend derivation shows that a profunctor $P\colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$ is laxly copied. In the case of representable profunctors, this is an isomorphism.

$$\begin{aligned}
&\int^X P(A, X) \times \hom_{\mathbf{A}}(X, Y_1) \times \hom_{\mathbf{B}}(X, Y_2) \\
\to\ & \\
&P(A, Y_1) \times P(A, Y_2) \\
\cong\ & \\
&\int^{X_1, X_2} \hom_{\mathbf{A}}(A, X_1) \times \hom_{\mathbf{A}}(A, X_2) \times P(X_1, Y_1) \times P(X_2, Y_2). \qquad \square
\end{aligned}$$

8.2. The Monoidal Bicategory of Pointed Profunctors.

Definition 8.6. A *pointed category* $(\mathbf{A}, X)$ is a category **A** equipped with a chosen object X, which can be regarded as a functor from the terminal category. There exists a symmetric monoidal bicategory $\mathbf{Prof}_*$ having as 0-cells pairs $(\mathbf{A}, X)$ where **A** is a (small) category and $X \in \mathbf{A}$ is an object of that category; 1-cells from $(\mathbf{A}, X) \to (\mathbf{B}, Y)$ pairs (P, p) given by a profunctor $P\colon \mathbf{A}^{op} \times \mathbf{B} \to \mathbf{Set}$ and a point $p \in P(X, Y)$; 2-cells from $(P, p) \to (Q, q)$

[2]The graphical calculus of the bicategory makes these equations much clearer. We are emphasizing the monoidal bicategory structure here only for the sake of coherence.

are natural transformations $\eta\colon P \to Q$ such that $\eta_{X,Y}(p) = q$. Composition of 1-cells $(P,p)\colon (\mathbf{A},X) \to (\mathbf{B},Y)$ and $(Q,q)\colon (\mathbf{B},Y) \to (\mathbf{C},Z)$ is given by $(Q \diamond P, \langle q,p\rangle)$, where $\langle q,p\rangle \in (Q \diamond P)(X,Z)$ is the equivalence class under the coend of the pair (q,p). The identity 1-cell in $(\mathbf{A},X)$ is $(\hom_{\mathbf{A}}, \mathrm{id}_X)\colon (\mathbf{1},1) \to (\mathbf{A},X)$.

Strong unitality $\ell_{(P,p)}\colon (\hom_{\mathbf{A}} \diamond P, \langle \mathrm{id}_X, p\rangle) \to (P,p)$ and $\varrho_{(P,p)}\colon (P \diamond \hom_{\mathbf{A}}, \langle p, \mathrm{id}_X\rangle) \cong (P,p)$ is given by the Yoneda isomorphisms.

$$\ell_P\colon \int^{Z\in\mathbf{A}} \hom_{\mathbf{A}}(X,Z) \times P(Z,Y) \cong P(X,Y)$$
$$\varrho_P\colon \int^{Z\in\mathbf{A}} P(X,Z) \times \hom_{\mathbf{A}}(Z,Y) \cong P(X,Y)$$

The Yoneda isomorphisms are such that $\ell_P \langle \mathrm{id}_X, p\rangle = \mathrm{id}_X \circ p = p$ and $\varrho_P \langle p, \mathrm{id}_Y\rangle = p \circ \mathrm{id}_Y = p$. This confirms they are valid 2-cells of $\mathbf{Prof}_*$.

Strong associativity $a_{(P,p,Q,q,R,r)}\colon ((P \diamond Q) \diamond R, \langle\langle p,q\rangle, r\rangle) \to (P \diamond (Q \diamond R), \langle p, \langle q,r\rangle\rangle)$ is given by the isomorphism described by continuity and associativity of the cartesian product.

$$\int^V \left(\int^U P(X,U) \times Q(U,V)\right) \times R(V,Y) \cong \int^U P(X,U) \times \left(\int^V Q(U,V) \times R(V,Y)\right).$$

It is defined by $a(\langle\langle p,q\rangle, r\rangle) = \langle p, \langle q,r\rangle\rangle$, proving that it is a valid 2-cell of $\mathbf{Prof}_*$.

We will show now it is also symmetric monoidal. There is a distinguished object $(\mathbf{1},1)$, given by the terminal category and its only object. There is a pseudofunctor $(\otimes)\colon \mathbf{Prof}_* \times \mathbf{Prof}_* \to \mathbf{Prof}_*$ defined

- on 0-cells by $(\mathbf{A},X) \otimes (\mathbf{B},Y) \coloneqq (\mathbf{A} \times \mathbf{B}, (X,Y))$;
- on 1-cells by $(P,p) \otimes (Q,q) \coloneqq (P \otimes Q, (p,q))$;
- on 2-cells by $(\gamma \otimes \delta)\colon P \diamond Q \to P' \diamond Q'$ the transformation that applies $\gamma\colon P \to P'$ and $\delta\colon Q \to Q'$ to both factors of the coend,
$$\left(\int^Z P(X,Z) \times Q(Z,Y)\right) \to \left(\int^Z P'(X,Z) \times Q'(Z,Y)\right);$$
- natural isomorphisms $((P \diamond Q) \otimes (P' \diamond Q'), (\langle p,q\rangle, \langle p',q'\rangle)) \cong ((P \otimes P') \diamond (Q \otimes Q'), \langle (p,p'), (q,q')\rangle)$ and $(\hom_{\mathbf{A}\times\mathbf{B}}, \mathrm{id}_{A\times B}) \cong (\hom_{\mathbf{A}} \times \hom_{\mathbf{B}}, (\mathrm{id}_A, \mathrm{id}_B))$.

The symmetric monoidal structure requires the following natural transformations.

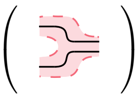
- the left unitor $\lambda^{\otimes}_P\colon (\mathrm{id}_{\mathbf{1}} \otimes P, (\mathrm{id}_*, p)) \to (P,p)$ and right unitor $\rho^{\otimes}_P\colon (P \otimes \mathrm{id}_{\mathbf{1}}, (p, \mathrm{id}_*)) \to (P,p)$;
- the associator $\alpha^{\otimes}_{P,Q,R}\colon ((P \otimes Q) \otimes R, ((p,q),r)) \to (P \otimes (Q \otimes R), (p,(q,r)))$, given by the associator of the cartesian category of sets;
- the braiding $\beta\colon (P \otimes Q, (p,q)) \to (Q \otimes P, (q,p))$;
- such that the two relevant squares and the relevant hexagon commute, making it a syllepsis and a symmetry.

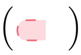
8.3. **Pseudofunctor box.**

Proposition 8.7. *Let* $\mathbf{A}$ *be a small category. There exists a pseudofunctor* $\mathbf{A} \to \mathbf{Prof}_*$ *sending every object* $A \in \mathbf{A}$ *to the 0-cell pair* $(\mathbf{A}, A)$ *and every morphism* $f \in \hom_{\mathbf{A}}(A,B)$ *to the 1-cell pair* $(\hom_{\mathbf{A}}, f)$. *Moreover, when* $(\mathbf{A}, \otimes, I)$ *is monoidal, the pseudofunctor is lax and oplax monoidal (weak pseudofunctor in* [MV18]*, with oplaxators being left adjoint to laxators). This would be an op-ajax monoidal pseudofunctor, following the notion of* ajax monoidal functor *from* [FS18].

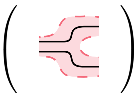
We only sketch the construction. The invertible 2-cells witnessing pseudofunctoriality use the fact that the Yoneda isomorphisms (unitors and associators) send pairs of points to their composition, coinciding with the composition on the base category $\mathbf{A}$.

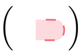
The following natural transformations make the functor lax monoidal.


$$:= (\mathrm{hom}(-,-\otimes -), \mathrm{id}_{A\otimes B})\colon (\mathbf{A}\times\mathbf{A},(A,B)) \to (\mathbf{A}, A\otimes B)$$

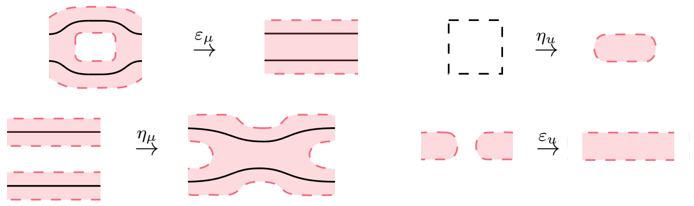

$$:= (\mathrm{hom}(I,-), \mathrm{id}_{I})\colon (\mathbf{1},*) \to (\mathbf{A}, I)$$

The following natural transformations make the functor oplax monoidal.

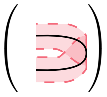

$$:= (\mathrm{hom}(-\otimes -,-), \mathrm{id}_{A\otimes B})\colon (\mathbf{A}, A\otimes B) \to (\mathbf{A}\times\mathbf{A},(A,B))$$

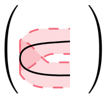

$$:= (\mathrm{hom}(-,I), \mathrm{id}_{I})\colon (\mathbf{A}, I) \to (\mathbf{1},*)$$

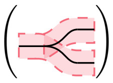
Composition and identities give the counits and units of the adjunctions. The fact that identity is the unit for composition makes the following transformations be 2-cells of $\mathbf{Prof}_*$.

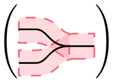

$\overset{\varepsilon_\mu}{\to}$
$\overset{\eta_u}{\to}$
$\overset{\eta_\mu}{\to}$
$\overset{\varepsilon_u}{\to}$

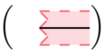
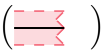
The following morphisms follow the cups, caps, splitting and merging structure from $\mathbf{Prof}$ in $\mathbf{Prof}_*$. Morphisms circulate through them as expected: turning to morphisms in the opposite category, being copied and discarded.


$$:= (\mathrm{hom}(-,-), \mathrm{id}_{A})\colon (\mathbf{A}\times\mathbf{A}^{op},(A,A)) \to (\mathbf{1},1),$$


$$:= (\mathrm{hom}(-,-), \mathrm{id}_{A})\colon (\mathbf{1},1) \to (\mathbf{A}\times\mathbf{A}^{op},(A,A)),$$


$$:= (\mathrm{hom}(-^0,-^1)\times\mathrm{hom}(-^0,-^2), (\mathrm{id}_{A},\mathrm{id}_{A}))\colon (\mathbf{A}, A) \to (\mathbf{A}\times\mathbf{A},(A,A)),$$

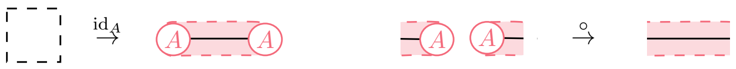

$$:= (\mathrm{hom}(-^1,-^0)\times\mathrm{hom}(-^2,-^0), (\mathrm{id}_{A},\mathrm{id}_{A}))\colon (\mathbf{A}\times\mathbf{A},(A,A)) \to (\mathbf{A}, A),$$


$$:= (1,*)\colon (\mathbf{1},1) \to (\mathbf{A},A);$$

$$:= (1,*)\colon (\mathbf{A},A) \to (\mathbf{1},1).$$

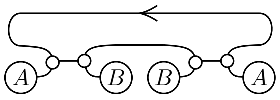
Proposition 8.8. *Let* $\mathbf{A}$ *be a category. For every* $A\in\mathbf{A}$*, there exist 1-cells*

$$(hom_{\mathbf{A}}(A,-), \mathrm{id}_A)\colon (\mathbf{1},1) \to (\mathbf{A},A) \quad \textit{and} \quad (\mathrm{hom}_{\mathbf{A}}(-,A), \mathrm{id}_A)\colon (\mathbf{A},A) \to (\mathbf{1},1)$$

given by the Yoneda embeddings of A *and the identity morphism. Composition and identities define an adjunction.*

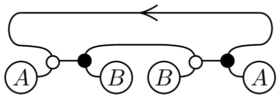


8.4. **Example: Learners.** A *learner* [FST19, Definition 4.1] in a cartesian category is given by a *parameters* object $P \in \mathbf{C}$, an *implementation* morphism $i \colon P \times A \to B$, and *update* morphism $u \colon P \times A \times B \to P$, and a *request* morphism $r \colon P \times A \times B \to A$. A monoidal generalization, dinatural on the parameters object, has been proposed in [Ril18, Definition 6.4.1]; the following derivation shows how it particularizes into the cartesian case.



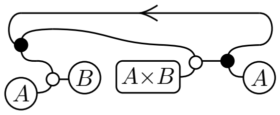
$$\int^{P,Q} \mathbf{C}(P \times A, Q \times B) \times \mathbf{C}(Q \times B, P \times A)$$

$\cong$ $\{(\text{-}\circ) \cong (\text{-}\bullet)\}$ $\qquad\cong$ {Universal property of the product}



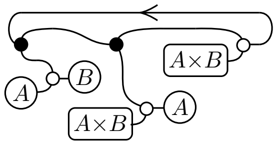
$$\int^{P,Q} \mathbf{C}(P \times A, Q) \times \mathbf{C}(P \times A, B) \times \mathbf{C}(Q \times B, P \times A)$$

$\cong$ $\{(\text{-}\bullet)$ copies$\}$ $\qquad\cong$ {Yoneda lemma}



$$\int^{P} \mathbf{C}(P \times A, B) \times \mathbf{C}(P \times A \times B, P \times A)$$

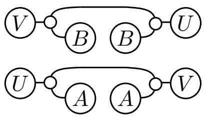
$\cong$ $\{(\text{-}\bullet)$ copies$\}$ $\qquad\cong$ {Universal property of the product}



$$\int^{P} \mathbf{C}(P \times A, B) \times \mathbf{C}(P \times A \times B, A) \times \mathbf{C}(P \times A \times B, P)$$

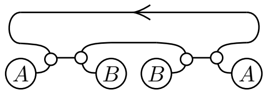
FIGURE 13. From monoidal to cartesian learners.

Proposition 8.9. *A pair of lenses $(U, V) \to (A, A)$ and $(V, U) \to (B, B)$ define a learner.*



$$\int^{M,N} \mathbf{C}(U, M \otimes A) \times \mathbf{C}(M \otimes A, V) \times \mathbf{C}(V, N \otimes B) \times \mathbf{C}(N \otimes B, U)$$

$\to$ $\{\varepsilon_U, \varepsilon_V\}$ $\qquad\to$ {Composition along U and V}



$$\int^{P} \mathbf{C}(P \times A, B) \times \mathbf{C}(P \times A \times B, A) \times \mathbf{C}(P \times A \times B, P)$$

FIGURE 14. From lenses to learners.

6. Collages of String Diagrams

Dylan Braithwaite, Mario Román
Applied Category Theory (ACT, 2022)

Abstract: We introduce collages of string diagrams as a diagrammatic syntax for glueing multiple monoidal categories. Collages of string diagrams are interpreted as pointed bimodular profunctors. As the main examples of this technique, we introduce string diagrams for bimodular categories, string diagrams for functor boxes, and string diagrams for internal diagrams.

Declaration: *Hereby I declare that my contribution to this manuscript was to: write most of the manuscript, identify and prove most of the results. Both the author and Dylan Braithwaite identified the research question independently.*

Collages of String Diagrams

Dylan Braithwaite
University of Strathclyde
dylan.braithwaite@strath.ac.uk

Mario Román
Tallinn University of Technology
mroman@ttu.ee

We introduce collages of string diagrams as a diagrammatic syntax for glueing multiple monoidal categories. Collages of string diagrams are interpreted as pointed bimodular profunctors. As the main examples of this technique, we introduce string diagrams for bimodular categories, string diagrams for functor boxes, and string diagrams for internal diagrams.

1 Introduction

String diagrams are a convenient and intuitive, sound and complete syntax for monoidal categories [29]. Monoidal categories are algebras of processes composing in parallel and sequentially [34]; string diagrams formalize the process diagrams of engineering [6, 8]. Formalization is not only of conceptual interest: it means we can sharpen our reasoning, scale our diagrams, or explain them to a computer [42].

However, the formal syntax of monoidal categories is not enough for all applications and, sometimes, we need to extend it. Functor boxes allow us to reason about translations between theories of processes [15, 37], ownership [39], higher-order processes [1], or programming effects [43]. Quantum combs not only model some classes of supermaps [12, 16, 23], but they coincide with the monoidal lenses of functional programming [5, 13, 50] and compositional game theory [22, 7]. Premonoidal categories, which appear in Moggi's semantics of programming effects [38, 30, 51], are now within the realm of string diagrammatic reasoning [46]. Internal diagrams extend the syntax of monoidal categories allowing us to draw diagrams inside tubular cobordisms and reason about topological quantum field theories [3], but also coends [47] and traces [26].

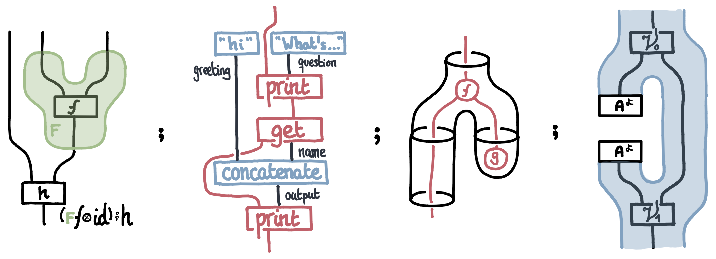


Figure 1: Examples from the literature. From left to right: functor boxes [37], premonoidal categories [46], internal diagrams [3], and combs or optics [12, 13, 23].

The extensions showcase the expressive power of string diagrams on surprisingly diverse application domains. At the same time, these different ideas could be regarded as separate ad-hoc extensions: they belong to different fields; they use different categorical formalisms. The overhead of learning and combining each one of them prevents the exchange of ideas between the different domains of application: e.g. an idea about topological quantum field diagrams does not transfer to premonoidal diagrams.

Submitted to:
ACT 2023

Collages. This manuscript claims that this division is only apparent and that all these extensions are particular instances of the same encompassing idea: that of glueing multiple string diagrams into what we call a *collage of string diagrams*. We introduce a formal notion of collage (Section 4.4) and employ string diagrammatic syntaxes for them, based on the calculus of bicategories (Sections 2.1, 3.1 and 5).

Even when collages of string diagrams are our novel contribution, collages are not yet another new concept to category theory. "Collage" was Bob Walters' term for a lax colimit in a module-like category [52]. This can be considered as a glueing of objects together along the action of a scalar. For example, given two sets A and B, with an action of a monoid M, we can construct their tensor product $A \otimes_M B$, where $(a \cdot m) \otimes b = a \otimes (m \cdot b)$ for any scalar $m \in M$. Categorifying this idea in a possible direction we obtain monoidal categories acting on *bimodular categories*. The following is the takeaway of this work.

> Collages of string diagrams consist of multiple string diagrams of different monoidal categories glued together. Collages can be interpreted as *pointed bimodular profunctors* between *bimodular categories*.

A bimodular category, sometimes referred to as a biactegory [10], is to a bimodule what a monoidal category is to a monoid. This is, a plain category $\mathbb{A}$ endowed with a left action of a monoidal category $(\triangleright)\colon \mathbb{M} \times \mathbb{A} \to \mathbb{A}$ and a right action of another, possibly different, monoidal category $(\triangleleft)\colon \mathbb{A} \times \mathbb{N} \to \mathbb{A}$. We can *collage* two bimodular categories along a common monoidal category that acts on both. Later on the paper, exploiting a second axis of categorification, we pass from bimodular categories to *bimodular profunctors*, which are a kind of 2-dimensional bimodule, and we define their collage. This structure facilitates glueing categories together in 2-dimensions: we can represent complexes of morphisms from different categories and glue them together. Collages of string diagrams are the syntactic representations of this glueing, in the same sense that ordinary string diagrams represent tensors in monoidal categories.

We observe that collages of bimodular categories embed into a tricategory of pointed bimodules. This provides a versatile setting where we can interpret many syntaxes already present in the literature.

Contributions. We introduce string diagrams of bimodular categories and we prove they construct the free bimodular category on a signature (Theorem 2.7). We introduce novel string diagrammatic syntax for *functor boxes* and we prove it constructs the free lax monoidal functor on a suitable signature (Theorem 3.4). We describe the tricategory of pointed bimodular profunctors (Definition 4.6) and, in terms of it, we explain the semantics of functor boxes (Proposition 4.9) and internal diagrams (Theorem 5.3), for which we also provide a novel explicit formal syntax (Definition 5.2).

2 String Diagrams of Bimodular Categories

In algebra, a *bimodule* is a structure that has both a left and a right action such that they are compatible. *Bimodular categories are to bimodules what monoidal categories are to monoids*. This means that a bimodular category is a category, $\mathbb{C}$, acted on by two monoidal categories, $\mathbb{M}$ and $\mathbb{N}$ [53]. Bimodular categories are also known as "biactegories" [10, 36], while the name "bimodule category" has been reserved for actions of vector enriched categories with extra properties [19]. For our purposes, bimodular categories, $\mathbb{C}$, glue the string diagrams of their two monoidal categories, $\mathbb{M}$ and $\mathbb{N}$.

Definition 2.1. A *bimodular category* $(\mathbb{C}, \mathbb{M}, \mathbb{N})$ is a category $\mathbb{C}$ endowed with a left monoidal action $(\triangleright)\colon \mathbb{M} \times \mathbb{C} \to \mathbb{C}$, and a right monoidal action $(\triangleleft)\colon \mathbb{C} \times \mathbb{N} \to \mathbb{C}$. These two actions must be compatible, meaning that there exists a natural isomorphism, $\gamma_{M,N,X} : M \triangleright (X \triangleleft N) \longrightarrow (M \triangleright X) \triangleleft N$, such that all formal equations between these isomorphisms and the coherence isomorphisms of both monoidal categories and monoidal actions hold.

A bimodular category is a *strict bimodular category* whenever the two monoidal categories are strict, their two actions are strict and, moreover, the compatibility isomorphism is an identity. Every monoidal category $(\mathbb{C}, \otimes, I)$ is a $(\mathbb{C}, \mathbb{C})$-bimodular category with its own tensor product defining the two actions.

Proposition 2.2. *Strict bimodular categories over arbitrary strict monoidal categories form a category,* **sBimod**. *Morphisms $(F, H, K) \colon (\mathbb{C}, \mathbb{M}, \mathbb{N}) \to (\mathbb{D}, \mathbb{P}, \mathbb{Q})$ consist of two strict monoidal functors $H \colon \mathbb{M} \to \mathbb{P}$ and $K \colon \mathbb{N} \to \mathbb{Q}$ and a functor $F \colon \mathbb{C} \to \mathbb{D}$ that strictly preserves monoidal actions according to H and K.*

2.1 Signature of a Bimodular Category

The next sections prove that a variant of string diagrams is a sound and complete syntax for bimodular categories. String diagrams for bimodular categories consist of two monoidal regions glued by a bimodular wire. We first introduce a notion of bimodular signature (Definition 2.3) and then construct an adjunction (Theorem 2.8) using the notion of *collages*.

Definition 2.3. A *bimodular graph* $(\mathscr{A}, \mathscr{M}, \mathscr{N})$ (the bimodular analogue of a multigraph [48]) is given by three sets of objects $(\mathscr{A}_{obj}, \mathscr{M}_{obj}, \mathscr{N}_{obj})$ and three different types of edges:

- the left-acting edges, a set $\mathscr{M}(M_0, \dots, M_m; P_0, \dots, P_p)$ for each $M_0, \dots, M_m, P_0, \dots, P_p \in \mathscr{M}_{obj}$; and
- the right-acting edges, a set $\mathscr{N}(N_0, \dots, N_n; Q_0, \dots, Q_q)$ for each $N_0, \dots, N_n, Q_0, \dots, Q_q \in \mathscr{N}_{obj}$;
- the *central edges*, a set of edges $\mathscr{A}(M_0, \dots, M_m, A, N_0, \dots, N_n; O_0, \dots, P_p, B, Q_0, \dots, Q_q)$, for each $M_0, \dots, M_m, P_0, \dots, P_p \in \mathscr{M}_{obj}$; each $N_0, \dots, N_n, Q_0, \dots, Q_q \in \mathscr{N}_{obj}$ and each $A, B \in \mathscr{A}_{obj}$.

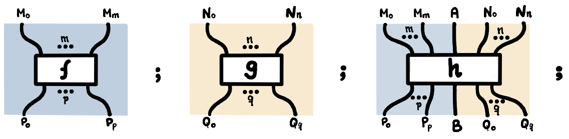


Figure 2: Left, right, and central edges of a bimodular graph.

Proposition 2.4. *Bimodular graphs form a category* **bmGraph**. *We define a morphism of bimodular graphs $(l, f, g) \colon (\mathscr{A}, \mathscr{M}, \mathscr{N}) \to (\mathscr{A}', \mathscr{M}', \mathscr{N}')$ to be a triple of functions on objects, $(l_{obj}, f_{obj}, g_{obj})$, that extend to the morphism sets. There exists a forgetful functor $U \colon$* **sBimod** $\to$ **bmGraph**.

Proof. See Appendix, Proposition B.3. □

So far we have described a syntactic presentation of strict bimodular categories. We would like to, additionally, go the other way and construct a free model from a syntactic presentation. Our approach is to note that the central edges in a bimodular graph can be considered as dividing the graph into two regions: one containing the left-acting vertices and edges and one containing the right-acting vertices and edges. Diagrams of this sort with multiple labelled regions can naturally be considered as string diagrams for bicategories: explicitly, the diagrams of the *collage* of the bimodular category.

2.2 The Collage of a Bimodular Category

Each profunctor induces a *collage category*; in an analogous fashion, a bimodular category induces a *collage bicategory*. This section proves that constructing the collage of a bimodular category is left adjoint to considering the bimodular hom-category between any two cells of a 2-category.

Definition 2.5. The *collage* of an $(\mathbb{M},\mathbb{N})$-bimodular category $\mathbb{C}$ is a bicategory, $\mathsf{Coll}_{\mathbb{C}}$. This bicategory has two 0-cells, M and N, and it is defined by the following hom-categories. Endocells on M are given by the monoidal category $\mathsf{Coll}_{\mathbb{C}}(M,M) = \mathbb{M}$; likewise, endocells on N are given by the monoidal category, $\mathsf{Coll}_{\mathbb{C}}(N,N) = \mathbb{N}$. The 1-cells from M to N are given by the category $\mathsf{Coll}_{\mathbb{C}}(M,N) = \mathbb{C}$; and composition of 1-cells is given by the monoidal actions. Finally, $\mathsf{Coll}_{\mathbb{C}}(N,M)$ is the empty category.

Definition 2.6. The category of strict bipointed 2-categories, $\mathbf{2Cat}_2$, has objects, $(\mathbb{A},M,N)$, given by a strict 2-category $\mathbb{A}$ and two chosen 0-cells on it, $M \in \mathbb{A}$ and $N \in \mathbb{A}$. A morphism of bipointed 2-categories is a strict 2-functor preserving the two chosen 0-cells.

Theorem 2.7. *There exists an adjunction between strict bimodular categories and bipointed 2-categories given by the collage,* $\mathsf{Coll}_{\mathbb{C}} \colon \mathbf{sBimod} \to \mathbf{2Cat}_2$, *and picking the hom-category between the chosen 0-cells,* $\mathsf{Chosen} \colon \mathbf{2Cat}_2 \to \mathbf{sBimod}$. *Moreover, the unit of this adjunction is a natural isomorphism.*

Proof. See Appendix, Theorem B.7. □

2.3 String Diagrams of Bimodular Categories, via Collages

We have the two ingredients for string diagrams of bimodular categories: string diagrams for bicategories, and collages, a way of embedding a bimodular category into a bicategory. This section combines both results to provide an adjunction from bimodular graphs to bimodular categories.

$$\begin{tikzcd}
\mathbf{2Graph} \arrow[d, bend right, "\mathsf{Str}"'] \arrow[d, phantom, "\dashv"] & \mathbf{2Graph}_2 \arrow[d, bend right, "\mathsf{Str}_2"'] \arrow[d, phantom, "\dashv"] \arrow[r, bend right, "u"'] \arrow[r, phantom, "\perp"] & \mathbf{bmGraph} \arrow[l, bend right, "i"'] \arrow[d, bend left, "\mathsf{bmStr}"] \arrow[d, phantom, "\vdash"] \\
\mathbf{2Cat} \arrow[u, bend right, "\mathsf{U}"'] & \mathbf{2Cat}_2 \arrow[u, bend right, "\mathsf{U}_2"'] \arrow[r, bend left, "\mathsf{Chosen}"] \arrow[r, phantom, "\top"] & \mathbf{sBimod} \arrow[l, bend left, "\mathsf{Coll}"] \arrow[u, bend left, "\mathsf{U}"]
\end{tikzcd}$$

Figure 3: Summary of adjunctions for the string diagrams of bimodular categories.

Theorem 2.8. *There exists an adjunction between bimodular graphs and strict bimodular categories. The left side of this adjunction is given by finding the bimodular category whose collage is the free 2-category on the bimodular graph,* $\mathsf{bmStr} \colon \mathbf{bmGraph} \to \mathbf{sBimod}$. *The right side of the adjunction is the previously mentioned forgetful functor* $\mathsf{U} \colon \mathbf{sBimod} \to \mathbf{bmGraph}$.

Proof. See Appendix, Theorem 2.8, the proof follows Figure 3. □

Remark 2.9. The string diagrams of bimodular categories particularize into the string diagrams of premonoidal and effectful categories. See the Appendix B.2 for details.

We have presented string diagrams for bimodular categories via the string diagrams of bicategories, and we will now give an example. We take inspiration from this first result to address now other syntaxes that depend on string diagrams of bicategories: the next section proposes string diagrams for functor boxes.

2.4 Example: Shared State

In the same way that premonoidal categories are particularly well-suited to describe stateful computations, bimodular categories are particularly well-suited to describe shared state between two processes. These two processes can be different and even live on different categories. As an example, consider the generators in Figure 4. They represent two different process theories (two different monoidal categories, $\mathbb{A}$ and $\mathbb{B}$) that access a common state with get and put operations.

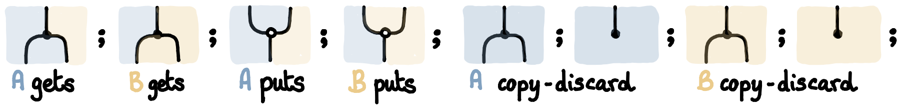


Figure 4: Signature for shared state.

In the same way that monoidal categories are a good setting where to define monoids and comonoids, bimodular categories are a good setting where to define bimodules. In order to capture interacting shared state, the generators of Figure 4 are quotiented by the equations of a pair of semifrobenius modules with compatible comonoid actions and semimonoid actions (see Appendix, Figure 14, for details).

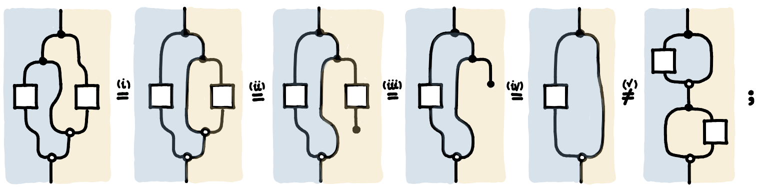


Figure 5: Race condition in bimodular string diagrams.

This setup is enough to exhibit one of the most salient features of shared state: *race conditions*. Race conditions were first studied by Huffman in 1954, who used diagrams to show how the behaviour of shared state is dependent on the relative timing of the actions of the parties [27]. We employ string diagrams of bimodular categories to show how two different timings of the actions – the leftmost and rightmost sides of the equation in Figure 5 – result in two different executions: even when the two get statements are compatible *(i)*, the two put statements interact causing the earlier of the two to be discarded *(ii,iii,iv)*; this causes the discrepancy with the intended protocol *(v)*.

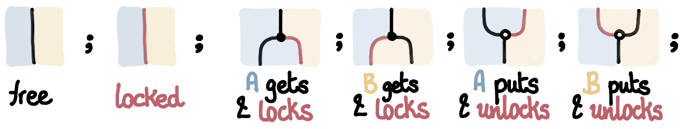


Figure 6: Binary semaphore in bimodular string diagrams.

Race conditions have a commonly accepted workaround: the *binary semaphore* [49]. Dijkstra described general semaphores with the aid of flow diagrams [18]; we use instead string diagrams of bimodular categories to implement a binary semaphore (Figure 6). We consider two different object generators for our bimodular category (free and locked): each operation must suitably lock or unlock the semaphore.

This renders race conditions ill-typed, and renders most of the interaction equations unnecessary (in the Appendix, Figure 14).

3 String Diagrams of Functor Boxes

Functor boxes are a extension of the string diagrammatic notation that represents plain functors, lax, oplax and strong monoidal functors. Functor boxes were introduced by Cockett and Seely [15] and later studied by Melliès [37]. We introduce here a syntactic presentation of (op)lax functor boxes that has the advantage of treating each piece of the box as a separate entity in a bicategory and apply the string diagrammatic calculus of bicategories.

3.1 Functor box signatures

Definition 3.1. A *functor box signature* $\mathscr{F} = (\mathscr{A}, \mathscr{X}, \mathscr{F}_\bullet, \mathscr{F}^\bullet)$ consists of a pair of sets, $\mathscr{A}_{obj}$ and $\mathscr{X}_{obj}$, and four different types of edges:

- the plain edges, $\mathscr{A}(A_0,\dots,A_n;B_0,\dots,B_m)$ for any objects $A_0,\dots,A_n,B_0,\dots,B_m \in \mathscr{A}_{obj}$;
- the functor box edges, $\mathscr{X}(X_0,...,X_n;Y_0,...,Y_m)$ for any objects $X_0,\dots,X_n,Y_0,\dots,Y_m \in \mathscr{X}_{obj}$;
- the in-box edges, $\mathscr{F}_\bullet(A_0,...,A_n;Y_0,...,Y_m)$ for any $A_0,...,A_n \in \mathscr{A}_{obj}$ and $Y_0,...,Y_m \in \mathscr{X}_{obj}$
- the out-box edges, $\mathscr{F}^\bullet(X_0,...,X_n;B_0,...,B_m)$ for any $B_0,...,B_m \in \mathscr{A}_{obj}$ and $X_0,...,X_n \in \mathscr{X}_{obj}$.

A *functor box signature morphism* $(h,k,l)\colon (\mathscr{A},\mathscr{X},\mathscr{F}) \to (\mathscr{B},\mathscr{Y},\mathscr{G})$ is a pair of functions between the object sets, $h_{obj}\colon \mathscr{A}_{obj} \to \mathscr{B}_{obj}$ and $k_{obj}\colon \mathscr{X}_{obj} \to \mathscr{Y}_{obj}$, that extend to a function between the edge sets;

- $h\colon \mathscr{A}(A_0,...,A_n;B_0,...,B_m) \to \mathscr{B}(h(A_0),...,h(A_n);h(B_0),...,h(B_m))$;
- $k\colon \mathscr{X}(X_0,...,X_n;Y_0,...,Y_m) \to \mathscr{Y}(k(X_0),...,k(X_n);k(Y_0),...,k(Y_m))$;
- $l_\bullet\colon \mathscr{F}_\bullet(A_0,...,A_n;Y_0,...,Y_m) \to \mathscr{G}_\bullet(h(A_0),...,h(A_n);k(Y_0),...,k(Y_m))$;
- $l^\bullet\colon \mathscr{F}^\bullet(X_0,...,X_n;B_0,...,B_m) \to \mathscr{G}^\bullet(k(X_0),...,k(X_n);h(B_0),...,h(B_m))$.

Functor box signatures and homomorphisms form a category, **Fbox**.

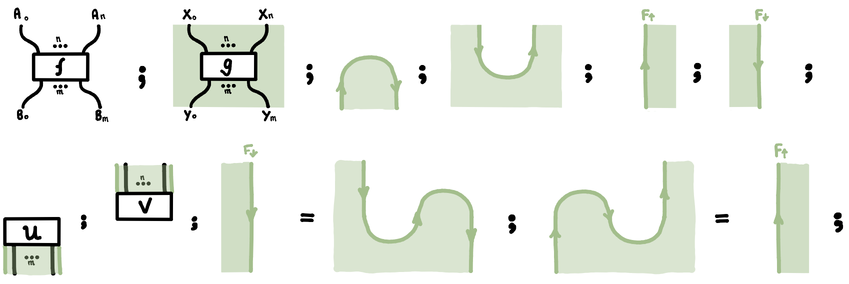


Figure 7: Syntactic bicategory of a lax monoidal functor box signature.

Definition 3.2. The syntactic bicategory of a functor box signature $\mathscr{F} = (\mathscr{A}, \mathscr{X}, \mathscr{F}_\bullet, \mathscr{F}^\bullet)$ is the bicategory freely presented by Figure 7, which we call $\mathbb{S}_{\mathscr{A},\mathscr{X},\mathscr{F}}$.

In other words, the bicategory $\mathbb{S}_{\mathscr{A},\mathscr{X},\mathscr{F}}$ contains exactly two 0-cells, labelled $\mathscr{A}$ and $\mathscr{X}$; it contains a 1-cell $A\colon \mathscr{A} \to \mathscr{A}$ for each $A \in \mathscr{A}_{obj}$, a 1-cell $X\colon \mathscr{X} \to \mathscr{X}$ for each $X \in \mathscr{X}_{obj}$ and, moreover, a pair of adjoint 1-cells $F^\uparrow\colon \mathscr{A} \to \mathscr{X}$ and $F^\downarrow\colon \mathscr{X} \to \mathscr{A}$. Finally, it contains a pair of 2-cells witnessing the adjunction $F^\uparrow \dashv F^\downarrow$, given by $n\colon \mathrm{id} \to F^\uparrow ⨾ F^\downarrow$ and $e\colon F^\downarrow ⨾ F^\uparrow \to \mathrm{id}$ which additionally satisfy the snake equations; and it also contains

- a 2-cell, $f \in \mathbb{S}(\mathscr{A},\mathscr{A})(A_0 ⨾ \dots ⨾ A_n;\ B_0 ⨾ \dots ⨾ B_m)$, for each *plain edge*;
- a 2-cell, $g \in \mathbb{S}(\mathscr{X},\mathscr{X})(X_0 ⨾ ... ⨾ X_n;\ Y_0 ⨾ ... ⨾ Y_m)$, for each *functor box edge*;
- a 2-cell, $u \in \mathbb{S}(\mathscr{A},\mathscr{A})(A_0 ⨾ ... ⨾ A_n;\ F^\uparrow ⨾ Y_0 ⨾ ... ⨾ Y_m ⨾ F^\downarrow)$ for each *in-box edge*; and
- a 2-cell, $v \in \mathbb{S}(\mathscr{A},\mathscr{A})(F^\uparrow ⨾ X_0 ⨾ ... ⨾ X_n ⨾ F^\downarrow;\ B_0 ⨾ \dots ⨾ B_m)$ for each *out-box edge*.

3.2 Lax Monoidal Functor Semantics

Definition 3.3 (Lax functors category)**.** An object of the *lax functors category*, **Lax**, is a pair of strict monoidal categories $(\mathbb{A},\mathbb{X})$ together with a lax monoidal functor between them, (F,ε,μ); that is, a functor $F\colon \mathbb{X} \to \mathbb{A}$ endowed with two natural transformations $\varepsilon\colon I \to FI$, and $\mu\colon FX \otimes FY \to F(X \otimes Y)$, satisfying associativity $(\mu \otimes id) ⨾ \mu = (id \otimes \mu) ⨾ \mu$, left unitality $(\varepsilon \otimes id) ⨾ \mu = id$ and right unitality $(id \otimes \varepsilon) ⨾ \mu = id$.

A morphism of the *lax functors category*, from $(\mathbb{A},\mathbb{X},F,\varepsilon_F,\mu_F)$ to $(\mathbb{B},\mathbb{Y},G,\varepsilon_G,\mu_G)$ is a pair of strict monoidal functors $H\colon \mathbb{X} \to \mathbb{A}$ and $K\colon \mathbb{A} \to \mathbb{B}$ such that $F ⨾ K = H ⨾ G$ and such that $K(\varepsilon_F) = \varepsilon_G$ and $K(\mu_F) = \mu_G$.

Theorem 3.4. *There exists an adjunction between the category of functor box signatures,* **Fbox***, and the category of pairs of strict monoidal categories with a lax monoidal functor between them,* **Lax***. The free side of this adjunction is given by the syntax of Figure 7.*

Proof. See Appendix, Theorem C.3. □

Collages, by themselves, explained the 2-region diagrams of bimodular categories; collages will also explain the two-region diagrams of functor boxes in Section 4.5. However, as currently defined, collages are only sufficient to encode the vertical boundaries. In order to additionally represent boundaries along the horizontal axis we can make use of profunctors between bimodular categories and extend our notion of collage to these structures (described in Appendix F). Following this thread we find that collages embed into a tricategory of pointed bimodular profunctors, described in the next section, which we consider a universe of interpretation for all of the graphical theories described.

4 Bimodular Profunctors

Where can we interpret all these string diagrams and provide compositional semantics for them? In this section, we introduce a single structure where all the previous calculi take semantics.

We will need two different ingredients: *coends* and *bimodularity*. Coends and profunctors [32, 33], far from being a obscure concept from category theory, can be seen as the right tool to glue together morphisms from different categories [17, 47]; we follow a explicitly *pointed* version of coend calculus, which keeps track of the transformation between profunctors we are constructing (Section 4.3). In a similar sense, bimodular categories tensor together objects from different monoidal categories. Both ideas combine into the calculus of pointed bimodular profunctors.

4.1 Bimodular Profunctors

Consider $\mathbb{C}$ and $\mathbb{D}$, both $(\mathbb{M},\mathbb{N})$-bimodular categories. A natural notion of morphism between them is a functor $\mathbb{C} \to \mathbb{D}$ which is linear in both actions. However, there is another notion of morphism between them, which is a generalisation of a profunctor between categories to this bimodular setting. Bimodular profunctors are a generalized reformulation of the Tambara modules of Pastro and Street [41].

Definition 4.1. Let $\mathbb{M}$ and $\mathbb{N}$ be two monoidal categories and let $\mathbb{C}$ and $\mathbb{D}$ be two $(\mathbb{M},\mathbb{N})$-bimodular categories. A *bimodular profunctor* from $\mathbb{C}$ to $\mathbb{D}$ is a profunctor $T \colon \mathbb{C}^{op} \times \mathbb{D} \to \mathrm{Set}$ with a natural family of strengths,

$$t_M : T(X,Y) \to T(M \triangleright X, M \triangleright Y), \quad \text{and} \quad t_N : T(X,Y) \to T(X \triangleleft N, Y \triangleleft N),$$

such that the actions are associative $t_M \mathbin{⨾} t_N = t_{M \otimes N}$, unital $t_I = id$, and compatible, $t_M \mathbin{⨾} t_N = t_N \mathbin{⨾} t_M$, up to the coherence isomorphisms of the monoidal category. See Appendix B for details.

Proposition 4.2. *For any pair of monoidal categories, $\mathbb{M}$ and $\mathbb{N}$, there is a bicategory ${}_{\mathbb{M}}\mathbf{Mod}_{\mathbb{N}}$ of $(\mathbb{M},\mathbb{N})$-bimodular categories, bimodular profunctors, and natural transformations between them.*

These will form the hom-bicategories of the tricategory we later define. The other significant piece of data we require is a family of tensors $\otimes : {}_{\mathbb{M}}\mathbf{Mod}_{\mathbb{N}} \times {}_{\mathbb{N}}\mathbf{Mod}_{\mathbb{O}} \to {}_{\mathbb{M}}\mathbf{Mod}_{\mathbb{O}}$, which we now study.

4.2 Tensor of Bimodular Profunctors

The tensor of bimodular categories is similar to the tensor of modules over a monoid in classical algebra: we consider pairs of elements and we quotient out the action of a common scalar. In this case, the quotienting is substituted by an appropiate structural isomorphism: the *equilibrator*.

Definition 4.3 (Tensor of bimodular categories)**.** Let $\mathbb{C}$ be a $(\mathbb{M},\mathbb{N})$-bimodular category and let $\mathbb{D}$ be a $(\mathbb{N},\mathbb{O})$-bimodular category. Their tensor product, $\mathbb{C} \otimes_{\mathbb{N}} \mathbb{D}$, is a category with the same objects as $\mathbb{C} \times \mathbb{D}$: we write them as $X \otimes_{\mathbb{N}} Y$. The category is presented by the morphisms of $\mathbb{C} \times \mathbb{D}$ and a free family of natural isomorphisms, called the *equilibrators*,

$$\tau_{X,N,Y} \colon (X \triangleleft N) \otimes_N Y \to X \otimes_N (N \triangleright Y), \text{ for each } N \in \mathbb{N}, X \in \mathbb{C}, Y \in \mathbb{D},$$

which are additionally quotiented by the following equations up to the structure isomorphisms of the monoidal actions, $\tau_{X,M \otimes N,Y} = \tau_{X \triangleleft M,N,Y} \mathbin{⨾} \tau_{X,M,N \triangleright Y}$, and $\tau_{X,I,Y} = \mathrm{id}$.

Definition 4.4. Let $\mathbb{C}$ and $\mathbb{C}'$ be two $(\mathbb{M},\mathbb{N})$-bimodular categories and let $\mathbb{D}$ and $\mathbb{D}'$ be a $(\mathbb{N},\mathbb{O})$-bimodular categories. Given two bimodular profunctors, $T \colon \mathbb{C} \to \mathbb{C}'$ and $R \colon \mathbb{D} \to \mathbb{D}'$, their tensor is a bimodular profunctor, $T \otimes_{\mathbb{N}} R \colon \mathbb{C} \otimes_{\mathbb{N}} \mathbb{D} \to \mathbb{C}' \otimes_{\mathbb{N}} \mathbb{D}'$, defined by

$$T \otimes_{\mathbb{N}} R(X \otimes_N Y; X' \otimes_N Y') = T(X;X') \times R(Y,Y')/(\sim),$$

where $(\sim)$ is the equivalence relation generated by $(t_N(x),y) \sim (x,t_N(y))$.

4.3 Pointed Profunctors

Profunctors deal with families of morphisms, and their natural isomorphisms determine correspondences between these families. However, when we use profunctors for the semantics of string diagrams, we most often want to single out a particular morphism between a particular pair of objects. A simple technique to achieve this is to use *pointed profunctors* instead of simply profunctors: this technique was explicitly described by this second author [47] although it has implicit appearances in the literature [3, 26].

Definition 4.5. A pointed profunctor $(P, p) \colon (\mathbb{A}, X) \to (\mathbb{B}, Y)$ between two pointed categories with a chosen object $X \in \mathbb{A}_{obj}$ and $Y \in \mathbb{B}_{obj}$ is a profunctor $P \colon \mathbb{A} \to \mathbb{B}$ together with an element $p \in P(A, B)$ of the profunctor evaluated on the chosen object of the categories.

From now on, we work using pointed profunctors instead of plain profunctors, see the Appendix Appendix D.1 for a short reference on "pointed coend calculus".

4.4 The Tricategory of Pointed Bimodular Profunctors

We call *collages of string diagrams* to the diagrams of the tricategory of pointed bimodular profunctors.

Definition 4.6. The tricategory of pointed bimodular profunctors, $\mathbb{BmProf}_{\mathsf{pt}}$, has as 0-cells the monoidal categories, $\mathbb{M}, \mathbb{N}, \mathbb{O}, \dots$. The 1-cells between two monoidal categories $\mathbb{M}$ and $\mathbb{N}$ are *pointed bimodular categories*, $(\mathbb{A}, \triangleright, \triangleleft, A)$, consisting of a $(\mathbb{M}, \mathbb{N})$-bimodular category with two actions $(\mathbb{A}, \triangleright, \triangleleft)$ and some object of that category, $A \in \mathbb{A}$. Pointed bimodular categories compose by the tensor of bimodular categories,

$$(\mathbb{A}, \triangleright, \triangleleft, A) \otimes_{\mathbb{N}} (\mathbb{B}, \triangleright, \triangleleft, B) = (\mathbb{A} \otimes_{\mathbb{N}} \mathbb{B}, \triangleright, \triangleleft, A \otimes_{\mathbb{N}} B).$$

The 2-cells between two pointed bimodular categories $(\mathbb{A}, \triangleright, \triangleleft, A)$ and $(\mathbb{B}, \triangleright, \triangleleft, B)$ are *pointed bimodular profunctors* (P, t, p), consisting of a profunctor $P \colon \mathbb{A} \to \mathbb{B}$ together with a point $p \in P(A, B)$ that are moreover bimodular with compatible natural transformations $t_M \colon P(A; B) \to P(M \triangleright A; M \triangleright B)$, and $t_N \colon P(A; B) \to P(A \triangleleft N; B \triangleleft N)$. These 2-cells compose by profunctor composition and by the tensor of bimodular profunctors.

Finally, the 3-cells between two pointed bimodular profunctors (P, t, p) and (Q, r, q) are bimodular natural transformations that preserve the point, consisting of a natural transformation $\alpha \colon P \to Q$ such that the $\alpha(p) = q$ and moreover $t_M \mathbin{⨾} \alpha = \alpha \mathbin{⨾} r_M$ and $t_N \mathbin{⨾} \alpha = \alpha \mathbin{⨾} r_N$.

Remark 4.7. At the moment of writing, it is unclear to the authors whether a string diagrammatic calculus for tricategories, described by transformations of the string diagrammatic calculus of bicategories, has been fully described and proved sound and complete. However, there seems to be consensus in that this would be the right language for tricategories: much literature assumes it. Let us close this section by tracking explicitly the assumptions we need to employ a diagrammatic syntax for bimodular profunctors.

Conjecture 4.8. *The previous data satisfies all coherence conditions of a tricategory. Moreover, we can reason with tricategories using the calculus of deformations of string diagrams, extending the string diagrams for quasistrict monoidal 2-categories of Bartlett [2].*

4.5 Functor Boxes via Collages of String Diagrams

The following Figure 8 details how to interpret functor boxes as collages of string diagrams. The colored region represents the domain of the lax monoidal functor; the white region represents the codomain. Morphisms of both categories are interpreted as elements of their respective hom-profunctors; and the laxators are used to merge colored regions. The only element that we will explicitly detail is the bimodular category that appears in the closing and opening wires of a functor box.

Proposition 4.9 (Bimodular categories of a lax monoidal functor)**.** *Let $\mathbb{X}$ and $\mathbb{A}$ be two monoidal categories and let $F \colon \mathbb{X} \to \mathbb{A}$ be a monoidal functor between them, endowed with natural transformations $\psi_0 \colon J \to FI$ and $\psi_2 \colon FX \otimes FY \to F(X \otimes Y)$. The following profunctors, $\mathbb{A} \rtimes_F \mathbb{X} \colon \mathbb{A} \times \mathbb{X} \to \mathbb{A} \times \mathbb{X}$ and*

$\mathbb{X} \ltimes_F \mathbb{A} \colon \mathbb{X} \times \mathbb{A} \to \mathbb{X} \times \mathbb{A}$ *determine two promonads, and therefore two Kleisli categories.*

$$\mathbb{A} \rtimes_F \mathbb{X}(A,X;B,Y) = \int^{M\in\mathbb{X}} \mathbb{A}(A;B\otimes FM) \times \mathbb{X}(M\otimes X;Y);$$
$$\mathbb{X} \ltimes_F \mathbb{A}(X,A;Y,B) = \int^{M\in\mathbb{X}} \mathbb{A}(A;FM\otimes B) \times \mathbb{X}(M\otimes A;B);$$

These two Kleisli categories are $(\mathbb{A},\mathbb{X})$ *and* $(\mathbb{X},\mathbb{A})$*-bimodular, respectively.*

Proof. See Appendix, Proposition D.4. The construction uses the laxity of the monoidal functor. □

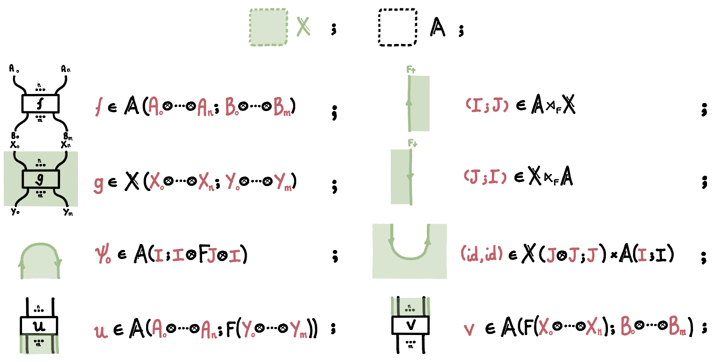

X ;
A ;
f ∈ A(A₀⊗···⊗Aₙ; B₀⊗···⊗Bₘ) ;
(I;J) ∈ A⋊F X ;
g ∈ X(X₀⊗···⊗Xₙ; Y₀⊗···⊗Yₘ) ;
(J;I) ∈ X⋉F A ;
ψ₀ ∈ A(I; I⊗FJ⊗I) ;
(id,id) ∈ X(J⊗J;J)×A(I;I) ;
u ∈ A(A₀⊗···⊗Aₙ; F(Y₀⊗···⊗Yₘ)) ;
v ∈ A(F(X₀⊗···⊗Xₙ); B₀⊗···⊗Bₘ) ;

Figure 8: Semantics for functor boxes in terms of pointed bimodular profunctors.

5 String Diagrams of Internal Diagrams

The tubular 3-dimensional cobordisms of internal diagrams are first described as a Frobenius algebra by Bartlett, Douglas, Schommer-Pries and Vicary [3]. We are indebted to this first introduction, which made internal diagrams into a convenient graphical notation in topological quantum field theory [3]. Internal diagrams themselves were later given a expiclit semantics in a monoidal bicategory of pointed profunctors; this was the subject of this second author's contribution to *Applied Category Theory 2020* [45]. An important aspect of the syntax of internal diagrams is their 3-dimensional nature: the syntax not only contains string diagrams, but also reductions between them.

We introduce here a novel syntactic presentation of *internal diagrams* that has the advantage of treating each piece of an internal diagram (including the closing and opening of tubes) as a separate entity in a tricategory. As a consequence, we are later able to introduce for the first time a more refined semantics in terms of a tricategory of *pointed bimodular profunctors*.

Definition 5.1. A *polygraph*, $\mathscr{G}$, is the signature for the string diagrams of a monoidal category. It consists of a set of objects, $\mathscr{G}_{obj}$, and a set of morphisms $\mathscr{G}(A_0,...,A_n;B_0,...,B_m)$ between any two lists of objects, $A_0,...,A_n,B_0,...,B_m \in \mathscr{G}_{obj}$.

Definition 5.2. The *syntactic tricategory of internal diagrams* over a polygraph $\mathscr{G}$ is the tricategory $\mathbf{G}$ presented by the cells in Figure 9. In other words, it contains two 0-cells, $\mathscr{I}$ and $\mathscr{G}$, in white and blue in the figure, respectively. It contains a 1-cell $A\colon \mathscr{G} \to \mathscr{G}$ for each object $A \in \mathscr{G}_{obj}$ and two 1-cells,

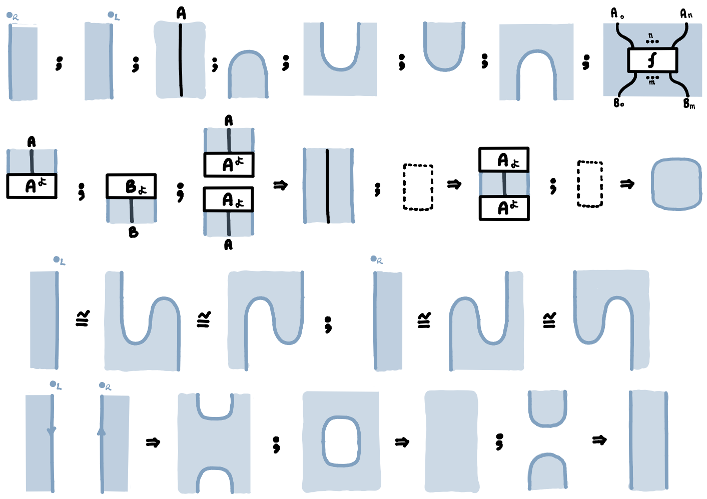


Figure 9: Syntax for open internal diagrams.

$L_\bullet \colon \mathscr{I} \to \mathscr{G}$ and $R_\bullet \colon \mathscr{G} \to \mathscr{I}$ forming two 2-adjunctions $(L_\bullet) \dashv (R_\bullet)$ and $(R_\bullet) \dashv (L_\bullet)$ up to a 3-cell. It contains the following 2-cells,

- two 2-cells $n_1 \colon \mathrm{id} \to L_\bullet \mathbin{⨾} R_\bullet$ and $e_1 \colon R_\bullet \mathbin{⨾} L_\bullet \to \mathrm{id}$ witnessing the 2-adjunction $(L_\bullet) \dashv (R_\bullet)$ and two 2-cells $n_2 \colon 1 \to R_\bullet \mathbin{⨾} L_\bullet$ and $e_2 \colon L_\bullet \mathbin{⨾} R_\bullet \to \mathrm{id}$ witnessing the 2-adjunction $(R_\bullet) \dashv (L_\bullet)$ – see Vicary and Heunen [24] for a reference on 2-adjunctions and the swallowtail equations;
- two 2-cells, $A^{\dagger} \colon L_\bullet \mathbin{⨾} A \mathbin{⨾} R_\bullet \to \mathrm{id}$ and $A_{\dagger} \colon \mathrm{id} \to L_\bullet \mathbin{⨾} A \mathbin{⨾} R_\bullet$, forming an adjunction $A^{\dagger} \dashv A_{\dagger}$ for each object $A \in \mathscr{G}_{obj}$; and a 2-cell, $f \colon A_0 \mathbin{⨾} \ldots \mathbin{⨾} A_n \to B_0 \mathbin{⨾} \ldots \mathbin{⨾} B_m$, for each edge $f \in \mathscr{G}(A_0, \ldots, A_n; B_0, \ldots, B_m)$.

Finally, it contains the following 3-cells,

- two invertible 3-cells, $\alpha_1 \colon (1 \otimes n_1) \mathbin{⨾} (e_1 \otimes 1) \to 1$ and $\beta_1 \colon (n_1 \otimes 1) \mathbin{⨾} (1 \otimes e_1) \to 1$, witnessing the 2-adjunction $(L_\bullet) \dashv (R_\bullet)$ and satisfying the swallowtail equations; and two invertible 3-cells, $\alpha'_2 \colon (1 \otimes n_2) \mathbin{⨾} (e_2 \otimes 1) \to 1$ and $\beta_2 \colon (n_1 \otimes 1) \mathbin{⨾} (1 \otimes e_1) \to 1$, witnessing the 2-adjunction $(R_\bullet) \dashv (L_\bullet)$ and and satisfying the swallowtail equations;
- two 3-cells, $c \colon A^{\dagger} \mathbin{⨾} A_{\dagger} \to 1$ and $i \colon 1 \to A_{\dagger} \mathbin{⨾} A^{\dagger}$, witnessing the adjunction $A^{\dagger} \dashv A_{\dagger}$ and satisfying the snake equations;
- two 3-cells, $u_i \colon n_1 \mathbin{⨾} e_2 \to 1$ and $v_i \colon 1 \to e_2 \mathbin{⨾} n_1$ witnessing an adjunction $e_2 \dashv n_1$ and satisfying the snake equations; two 3-cells $u_j \colon 1 \to n_2 \mathbin{⨾} e_1$ and $v_i \colon e_1 \mathbin{⨾} n_2 \to 1$ witnessing an adjunction $n_2 \dashv e_1$ and satisfying the snake equations.

Theorem 5.3. *There is a 3-functor from the syntactic tricategory of internal diagrams into pointed bimodular profunctors for any interpretation of the polygraph into a monoidal category.*

Proof. See Appendix, Theorem E.1. □

Remark 5.4. This syntax can be exemplified by evaluating a quantum comb [12], or a monoidal lens [44] with a morphism, in terms of internal string diagrams [26], see Figure 10. It has been used more generally to reason about coends in monoidal categories [47] and topological quantum field theory [3].

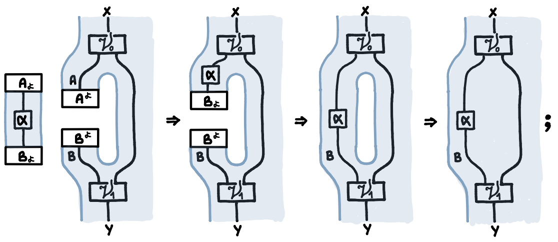


Figure 10: Evaluating a comb in terms of internal string diagrams.

6 Conclusions

Collages of string diagrams provide an abundant graphical calculus. Functor boxes, tensors of bimodular categories and internal diagrams all exist in the graphical calculus of collages. Their technical undepinning is complex: we characterized them as diagrams of pointed bimodular profunctors, but these arrange themselves into a tricategory, which may be difficult to reason about.

Apart from introducing the technique of collages and formalizing multiple extensions to string diagrams, we would like to call the attention to the techniques we use: most of our results on soundness and completeness of diagrams are arranged into adjunctions, which allows us to prove them by reusing the better known results on soundness and completenss for monoidal categories and bicategories.

Related work. An important line of research revolves around *module categories* and *fusion categories*, some specific enriched categories with actions with applications in topological quantum field theories [19, 20, 40]. Specially relevant and recent is Hoek's work, which constructs diagrams for a bimodule category [25, Theorem 3.5.2]. We follow the more elementary notion of bimodular category, called "*biactegory*" in the taxonomy of Capucci and Gavranović [10]. Cockett and Pastro [14] have used instead *linear actions* for concurrency, and even when we take inspiration from their work, their approach is more sophisticated and expressive than our toy example demonstrating bimodular categories (Figure 5).

Most work has been presented for some particular cases of collages: functor boxes have been extensively employed, but never reduced to string diagrams [15, 37]; internal diagrams have served both quantum theory and category theory [3, 26, 31], and can be given semantics into pointed profunctors [45], but again a presentation as string diagrams was missing. A convenient algebra of lenses [44], a particular type of incomplete diagram, has been recently introduced [21], but this is still independent of the semantics of arbitrary internal diagrams.

Finally, the first author has published a blog post that accompanies this manuscript [9].

Further work. It should be possible to "destrictify" many of the results of this paper. We have only presented a 1-adjunction between strict bimodular categories and bipointed 2-categories; but a higher adjunction would allow us to reuse coherence for bicategories to automatically obtain coherence for bimodular categories. We have marked along the paper the conjectures where further work is warranted.

We conjecture pointed bimodular profunctors to also form a compact closed tricategory, with the dual of each monoidal category being the *reverse monoidal category*, $A \otimes_{Rev} B = B \otimes A$. Even when it may be conceptually clear what a compact tricategory should be, it is technically challenging to come up with a concrete definition for it in terms of coherence equations.

Acknowledgements

The authors want to thank David A. Dalrymple for discussion on the string diagrammatic interpretation of functor boxes; and Matteo Capucci for several insightful conversations about notions of 2-dimensional profunctor, that helped us understand how to tie disparate aspects of this story together.

Dylan Braithwaite was supported by an Industrial CASE studentship from the UK Engineering and Physical Sciences Research Council (EPSRC) and the National Physical Laboratory. Mario Román was supported by the European Union through the ESF Estonian IT Academy research measure (2014-2020.4.05.19-0001).

References

[1] Mario Alvarez-Picallo, Dan R. Ghica, David Sprunger & Fabio Zanasi (2021): *Functorial String Diagrams for Reverse-Mode Automatic Differentiation*. arXiv:2107.13433.

[2] Bruce Bartlett (2014): *Quasistrict symmetric monoidal 2-categories via wire diagrams*. arXiv:1409.2148.

[3] Bruce Bartlett, Christopher L. Douglas, Christopher J. Schommer-Pries & Jamie Vicary (2015): *Modular categories as representations of the 3-dimensional bordism 2-category*. arXiv:1509.06811.

[4] Jean Bénabou (1967): *Introduction to bicategories*. In: *Reports of the Midwest Category Seminar*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 1–77.

[5] Guillaume Boisseau & Jeremy Gibbons (2018): *What you needa know about Yoneda: profunctor optics and the Yoneda lemma (functional pearl)*. *Proc. ACM Program. Lang.* 2(ICFP), pp. 84:1–84:27, doi:10.1145/3236779. Available at `https://doi.org/10.1145/3236779`.

[6] Guillaume Boisseau & Pawel Sobocinski (2021): *String Diagrammatic Electrical Circuit Theory*. In Kohei Kishida, editor: *Proceedings of the Fourth International Conference on Applied Category Theory, ACT 2021, Cambridge, United Kingdom, 12-16th July 2021*, *EPTCS* 372, pp. 178–191, doi:10.4204/EPTCS.372.13. Available at `https://doi.org/10.4204/EPTCS.372.13`.

[7] Joe Bolt, Jules Hedges & Philipp Zahn (2019): *Bayesian open games*. *CoRR* abs/1910.03656. arXiv:1910.03656.

[8] Filippo Bonchi, Joshua Holland, Robin Piedeleu, Paweł Sobociński & Fabio Zanasi (2019): *Diagrammatic algebra: from linear to concurrent systems*. *Proc. ACM Program. Lang.* 3(POPL), pp. 25:1–25:28, doi:10.1145/3290338. Available at `https://doi.org/10.1145/3290338`.

[9] Dylan Braithwaite (2023): *Diagrams for Actegories*. Available at `{https://dylanbraithwaite.github.io/2023/01/31/diagrams-for-actegories.html}`.

[10] Matteo Capucci & Bruno Gavranović (2022): *Actegories for the working amthematician*. *arXiv preprint arXiv:2203.16351*.

[11] Dimitri Chikhladze (2015): *Lax formal theory of monads, monoidal approach to bicategorical structures and generalized operads*. arXiv:1412.4628.

[12] Giulio Chiribella, Giacomo Mauro D'Ariano & Paolo Perinotti (2009): *Theoretical framework for quantum networks*. *Phys. Rev. A* 80, p. 022339, doi:10.1103/PhysRevA.80.022339. Available at `https://link.aps.org/doi/10.1103/PhysRevA.80.022339`.

[13] Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore & Mario Román (2022): *Profunctor Optics, a Categorical Update*. arXiv:2001.07488.

[14] J. Robin B. Cockett & Craig A. Pastro (2009): *The logic of message-passing*. *Sci. Comput. Program.* 74(8), pp. 498–533, doi:10.1016/j.scico.2007.11.005. Available at `https://doi.org/10.1016/j.scico.2007.11.005`.

[15] Robin B. Cockett & R. A. G. Seely (1999): *Linearly distributive functors*. *Journal of Pure and Applied Algebra* 143(1-3), pp. 155–203.

[16] Bob Coecke, Tobias Fritz & Robert W. Spekkens (2016): *A mathematical theory of resources*. *Inf. Comput.* 250, pp. 59–86, doi:10.1016/j.ic.2016.02.008. Available at `https://doi.org/10.1016/j.ic.2016.02.008`.

[17] Elena Di Lavore, Giovanni de Felice & Mario Román (2022): *Monoidal Streams for Dataflow Programming*. In: *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3531130.3533365.

[18] Edsger W. Dijkstra (1962): *Over de sequentialiteit van procesbeschrijvingen*. *Unpublished. Transcribed by Gerrit Jan Veltink for the E.W. Dijkstra Archive, Center for American History.* Available at `https://www.cs.utexas.edu/users/EWD/ewd00xx/EWD35.PDF`.

[19] Christopher L. Douglas, Christopher Schommer-Pries & Noah Snyder (2019): *The balanced tensor product of module categories*. *Kyoto Journal of Mathematics* 59(1), doi:10.1215/21562261-2018-0006. Available at `https://doi.org/10.1215%2F21562261-2018-0006`.

[20] Vladimir Drinfeld, Shlomo Gelaki, Dmitri Nikshych & Victor Ostrik (2010): *On braided fusion categories I*. *Selecta Mathematica* 16(1), pp. 1–119, doi:10.1007/s00029-010-0017-z.

[21] Matt Earnshaw, James Hefford & Mario Román (2023): *The Produoidal Algebra of Process Decomposition*. arXiv:2301.11867.

[22] Neil Ghani, Jules Hedges, Viktor Winschel & Philipp Zahn (2018): *Compositional Game Theory*. In: *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, Association for Computing Machinery, New York, NY, USA, p. 472–481, doi:10.1145/3209108.3209165. Available at `https://doi.org/10.1145/3209108.3209165`.

[23] James Hefford & Cole Comfort (2022): *Coend Optics for Quantum Combs*. arXiv:2205.09027.

[24] Chris Heunen & Jamie Vicary (2019): *Categories for Quantum Theory: an introduction*. Oxford University Press.

[25] Keeley Hoek (2019): *Drinfeld centers for bimodule categories*. Ph.D. thesis, MSc. thesis, The Australian National University.

[26] Nick Hu & Jamie Vicary (2021): *Traced Monoidal Categories as Algebraic Structures in Prof*. In Ana Sokolova, editor: *Proceedings 37th Conference on Mathematical Foundations of Programming Semantics, MFPS 2021, Hybrid: Salzburg, Austria and Online, 30th August - 2nd September, 2021*, *EPTCS* 351, pp. 84–97, doi:10.4204/EPTCS.351.6. Available at `https://doi.org/10.4204/EPTCS.351.6`.

[27] David A Huffman (1954): *The Synthesis of Sequential Switching Circuits*. *Journal of the Franklin Institute* 257(3), pp. 161–190.

[28] Niles Johnson & Donald Yau (2020): *2-Dimensional Categories*. arXiv:2002.06055.

[29] André Joyal & Ross Street (1991): *The geometry of tensor calculus, I*. *Advances in Mathematics* 88(1), pp. 55–112, doi:10.1016/0001-8708(91)90003-P. Available at `https://www.sciencedirect.com/science/article/pii/000187089190003P`.

[30] Paul Blain Levy (2022): *Call-by-Push-Value*. *ACM SIGLOG News* 9(2), p. 7–29, doi:10.1145/3537668.3537670.

[31] Leo Lobski & Fabio Zanasi (2022): *String Diagrams for Layered Explanations*. *CoRR* abs/2207.03929, doi:10.48550/arXiv.2207.03929. arXiv:2207.03929.

[32] Fosco Loregian (2021): *(Co)end Calculus*. London Mathematical Society Lecture Note Series, Cambridge University Press, doi:10.1017/9781108778657.

[33] Saunders Mac Lane (1971): *Categories for the Working Mathematician*. *Graduate Texts in Mathematics* 5, Springer Verlag, doi:10.1007/978-1-4757-4721-8.

[34] Saunders Mac Lane (1978): *Categories for the Working Mathematician*. Graduate Texts in Mathematics, Springer New York, doi:10.1007/978-1-4757-4721-8.

[35] Daniel Marsden (2014): *Category theory using string diagrams*. *arXiv preprint arXiv:1401.7220*.

[36] Paddy McCrudden (2000): *Categories of representations of coalgebroids*. *Advances in Mathematics* 154(2), pp. 299–332.

[37] Paul-André Melliès (2006): *Functorial Boxes in String Diagrams*. In Zoltán Ésik, editor: *Computer Science Logic, 20th International Workshop, CSL 2006, 15th Annual Conference of the EACSL, Szeged, Hungary, September 25-29, 2006, Proceedings*, *Lecture Notes in Computer Science* 4207, Springer, pp. 1–30, doi:10.1007/11874683_1. Available at `https://doi.org/10.1007/11874683_1`.

[38] Eugenio Moggi (1991): *Notions of Computation and Monads*. *Inf. Comput.* 93(1), pp. 55–92, doi:10.1016/0890-5401(91)90052-4.

[39] Chad Nester (2020): *A Foundation for Ledger Structures*. In Emmanuelle Anceaume, Christophe Bisière, Matthieu Bouvard, Quentin Bramas & Catherine Casamatta, editors: *2nd International Conference on Blockchain Economics, Security and Protocols, Tokenomics 2020, October 26-27, 2020, Toulouse, France*, *OASIcs* 82, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 7:1–7:13, doi:10.4230/OASIcs.Tokenomics.2020.7. Available at `https://doi.org/10.4230/OASIcs.Tokenomics.2020.7`.

[40] Victor Ostrik (2003): *Module categories, weak Hopf algebras and modular invariants*. *Transformation groups* 8, pp. 177–206.

[41] Craig Pastro & Ross Street (2007): *Doubles for monoidal categories*. *arXiv preprint arXiv:0711.1859*.

[42] Evan Patterson, David I. Spivak & Dmitry Vagner (2021): *Wiring diagrams as normal forms for computing in symmetric monoidal categories*. *Electronic Proceedings in Theoretical Computer Science* 333, pp. 49–64, doi:10.4204/eptcs.333.4. Available at `https://doi.org/10.4204%2Feptcs.333.4`.

[43] Maciej Piróg & Nicolas Wu (2016): *String diagrams for free monads (functional pearl)*. In Jacques Garrigue, Gabriele Keller & Eijiro Sumii, editors: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18-22, 2016*, ACM, pp. 490–501, doi:10.1145/2951913.2951947. Available at `https://doi.org/10.1145/2951913.2951947`.

[44] Mitchell Riley (2018): *Categories of optics*. *arXiv preprint arXiv:1809.00738*.

[45] Mario Román (2020): *Comb Diagrams for Discrete-Time Feedback*. *CoRR* abs/2003.06214. arXiv:2003.06214.

[46] Mario Román (2022): *Promonads and String Diagrams for Effectful Categories*. *CoRR* abs/2205.07664, doi:10.48550/arXiv.2205.07664. arXiv:2205.07664.

[47] Mario Román (2021): *Open Diagrams via Coend Calculus*. *Electronic Proceedings in Theoretical Computer Science* 333, p. 65–78, doi:10.4204/eptcs.333.5. Available at `http://dx.doi.org/10.4204/EPTCS.333.5`.

[48] Michael Shulman (2016): *Categorical logic from a categorical point of view*. *Available on the web*. Available at `https://mikeshulman.github.io/catlog/catlog.pdf`.

[49] Abraham Silberschatz, Peter Baer Galvin & Greg Gagne (2018): *Operating System Concepts, 10th Edition*. Wiley. Available at `http://os-book.com/OS10/index.html`.

[50] David I. Spivak (2022): *Generalized Lens Categories via functors* $\mathscr{C}^{\mathrm{op}} \to \mathsf{Cat}$. arXiv:1908.02202.

[51] Sam Staton & Paul Blain Levy (2013): *Universal properties of impure programming languages*. In Roberto Giacobazzi & Radhia Cousot, editors: *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '13, Rome, Italy - January 23 - 25, 2013*, ACM, pp. 179–192, doi:10.1145/2429069.2429091.

[52] Ross Street (1981): *Cauchy characterization of enriched categories*. *Rendiconti del Seminario Matematico e Fisico di Milano* 51(1), pp. 217–233.

[53] Zoran Škoda (2009): *Some equivariant constructions in noncommutative algebraic geometry*. *Georgian Mathematical Journal* 16(1), pp. 183–202. arXiv:0811.4770.

A Preliminaries

Proposition A.1 (Reducing an adjunction). *Let $F\colon \mathbb{A} \to \mathbb{C}$ and $H ⨟ U\colon \mathbb{C} \to \mathbb{A}$ determine an adjunction $(F, H ⨟ U, \eta, \varepsilon)$ and let $P\colon \mathbb{B} \to \mathbb{C}$ determine a second adjunction (P, H, u, c) such that the unit $u\colon I \to P ⨟ H$ is a natural isomorphism (as in Figure 11). Then, $F ⨟ H$ is left adjoint to U.*

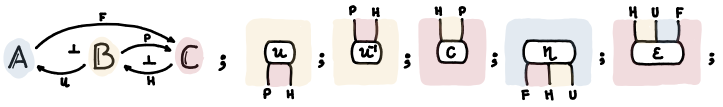


Figure 11: Setting for reducing an adjunction.

Proof. We employ the string diagrammatic calculus of bicategories to the bicategory of categories, functors and natural transformations [35]. We define the morphisms in Figure 12 to be the unit and the counit of the adjunction. We then prove that they satisfy the snake equations in Figures 12 and 13.

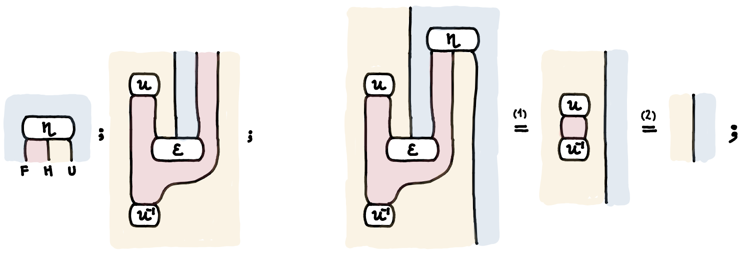


Figure 12: Unit and counit of the reduced adjunction (left). First snake equation (right).

In the first snake equation, in Figure 12, we use *(i)* that there is a duality (η, ε), and *(ii)* that u is invertible. In the second snake equation, in Figure 13, we use *(i)* that there is a duality (u, c), *(ii)* that u is invertible, *(iii)* that there is a duality (u, c), again; and *(iv)* that there is a duality (η, ε).

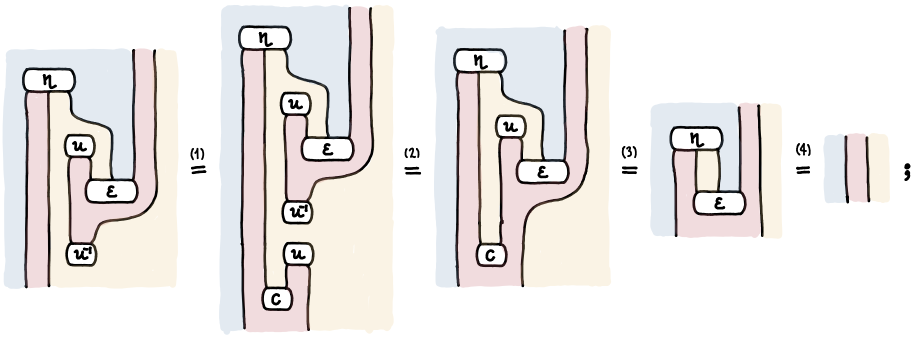


Figure 13: Second snake equation.

□

Proposition A.2. *Let $L\colon \mathbb{C} \to \mathbb{D}$ and $R\colon \mathbb{D} \to \mathbb{C}$ determine an adjunction (L,R,η,ε). For each object $A \in \mathbb{C}_{obj}$, this induces an adjunction between the coslice categories $\mathbb{C}\setminus A$ and $\mathbb{D}\setminus LA$.*

Proof. The functor $L_A\colon \mathbb{C}\setminus A \to \mathbb{D}\setminus LA$ is just an application of the functor L. The functor $R_A\colon \mathbb{D}\setminus LA \to \mathbb{C}\setminus A$ is defined using the unit of the adjunction as

$$R_A\left(LA \xrightarrow{f} B\right) = \left(A \xrightarrow{\eta} RLA \xrightarrow{Rf} B\right).$$

Note that a morphism $\alpha\colon LB \to C$ makes the first diagram in commute if and only if its adjunct, $\alpha^*\colon B \to RC$, makes the second diagram in commute.

$$\begin{tikzcd}
LB \arrow[r, "\alpha"] & C & & B \arrow[r, "\alpha^*"] & RC \\
A \arrow[u, "Lf"] \arrow[ur, "g"'] & & & A \arrow[u, "f"] \arrow[r, "\eta"] & RLA \arrow[u, "Rg"']
\end{tikzcd}$$

This is because two morphisms are equal if and only if their adjuncts are. We have that $(Lf \fatsemi \alpha)^* = f \fatsemi \alpha^*$ and $g^* = \eta \fatsemi Rg$. This induces the hom-set isomorphism of the adjunction, where each morphism is again adjunct to the same adjunct it was before. □

A.1 Bicategories

In the same way that a polygraph represents the signature for a monoidal category, a *2-graph* is the signature that allows us to freely generate a 2-category.

Definition A.3. A 2-graph $\mathscr{G}$ is given by a set of vertices, $\mathscr{G}_{obj}$; a set of edges between any two vertices, $\mathscr{G}(X;Y)$ for $X,Y \in \mathscr{G}_{obj}$; and a set of 2-edges for each pair of paths of vertices with the same source and target. That is, there is a set of 2-edges $\mathscr{G}(X;Y)(A_0,\dots,A_n;B_0,\dots,B_m)$, for each path $A_0 \in \mathscr{G}(X;U_0),\dots,A_n \in \mathscr{G}(U_{n-1};Y)$ and each path $B_0 \in \mathscr{G}(X;V_0),\dots,B_m \in \mathscr{G}(V_{m-1};Y)$.

A homomorhpism of 2-graphs is a family of functions on vertices, edges, and 2-edges that preserve their sources and targets; these form a category **2Graph** endowed with a forgetful functor $\mathsf{U}\colon$ **2Cat** $\to$ **2Graph** from the category of 2-categories and 2-functors.

Correctness of string diagrams for 2-categories [4, 35] amounts the fact that free 2-category over a 2-graph is given by string diagrams. It is difficult to find a proof for this exact result in the literature, but the widespread use of string diagrams for bicategories suggests that it is commonly accepted.

Theorem A.4 (String diagrams for bicategories)**.** *There exists an adjunction between 2-graphs and 2-categories given by progressive string diagrams* $\mathsf{Str}\colon$ **2Graph** $\to$ **2Cat** *and the previously mentioned forgetful functor* $\mathsf{U}\colon$ **2Cat** $\to$ **2Graph**.

A.2 Profunctors

Definition A.5. A *profunctor* $(P,<,>)$ between two categories $\mathbb{A}$ and $\mathbb{B}$ is a family of sets $P(A,B)$ indexed by objects $\mathbb{A}$ and $\mathbb{B}$, and endowed with jointly functorial left and right actions of the morphisms of $\mathbb{A}$ and $\mathbb{B}$, respectively. Explicitly, types of these actions are $(>)\colon \hom(A',A) \times P(A',B) \to P(A,B)$, and $(<)\colon \hom(B,B') \times P(A,B) \to P(A,B')$. These must satisfy

- compatibility, $(f > p) < g = f > (p < g)$,
- preserve identities, $id > p = p$, and $p < id = p$,
- and preserve composition, $(p < f) < g = p < (f \fatsemi g)$ and $f > (g > p) = (f \fatsemi g) > p$.

More succintly, a profunctor $P\colon \mathbb{A} \to \mathbb{B}$ is a functor $P\colon \mathbb{A}^{op} \times \mathbb{B} \to \mathbf{Set}$. When presented as a family of sets with a pair of actions, profunctors have been sometimes called *bimodules*. We avoid this term, which we reserve for the classical algebraic notion, and its categorification.

B Bimodular categories

We provide here an alternative spelled-out definition of bimodular profunctors.

Definition B.1. A *bimodular profunctor* from $\mathbb{C}$ to $\mathbb{D}$, a pair of $(\mathbb{M}, \mathbb{N}, \triangleright, \triangleleft)$-bimodular categories, is a profunctor $(P, <, >)\colon \mathbb{C} \to \mathbb{D}$ endowed with a pair of natural whiskering operators,

$$(\gtrdot)\colon \mathbb{M}(X;X') \times P(A;B) \to P(X \triangleright A; X' \triangleright B), \quad \text{and} \quad (\lessdot)\colon P(A;B) \times \mathbb{N}(Y;Y') \to P(A \triangleleft Y; B \triangleleft Y'),$$

satisfying the following axioms up to coherence of the monoidal actions.

$$\begin{aligned}
&(m \gtrdot f) < (m' \triangleright g) = (m \fatsemi m') \gtrdot (f < g), && (f \lessdot n) < (g \triangleleft n') = (f < g) \lessdot (n \fatsemi n'),\\
&(m' \triangleright g) > (m \gtrdot f) = (m' \fatsemi m) \gtrdot (g > f), && (g \triangleleft n') > (f \lessdot n) = (g > f) \lessdot (n' \fatsemi n),\\
&\mathrm{id} \gtrdot f = f = f \lessdot \mathrm{id}, && u \gtrdot (f \lessdot v) = (u \gtrdot f) \lessdot v,\\
&u \gtrdot (v \gtrdot f) = (u \otimes v) \gtrdot f, && (f \lessdot u) \lessdot v = f \lessdot (u \otimes v).
\end{aligned}$$

Remark B.2. Not only is this definition a natural extension of profunctors as category-bimodules to bimodular categories, but the strengths of bimodular profunctors can be seen as an additional bimodule structure in the form of an action of the monoidal categories on the profunctor.

$$\begin{tikzcd}
 & \mathbb{C} \arrow[d, dashed, "\text{functorial action}"] & \\
\mathbb{M} \arrow[ru, dashed, bend left, "\text{monoidal action}"] \arrow[r, dashed, "\text{left strength}"'] \arrow[rd, dashed, bend right] & T & \mathbb{N} \arrow[l, dashed] \arrow[lu, dashed, bend right] \arrow[ld, dashed, bend left] \\
 & \mathbb{D} \arrow[u, dashed] &
\end{tikzcd}$$

The vertical composition of bimodular profunctors is given by ordinary profunctor composition while the strength is obtained as the cartesian product of the two constituent strengths.

$$\begin{tikzcd}
 & \mathbb{C} \arrow[rd] \arrow[d, phantom, "\Downarrow P"] & & & & \mathbb{C} \arrow[rd] \arrow[dd, phantom, "\Downarrow P\odot Q"] & \\
\mathbb{M} \arrow[ru, no head] \arrow[r, no head] \arrow[rd, no head] & \mathbb{D} \arrow[r] \arrow[d, phantom, "\Downarrow Q"] & \mathbb{N} \arrow[r, mapsto] & \mathbb{M} \arrow[rru, no head, bend left] \arrow[rrd, no head, bend right] & & & \mathbb{N} \\
 & \mathbb{E} \arrow[ru] & & & & \mathbb{E} \arrow[ru] &
\end{tikzcd}$$

As with ordinary profunctors, this composition is not strictly associative or unital but indeed there is additionally a natural notion of morphism between bimodular profunctors with which structure maps can be defined. Namely these are natural transformations which commute with the strengths. This makes bimodular categories and the profunctors between them into a bicategory.

As noted in the main text, the strength of a bimodular profunctor could alternatively be considered as a horizontal action of a monoidal category on the profunctor. In fact such strengths are equivalent to the data of monoid actions for the lax monoidal functors $\mathbb{M}(-,-)$ and $\mathbb{N}(-,-)$, viewed as monoid objects in their respective functor categories. This gives a route for defining the horizontal composite of bimodular profunctors, just as we do for bimodular categories, by quotienting out the common action.

$$\begin{tikzcd} \mathbb{M} \arrow[r, bend left, "\mathbb{C}"{name=A}] \arrow[r, bend right, "\mathbb{C}'"'{name=B}] \arrow[from=A, to=B, Rightarrow, "T"] & \mathbb{N} \arrow[r, bend left, "\mathbb{D}"{name=C}] \arrow[r, bend right, "\mathbb{D}'"'{name=D}] \arrow[from=C, to=D, Rightarrow, "U"] & \mathbb{O} \arrow[r, phantom, "\mapsto"] & \mathbb{M} \arrow[r, bend left, "\mathbb{C}\otimes_{\mathbb{N}}\mathbb{D}"{name=E}] \arrow[r, bend right, "\mathbb{C}'\otimes_{\mathbb{N}}\mathbb{D}'"'{name=F}] \arrow[from=E, to=F, Rightarrow, "T\otimes_{\mathbb{N}}U"] & \mathbb{O} \end{tikzcd}$$

As in any category of modules we would like to define the composite by quotienting out the action of the common scalar, taking a coequaliser of the two action maps.

$$\begin{tikzcd} P(C,C')\times Q(D,D') \arrow[d, shift right, "\text{left strength}"'] \arrow[d, shift left, "\text{right strength}"] \\ \int^M P(C<M,C'<M)\times Q(D,D') + P(C,C')\times Q(M>D,M>D') \arrow[d, dashed] \\ P\otimes Q(C,C',D,D') \end{tikzcd}$$

The form of this diagram is slightly more complicated than the one we saw for bimodular categories. In the form we have seen them, the strengths of a Tambara module have different codomains for the two action maps. So to match the form of a coequaliser we embed both into their coproduct. This is alternatively the pushout of the two action maps, but we choose to present it as is to reinforce that this is just another instance of bimodule composition.

B.1 String Diagrams for Bimodular Categories

Proposition B.3 (From Proposition 2.4)**.** *There exists a forgetful functor U :* **SBimod** $\to$ **Bgraph***.*

Proof. Given $(\mathbb{A},\mathbb{M},\triangleright,\mathbb{N},\triangleleft)$, we define the following bimodular graph.

$$\begin{aligned}
&\mathscr{A}(M_0,...,M_m,A,N_0,...,N_n;P_0,...,P_p,B,Q_0,...,Q_q) = \\
&\qquad \mathbb{A}(M_0\triangleright...\triangleright M_m\triangleright A_n\triangleleft N_0\triangleleft...\triangleleft N_n;N_0\triangleright...\triangleright N_n\triangleright B\triangleleft Q_0\triangleleft...\triangleleft Q_q),\\
&\mathscr{M}(M_0,...,M_m;P_0,...,P_p) = \mathbb{M}(M_0\otimes...\otimes M_m;P_0\otimes...\otimes P_p),\\
&\mathscr{N}(N_0,...,N_n;Q_0,...,Q_q) = \mathbb{N}(N_0\otimes...\otimes N_n;Q_0\otimes...\otimes Q_q).
\end{aligned}$$

We now check that this assignment is functorial: indeed, the functors (F,H,K) induce three functions between the edge sets of the bimodular graph. □

Lemma B.4. *Collages induce a functor* Coll: **sBimod** $\to$ $\mathbf{2Cat}_2$ *from strict bimodular categories to bipointed 2-categories. Picking the hom-category between the chosen 0-cells of a bipointed 2-category induces a functor from bipointed 2-categories to strict bimodular categories* Chosen: $\mathbf{2Cat}_2$ $\to$ **sBimod***.*

Proof. See Appendix, Lemmas B.5 and B.6. □

Lemma B.5 (From Lemma B.4)**.** *Collages induce a functor* $\mathsf{Coll}\colon \mathbf{sBimod} \to \mathbf{2Cat}_2$ *from strict bimodular categories to bipointed 2-categories.*

Proof. On objects, the functor is defined by the collage with its two 0-cells, $\mathsf{Coll}(\mathbb{C},\mathbb{M},\mathbb{N}) = (\mathsf{Coll}_{\mathbb{C}}, M, N)$. Given a morphism of bimodular categories, $(F,H,K)\colon (\mathbb{C},\mathbb{M},\mathbb{N}) \to (\mathbb{D},\mathbb{P},\mathbb{Q})$, the functor takes it to the strict 2-functor defined by: sending $\mathsf{Coll}_{\mathbb{C}}(M,M) = \mathbb{M}$ to $\mathsf{Coll}_{\mathbb{D}}(P,P) = \mathbb{P}$ with H; sending $\mathsf{Coll}_{\mathbb{C}}(N,N) = \mathbb{N}$ to $\mathsf{Coll}_{\mathbb{D}}(Q,Q) = \mathbb{Q}$ with K; sending $\mathsf{Coll}_{\mathbb{C}}(M,N) = \mathbb{C}$ to $\mathsf{Coll}_{\mathbb{D}}(P,Q) = \mathbb{D}$ with F; and finally noticing that both $\mathsf{Coll}_{\mathbb{C}}(N,M)$ and $\mathsf{Coll}_{\mathbb{D}}(Q,P)$ are empty. This assignment defines a 2-functor preserving composition: this is thanks to the fact that composition has been defined in $\mathsf{Coll}_{\mathbb{C}}$ and $\mathsf{Coll}_{\mathbb{D}}$ using the strict monoidal actions, and the functors F, H and K do preserve the monoidal actions. □

Lemma B.6 (From Lemma B.4)**.** *Picking the hom-category between the chosen 0-cells of a bipointed 2-category induces a functor from bipointed 2-categories to strict bimodular categories* $\mathsf{Chosen}\colon \mathbf{2Cat}_2 \to \mathbf{sBimod}$.

Proof. On objects, the functor is defined by taking a bipointed bicategory the hom-category between the selected points, $\mathsf{Chosen}(\mathbb{A},M,N) = \mathbb{A}(M,N)$. This hom-category is a bimodular category acted on by the hom-categories $\mathbb{A}(M,M)$ and $\mathbb{A}(N,N)$; both of these are monoidal categories (bicategories with a single object) with the tensor defined by composition in the bicategory. The actions are also defined by pre and post-composition in the bicategory.

Consider now a strict 2-functor $S\colon (\mathbb{A},M,N) \to (\mathbb{B},P,Q)$ that sends $S(M) = P$ and $S(N) = Q$. It must induce strict monoidal functors $H = S(M,M)\colon \mathbb{A}(M,M) \to \mathbb{B}(P,P)$ and $K = S(N,N)\colon \mathbb{A}(N,N) \to \mathbb{B}(Q,Q)$ and a functor $F = S(M,N)\colon \mathbb{A}(M,N) \to \mathbb{B}(P,Q)$. All these functors must preserve composition in the original bicategory, so the triple $\mathsf{Chosen}(S) = (F,H,K)$ is a morphism of strict bimodular categories. □

Theorem B.7 (From Theorem 2.7)**.** *There exists an adjunction between strict bimodular categories and bipointed 2-categories given by the collage,* $\mathsf{Coll}_{\mathbb{C}}\colon \mathbf{sBimod} \to \mathbf{2Cat}_2$*, and picking the hom-category between the chosen 0-cells,* $\mathsf{Chosen}\colon \mathbf{2Cat}_2 \to \mathbf{sBimod}$*. Moreover, the unit of this adjunction is a natural isomorphism.*

Proof. We have already proven that both sides of the adjunction are indeed functors in Lemma B.4. Let us show that $\mathsf{Coll}_{\mathbb{C}}$ is the free bipointed 2-category on a bimodular category $\mathbb{C}$. We start by noting that there exists a homomorphism of bimodular categories

$$\mathscr{I}\colon (\mathbb{C},\mathbb{M},\mathbb{N}) \to (\mathsf{Coll}_{\mathbb{C}}(M,N), \mathsf{Coll}_{\mathbb{C}}(M,M), \mathsf{Coll}_{\mathbb{C}}(N,N)),$$

by construction of the collage; this determines the natural isomorphism of the unit of the adjunction we are constructing. Consider now a bipointed 2-category $(\mathbb{A},P,Q)$ and a homomorphism of bimodular categories

$$(F,H,K)\colon (\mathbb{C},\mathbb{M},\mathbb{N}) \to (\mathbb{A}(P,Q),\mathbb{A}(P,P),\mathbb{A}(Q,Q));$$

we will now prove that there exists a unique 2-functor $\mathscr{F}\colon \mathsf{Coll}_{\mathbb{C}} \to \mathbb{A}$ such that $\mathscr{I} \mathbin{⨟} \mathscr{F} = (F,H,K)$. Because the 2-functor is bipointed, we know that $\mathscr{F}(M) = P$ and that $\mathscr{F}(N) = Q$, so it is determined on 0-cells. We know that its component on $\mathsf{Coll}_{\mathbb{C}}(M,N)$, $\mathsf{Coll}_{\mathbb{C}}(M,M)$ and $\mathsf{Coll}_{\mathbb{C}}(N,N)$ must be given by F, H and K; while its only possible component on the empty category $\mathsf{Coll}_{\mathbb{C}}(N,M)$ is trivial; this determines it on 1-cells, but also on 2-cells, because F, H, and K are a functor and a pair of monoidal functors. □

Theorem B.8 (From Theorem 2.8)**.** *There exists an adjunction between bimodular graphs and strict bimodular categories. The left side of this adjunction is given by finding the bimodular category whose collage is the free 2-category on the bimodular graph,* $\mathsf{bmStr} \colon \mathbf{bmGraph} \to \mathbf{sBimod}$*. The right side of the adjunction is the previously mentioned forgetful functor* $\mathsf{U} \colon \mathbf{sBimod} \to \mathbf{bmGraph}$.

Proof. String diagrams for bicategories are based on an adjunction between 2-graphs and 2-categories in Theorem A.4, whose left adjoint is $\mathbf{Str} \colon \mathbf{2Graph} \to \mathbf{2Cat}$. By Proposition A.2, this induces an adjunction between bipointed 2-graphs and bipointed 2-categories, $\mathbf{Str}_2 \colon \mathbf{2Graph}_2 \to \mathbf{2Cat}_2$. We can compose this adjunction with the adjunction between bimodular graphs and bipointed 2-graphs; to obtain a left adjoint $\mathsf{bStr}_2 \colon \mathbf{bmGraph} \to \mathbf{2Cat}_2$.

Finally, we employ the adjunction given by collages from strict bimodular categories to bipointed 2-categories in Theorem 2.7. This adjunction has an invertible unit, and thus, by a general principle (Proposition A.1), it induces an adjunction with left adjoint $\mathsf{bmStr} \colon \mathbf{bmGraph} \to \mathbf{sBimod}$. □

Remark B.9 (Shared state). The equations in Figure 14 present the theory of shared state over the string diagrams of bimodular categories. Semantics can be given in two different theories of processes sharing the same state. For instance, the first category can allow for probabilistic processes, $\mathbb{A} = \mathbf{Stoch}$, while the second can be deterministic, $\mathbb{B} = \mathbf{Set}$. The bimodular category determined by the promonad $\mathbf{StochState}_S(A,B) = \mathbf{Set}(S \times A, \mathsf{Stoch}(S \times B))$ can give semantics to both and has suitable monoidal actions: the actions to the common wire become the get and put functions of the state promonad.

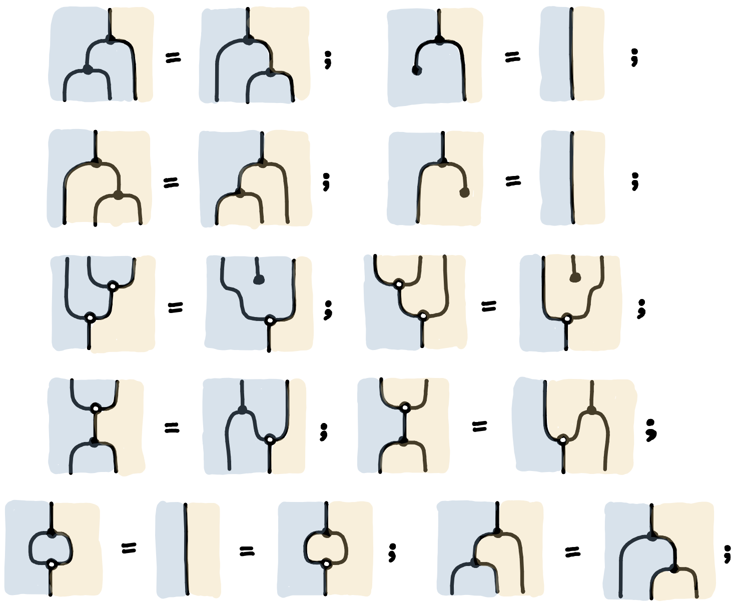


Figure 14: Theory of shared state.

B.2 String Diagrams of Premonoidal Categories

Definition B.10. A *premonoidal category* is a category $\mathbb{C}$ endowed with an object $I \in \mathbb{C}$ and an object $A \otimes B \in \mathbb{C}$ for each $A, B \in \mathbb{C}_{obj}$; and two functors $(A \otimes \bullet) \colon \mathbb{C} \to \mathbb{C}$ and $(\bullet \otimes B) \colon \mathbb{C} \to \mathbb{C}$ that coincide on $(A \otimes B)$, even if $(\bullet \otimes \bullet)$ is not itself a functor. Finally, it is endowed with the following cohernece isomorphisms, $\alpha_{A,B,C} \colon A \otimes (B \otimes C) \to (A \otimes B) \otimes C$, $\lambda_A \colon A \otimes I \to A$ and $\rho_A \colon I \otimes A \to A$, which interchange with any other morphism, are natural at each given component and satisfy the pentagon and triangle equations.

Definition B.11. An *effectful category* is an identity-on-objects functor, $\mathbb{V} \to \mathbb{C}$, from a monoidal category $\mathbb{V}$ to a premonoidal category $\mathbb{C}$ that strictly preserves all of the premonoidal structure and whose image is central.

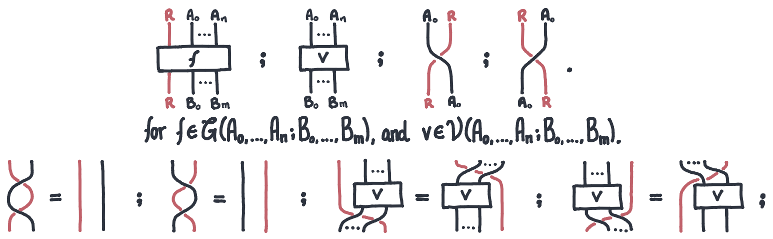

R
A₀
Aₙ
f
R
B₀
Bₘ
A₀
Aₙ
v
B₀
Bₘ
A₀
R
R
A₀
R
A₀
A₀
R
for f ∈ G(A₀,...,Aₙ;B₀,...,Bₘ), and v ∈ V(A₀,...,Aₙ;B₀,...,Bₘ).
v
v
v
v

Figure 15: Syntax for the string diagrams of premonoidal and effectful categories (Román, 2020 [46]).

Proposition B.12. *Effectful categories $\mathbb{V} \to \mathbb{C}$ are equivalent to $(\mathbb{V}, \mathbb{V})$-bimodular categories such that there exists an identity on objects functor that preserves the monoidal actions [30].*

In this sense, the string diagrams of premonoidal categories and effectful categories are particular cases of the string diagrams for bimodular categories. The extra wire that appears in the string diagrams of an effectful category $\mathbb{V} \to \mathbb{C}$ is precisely the bimodular category $\mathbb{C}$, with its two actions on $\mathbb{V}$. Morphisms in $\mathbb{C}$ need this wire as an input and as an output, while morphisms in $\mathbb{V}$ do not. A detailed discussion of the string diagrams of premonoidal categories was presented to the last Applied Category Conference by this second author [46].

C Functor Boxes

Proposition C.1. *There exists a forgetful functor from the lax functors category to the category of functor box signatures,* $\mathsf{Ulax} \colon \mathbf{Lax} \to \mathbf{Fbox}$.

Proof. Any lax monoidal functor induces a functor box signature $\mathsf{Ulax}(\mathbb{A}, \mathbb{X}, F, \varepsilon, \mu) = (\mathscr{A}, \mathscr{X}, \mathscr{F}_\bullet, \mathscr{F}^\bullet)$ defined by $\mathscr{A}_{obj} = \mathbb{A}_{obj}$, by $\mathscr{X}_{obj} = \mathbb{X}_{obj}$ and taking edges to be morphisms,

- $\mathscr{A}(A_0, ..., A_n; B_0, ..., B_m) = \mathbb{A}(A_0 \otimes ... \otimes A_n; B_0 \otimes ... \otimes B_m)$,
- $\mathscr{X}(X_0, ..., X_n; Y_0, ..., Y_m) = \mathbb{X}(X_0 \otimes ... \otimes X_n; Y_0 \otimes ... \otimes Y_m)$,
- $\mathscr{F}_\bullet(A_0, ..., A_n; Y_0, ..., Y_m) = \mathbb{A}(A_0 \otimes ... \otimes A_n; F(Y_0 \otimes ... \otimes Y_m))$,
- $\mathscr{F}^\bullet(X_0, ..., X_n; B_0, ..., B_m) = \mathbb{A}(F(X_0 \otimes ... \otimes X_n); B_0 \otimes ... \otimes B_m)$.

Consider now a homomorphism of lax monoidal functors $(H,K)\colon (\mathbb{A},\mathbb{X},F) \to (\mathbb{B},\mathbb{Y},G)$. The pair of strict monoidal functors H and K extend to all the sets of edges. For instance, because of the condition $F \mathbin{⨾} K = H \mathbin{⨾} G$, the functor K induces a map

$$\mathbb{A}(A_0 \otimes \ldots \otimes A_n; F(Y_0 \otimes \ldots \otimes Y_m)) \to \mathbb{B}(K(A_0) \otimes \ldots \otimes K(A_n); G(H(Y_0) \otimes \ldots \otimes H(Y_m))),$$

and the rest of the maps are analogous. This assignment preserves the composition and identities of the lax monoidal functors category, which are precisely compositions and identities of functors. □

Lemma C.2. *The syntactic bicategory of a functor box signature $(\mathscr{A},\mathscr{X},\mathscr{F})$ induces a lax monoidal functor $S_{\mathscr{A},\mathscr{X}}\colon \mathbf{F}_{mon}(\mathscr{X}) \to \mathbb{S}(\mathscr{A},\mathscr{A})$ from the free monoidal category on $\mathscr{X}$ to the monoidal category of the endocells in $\mathscr{A}$. This assignment, S:* **Fbox** $\to$ **Lax***, is functorial.*

Proof. We begin by defining the assignment explicitly. We first consider $\mathbf{F}_{mon}(\mathscr{X})$, the free strict monoidal category on the polygraph $\mathscr{X}$. We then consider $\mathbb{S}(\mathscr{A},\mathscr{A})$, the strict monoidal category formed by the endocells of the syntactic 2-category on $\mathscr{A}$. A functor $F\colon \mathbf{F}_{mon}(\mathscr{X}) \to \mathbb{S}(\mathscr{A},\mathscr{A})$ is defined on objects by the composition $F(X_0) = F^{\uparrow} \mathbin{⨾} X_0 \mathbin{⨾} F_{\downarrow}$, and similarly on morphisms. It becomes a lax monoidal functor with thanks to the unit and counit maps provided by the adjunction $F^{\uparrow} \dashv F_{\downarrow}$.

We now prove that this assignment is functorial. Consider a functor box signature map determined by $(h,k)\colon (\mathscr{A},\mathscr{X}) \to (\mathscr{B},\mathscr{Y})$, inducing lax monoidal functors $F\colon \mathscr{F}_{mon}(\mathscr{X}) \to \mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A})$ and $G\colon \mathbf{F}_{mon}(\mathscr{Y}) \to \mathbb{S}_{\mathscr{B},\mathscr{Y}}(\mathscr{B},\mathscr{B})$. Because of the adjunction determining free strict monoidal categories, the map $h\colon \mathscr{X} \to \mathscr{Y}$ determines a strict monoidal functor $H\colon \mathbf{F}_{mon}(\mathscr{X}) \to \mathbf{F}_{mon}(\mathscr{Y})$. Now, because the syntactic bicategory of a functor box is also freely generated, we can describe a map of 2-categories $\mathbb{S}_{\mathscr{A},\mathscr{X}} \to \mathbb{S}_{\mathscr{B},\mathscr{Y}}$ induced by the functions h,k and sending the pieces determining the adjunction on one side to the adjunction on the other side. This 2-categorical functor restricts to a strict monoidal functor $K\colon \mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A}) \to \mathbb{S}_{\mathscr{B},\mathscr{Y}}(\mathscr{B},\mathscr{B})$.

Finally, by construction and because the 2-categorical functor sends $F^{\uparrow} \dashv F_{\downarrow}$ to $G^{\uparrow} \dashv G_{\downarrow}$, we have that the H and K here defined satisfy $F \mathbin{⨾} K = H \mathbin{⨾} G$ and preserve the structure maps of the lax monoidal functor. □

Theorem C.3 (From Theorem 3.4). *There exists an adjunction between the category of functor box signatures,* **Fbox***, and the category of pairs of strict monoidal categories with a lax monoidal functor between them,* **Lax***. The free side of this adjunction is given by the syntax of Figure 7.*

Proof. Given a functor box signature $(\mathscr{A},\mathscr{X})$ we will prove that the lax monoidal functor induced by its syntactic bicategory, $F\colon \mathscr{F}_{mon}(\mathscr{X}) \to \mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A})$, is the free one.

Consider a lax monoidal functor $G\colon \mathbb{B} \to \mathbb{Y}$ endowed with a box signature morphism $(\mathscr{A},\mathscr{X}) \to (\mathbb{B},\mathbb{Y})$. Already by the universal property of the free strict monoidal category, we know that there exists a unique strict monoidal functor $H\colon \mathscr{F}_{mon}(\mathscr{X}) \to \mathscr{F}_{mon}(\mathscr{Y})$ that, under the forgetful functor, commutes with the box signature morphism.

We need to show that there exists a unique strict monoidal functor $K\colon \mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A}) \to \mathbb{S}_{\mathscr{B},\mathscr{Y}}(\mathscr{B},\mathscr{B})$ that commutes with H and with the box signature morphism. We first define it on 1-cells. By structural induction, a 1-cell of $\mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A})$ is: *(i)* an object of $\mathscr{A}$ followed by a 1-cell; or *(ii)* a functor box opening $F^{\uparrow}$, a 1-cell of $\mathscr{F}_{mon}(\mathscr{X})$ and a functor box closing $F_{\downarrow}$, followed by a 1-cell of $\mathbb{S}_{\mathscr{A},\mathscr{X}}(\mathscr{A},\mathscr{A})$. In the first case, the object in $\mathscr{A}$ must be sent to the object determined by the box signature morphism; in the second case, because the condition $F \mathbin{⨾} K = H \mathbin{⨾} G$ must be satisfied, we must send the object $F(X_0)$ to $G(H(X_0))$.

We now define it on 2-cells. The plain edges need to be mapped according to the functor box signature homomorphism; the functor box edges are already mapped according to H; the in-box and out-box edges are mapped according to the functor box signature homomorphism. The unit of the adjunction must be preserved because it is a structure map of the lax monoidal functor. Finally, the unit of the adjunction must always appear enclosed in between the cells $F^{\uparrow}$ and $F_{\downarrow}$, which means it always represents the μ_F structure map of the lax monoidal functor and must be mapped accordingly to μ_G. □

D Pointed Bimodular Profunctors

D.1 The Point of Coend Calculus

Coend calculus is the name given to the algebraic manipulations of coends that prove isomorphisms or construct natural transformations between profunctors. In the same way that regular logic links relations, a coend calculus expression is a list of profunctors linked by some objects that are bound to a coend. Usually, the isomorphisms that we construct are never made explicit, and it is difficult for the reader to compute the precise map we constructed.

Fortunately, this has a straightforward solution. We propose to *point* the coends: to write an expression together with the *generic element* it computes. An expression of pointed coend calculus is a coend bounding some objects and a series of *pointed profunctors*. For instance,

$$\int^{M,N} f \in P(A;M,N) \times g \in Q(M;B) \times h \in \mathbb{C}(N;C), \text{ instead of } \int^{M,N} P(A;M,N) \times Q(M;B) \times \mathbb{C}(N;C).$$

The coend quotients expressions by dinaturality, meaning that any action on the left of a coend can be also written as an action on the right. In terms of pointed profunctors, this means that

$$\int^{N} (f < h) \in P(A;N) \times g \in Q(N;B) = \int^{M} f \in P(A;M) \times (h > g) \in Q(M;B).$$

Proposition D.1. *Let $\mathbb{C}$ be a category and let $F\colon \mathbb{C}^{op} \to$ **Set** and $G\colon \mathbb{C} \to$ **Set** be a presheaf and a copresheaf, respectively. The following are natural isomorphisms of pointed profunctors,*

$$\int^{X} f \in \mathbb{C}(X;A) \times h \in F(A) \;\cong\; (f > h) \in F(X); \qquad \int^{X} f \in \mathbb{C}(A;X) \times h \in G(A) \;\cong\; (h < f) \in G(X).$$

We call these isomorphisms the "pointed" Yoneda reductions.

Remark D.2. Using pointed coends, any derivation does also include the computation of the isomorphism it induces. As an example, compare the following with the usual coend derivation of a cartesian lens [13],

Proposition D.3. *In a cartesian monoidal category, the pairs of morphisms $f \in \mathbb{C}(A;M \times X)$ and $g \in \mathbb{C}(M \times Y;B)$, quotiented by dinaturality, are in bijective correspondence with the pairs of morphisms $\mathbb{C}(A;M)$ and $\mathbb{C}(M \times Y;B)$.*

$$\int^{M} f \in \mathbb{C}(A;M \times X) \times g \in \mathbb{C}(M \times Y;B) \qquad \cong (\textit{ by the adjunction } \Delta \dashv \times)$$

$$\int^{M} (f \mathbin{⨾} \pi_1) \in \mathbb{C}(A;M) \times (f \mathbin{⨾} \pi_2) \in \mathbb{C}(A;M) \times g \in \mathbb{C}(M \times Y;B) \qquad \cong (\textit{ by pointed Yoneda lemma })$$

$$(f \mathbin{⨾} \pi_2) \in \mathbb{C}(A;M) \times ((f \mathbin{⨾} \pi_1) \otimes id) \mathbin{⨾} g \in \mathbb{C}(M \times Y;B).$$

In the first step, we have used that the adjunction $(\Delta \dashv \times)$ is given by postcomposition with projections and; in the second step, we use that the action on the last profunctor is defined as $h > g = (h \otimes id) \mathbin{⨾} g$. The bijection has been explicitly constructed as sending the pair $(f;g)$ to $(f \mathbin{⨾} \pi_2; ((f \mathbin{⨾} \pi_1) \otimes id) \mathbin{⨾} g)$.

D.2 Semantics of Functor Boxes

Proposition D.4 (Bimodular categories of a lax monoidal functor, from Proposition 4.9)**.** *Let $\mathbb{X}$ and $\mathbb{A}$ be two monoidal categories and let $F\colon \mathbb{X} \to \mathbb{A}$ be a monoidal functor between them, endowed with natural transformations $\psi_0\colon J \to FI$ and $\psi_2\colon FX \otimes FY \to F(X \otimes Y)$. The following profunctors, $\mathbb{A} \rtimes_F \mathbb{X}\colon \mathbb{A} \times \mathbb{X} \to \mathbb{A} \times \mathbb{X}$ and $\mathbb{X} \ltimes_F \mathbb{A}\colon \mathbb{X} \times \mathbb{A} \to \mathbb{X} \times \mathbb{A}$ determine two promonads, and therefore two Kleisli categories.*

$$\mathbb{A} \rtimes_F \mathbb{X}(A,X;B,Y) = \int^{M \in \mathbb{X}} \mathbb{A}(A;B \otimes FM) \times \mathbb{X}(M \otimes X;Y);$$
$$\mathbb{X} \ltimes_F \mathbb{A}(X,A;Y,B) = \int^{M \in \mathbb{X}} \mathbb{A}(A;FM \otimes B) \times \mathbb{X}(M \otimes A;B);$$

These two Kleisli categories are $(\mathbb{A},\mathbb{X})$ and $(\mathbb{X},\mathbb{A})$-bimodular, respectively.

Proof. We prove that $\mathbb{A} \rtimes_F \mathbb{X}$ define a promonad and, in particular, the hom-sets of a category. We write the elements $\mathbb{A} \rtimes_F \mathbb{X}(A,X;B,Y)$ are given by pairs (f,α) where $f\colon A \to B \otimes FM$ and $\alpha\colon M \otimes X \to Y$ for some $M \in \mathbb{X}_{obj}$.

The unit of the promonad sends a pair of morphisms $u\colon A \to B$ and $r\colon X \to Y$ to the morphism $(u \otimes \psi_0, r)$, where $u \otimes \psi_0\colon A \to B \otimes FI$ and, modulo coherence, $r\colon I \otimes X \to Y$. The multiplication of the promonad sends $(f,\alpha)\colon (A,X) \to (Y,B)$ and $(g,\beta)\colon (Y,B) \to (Z,C)$, to the composite formed by $f \mathbin{⨾} (g \otimes \mathrm{id}) \mathbin{⨾} (\mathrm{id} \otimes \psi_2)\colon A \to Z \otimes F(M \otimes N)$ and $(\mathrm{id} \otimes \alpha) \mathbin{⨾} \beta\colon N \otimes M \otimes A \to B$. Finally, from the axioms of lax monoidal functors, it follows that this composition is associative and unital. □

E Internal Diagrams

Theorem E.1 (From Theorem 5.3)**.** *There is a 3-functor from the syntactic tricategory of internal diagrams into pointed bimodular profunctors for any interpretation of the polygraph into a monoidal category.*

Proof. The syntactic tricategory of internal diagrams has been constructed as a free tricategory, so it suffices to determine where the generators are sent. For this, we follow Figure 16. Let the square brackets, $[\![\bullet]\!]$, denote the interpretation of the polygraph into a monoidal category.

The region $\mathscr{G}$ is sent to the monoidal category $[\![\mathscr{G}]\!] = (\mathbb{A},\otimes,I)$, while the region $\mathscr{I}$ is sent to the terminal monoidal category. The generator $L_\bullet$ is sent to the pointed bimodular category $({}_{\mathbb{1}}\mathbb{A}_{\mathbb{A}}, I)$, while the generator $R_\bullet$ is sent to the bimodular category $({}_{\mathbb{A}}\mathbb{A}_{\mathbb{1}}, I)$. The generator A is sent to the pointed bimodular category $({}_{\mathbb{A}}\mathbb{A}_{\mathbb{A}}, [\![A]\!])$.

Let us consider the 2-cells. The 2-cells n_1 and e_2 are sent to the profunctors $\mathbb{A}(\bullet \otimes \bullet;\bullet)$ and $\mathbb{A}(\bullet;\bullet \otimes \bullet)$, pointed in the identities of the monoidal unit. The 2-cells n_2 and e_1 are sent to the profunctors $\mathbb{A}(I;\bullet)$ and $\mathbb{A}(\bullet;I)$, pointed in the identities of the monoidal unit.

The 2-cells $A^{\pitchfork}$ and $A_{\pitchfork}$ are sent to the representable and corepresentable profunctors of the object $[\![A]\!]$, which are pointed in id_A, the identity on that object. Finally, any 2-cell arising from an edge $f \in \mathscr{G}(A_0,...,A_n;B_0,...,B_m)$ is sent to the hom-profunctor, pointed in the relevant morphism,

$$[\![f]\!] \in \mathbb{A}([\![A_0]\!] \otimes ... \otimes [\![A_n]\!]; [\![B_0]\!] \otimes ... \otimes [\![B_m]\!]).$$

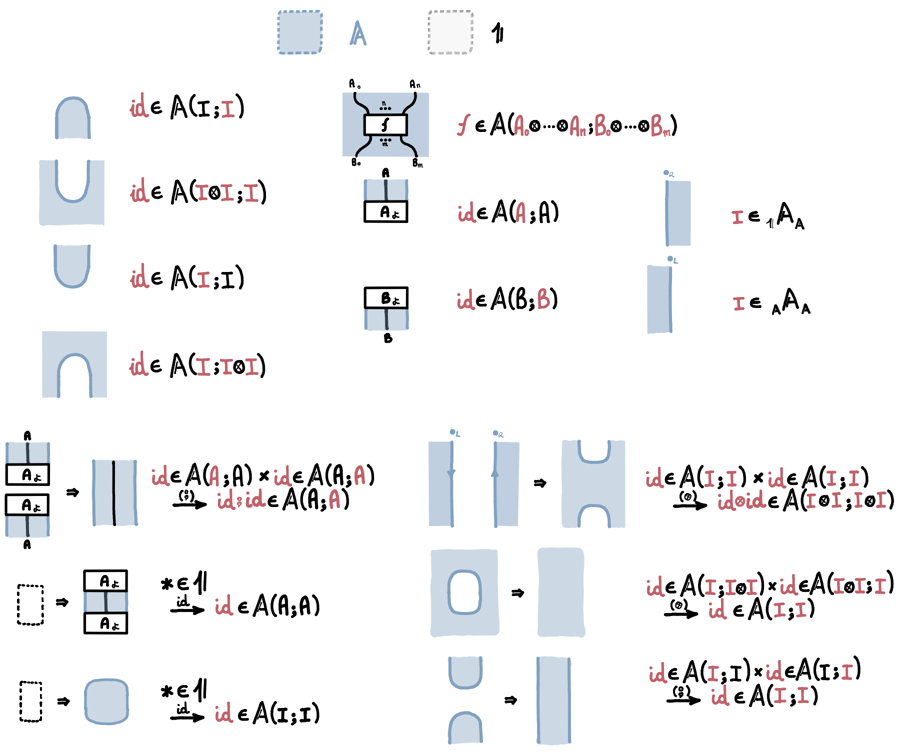

A
1
id ∈ A(I;I)
f ∈ A(A₀⊗⋯⊗Aₙ;B₀⊗⋯⊗Bₘ)
id ∈ A(I⊗I;I)
id ∈ A(A;A)
I ∈ ₁A_A
id ∈ A(I;I)
id ∈ A(B;B)
I ∈ _AA_A
id ∈ A(I;I⊗I)
id ∈ A(A;A) × id ∈ A(A;A) —(⨾)→ id⨾id ∈ A(A;A)
id ∈ A(I;I) × id ∈ A(I;I) —(⊗)→ id⊗id ∈ A(I⊗I;I⊗I)
∗ ∈ 1 —id→ id ∈ A(A;A)
id ∈ A(I;I⊗I) × id ∈ A(I⊗I;I) —(⨾)→ id ∈ A(I;I)
∗ ∈ 1 —id→ id ∈ A(I;I)
id ∈ A(I;I) × id ∈ A(I;I) —(⨾)→ id ∈ A(I;I)

Figure 16: Semantics for open internal diagrams in terms of pointed bimodular profunctors.

It is well-known that every representable profunctor is adjoint to its corepresentable profunctor, which gives semantics to the syntactic adjunctions. The only adjunctions missing are that between ${}_{\mathbb{A}}\mathbb{A}_{\mathbb{1}}$ and ${}_{\mathbb{1}}\mathbb{A}_{\mathbb{A}}$; for these, we must note that there is a Yoneda isomorphism of the following form,

$$\int^{M} \mathbb{A}(I;M) \times \mathbb{A}(M \otimes X;Y) \cong \mathbb{A}(X;Y), \qquad \int^{M} \mathbb{A}(X;Y \otimes M) \times \mathbb{A}(M;I) \cong \mathbb{A}(X;Y),$$

and analogous ones swapping the position of M in the tensor product. □

F The Collage of Bimodular Profunctors

To define collages in greater generality we use the notion of a *bimodular pasting diagram*: a composable arrangement of bimodular categories and profunctors. For a precise definition of 2-dimensional pasting diagrams, see the reference book by Johnson and Yau [28, Chapter 3].

Definition F.1 (Collages of Bimodular Profunctors)**.** Let $\mathscr{D}$ be a bimodular pasting diagram. We define a bicategory $\mathsf{Coll}(\mathscr{D})$ as follows:

- there is an object M for each vertex in $\mathscr{D}$, labelled by a monoidal category $\mathbb{M}$;

- endomorphism categories $\mathsf{Coll}(\mathscr{D})(M,M)$ are given by $\mathbb{M}$ with its monoidal structure;
- for each 1-edge labelled by an $(\mathbb{M},\mathbb{N})$-bimodular category $\mathbb{C}$, and each object $C \in \mathbb{C}$, we have a 1-cell $(\mathbb{C},C) : M \to N$;
- for each 2-edge labelled by an $(\mathbb{M},\mathbb{N})$-bimodular profunctor $P : \mathbb{C} \to \mathbb{D}$, and each $p \in P(C,D)$ we have a 2-cell $(p,C,D) : (\mathbb{C},C) \to (\mathbb{D},D)$;
- compositions are given by monoidal actions, actions of morphisms on profunctors, or quotienting maps of tensor products, where relevant.

Example F.2. In the case where the pasting diagram $\mathscr{D}$ is a single edge labelled by a bimodular category $\mathbb{C}$, we recover our earlier notion of collage from Definition 2.5.

Example F.3. Each of the profunctors in Figure 8 have a collage whose 2-cells describe string diagrams for the section of functor box depicted. More complex composable arrangements of these profunctors can be assembled giving rise to bicategories modelling whole functors boxes or arrangements of them.

Remark F.4. The definition of a bimodular pasting diagram can be seen as that of a trifunctor from the free 2-category on a 2-graph to a tricategory of monoidal categories, bimodular categories, bimodular profunctors, and natural transformations. We conjecture that the collage described here realises a lax colimit of such a diagram, where the target category is enlarged to a tricategory of bicategories and "2-profunctors" [11, 32].

Conjecture F.5. *The collage of a diagram of bimodular profunctors is the lax 3-colimit of this diagram when viewed as a functor into a tricategory of 2-profunctors between 2-categories*

$$\mathsf{Coll}(\mathscr{D}) = \mathsf{Colim}_{\mathsf{lax}}\left(\mathbb{I} \xrightarrow{\mathscr{D}} \mathbb{BmProf} \hookrightarrow 2\mathbb{Prof} \right)$$

Moreover the tricategory of pointed bimodular profunctors is the universal collage, given by the identity diagram

$$\mathbb{BmProf}_{\mathsf{pt}} = \mathsf{Colim}_{\mathsf{lax}}\left(\mathbb{BmProf} = \mathbb{BmProf} \hookrightarrow 2\mathbb{Prof} \right).$$

This perspective unifies our construction with the typical notion of a *collage* of profunctors [52], which can be considered as a lax colimit of a functor into the bicategory of profunctors. Additionally, this elucidates the relationship between the various syntactic bicategories we have constructed, and tricategory of pointed bimodulars which we pronounced as a universe in which all such diagrams can live. Indeed, if $\mathbb{BmProf}_{\mathsf{pt}}$ is a colimit of the terminal diagram, then we should obtain inclusions of all collages into this tricategory. We leave the development of these notions for further work.

7. A Canonical Algebra of Open Transition Systems

Elena Di Lavore, Alessandro Gianola, Mario Román, Pawel Sobocinski, Nicoletta Sabadini
Applied Category Theory (ACT, 2022)

Abstract: Feedback and state are closely interrelated concepts. Categories with feedback, originally proposed by Katis, Sabadini and Walters, are a weakening of the notion of traced monoidal categories, with several pertinent applications in computer science. The construction of the free such categories has appeared in several different contexts, and can be considered as state bootstrapping. We show that a categorical algebra for open transition systems, Span(Graph)*, also due to Katis, Sabadini and Walters, is the free category with feedback over Span(Set). Intuitively, this algebra of transition systems is obtained by adding state to an algebra of predicates, and therefore Span(Graph)* is, in this sense, the canonical such algebra.
Declaration: *Hereby I declare that my contribution to this manuscript was to: provide the main theorem and its proof, provide the main idea, write most of the paper with help from my supervisor Pawel Sobocinski and Elena Di Lavore, some examples were provided by Nicoletta Sabadini and Alessandro Gianola. This manuscript is the conference version of "Span(Graph): a Canonical Feedback Algebra of Open Transition Systems".*

A canonical algebra of open transition systems [⋆]

Elena Di Lavore[1], Alessandro Gianola[2], Mario Román[1], Nicoletta Sabadini[3], and Paweł Sobociński[1]

[1] Tallinn University of Technology, Ehitajate tee 5, 12616 Tallinn, Estonia
[2] Free University of Bozen-Bolzano, Piazza Domenicani, 3, 39100 Bolzano BZ, Italy
[3] Università degli Studi dell'Insubria, Via Ravasi, 2, 21100 Varese VA, Italy

Abstract. Feedback and state are closely interrelated concepts. Categories with feedback, originally proposed by Katis, Sabadini and Walters, are a weakening of the notion of traced monoidal categories, with several pertinent applications in computer science. The construction of the *free* such categories has appeared in several different contexts, and can be considered as *state bootstrapping.* We show that a categorical algebra for *open transition systems*, $\mathbf{Span}(\mathbf{Graph})_*$, also due to Katis, Sabadini and Walters, is the free category with feedback over $\mathbf{Span}(\mathbf{Set})$. Intuitively, this algebra of transition systems is obtained by adding state to an algebra of predicates, and therefore $\mathbf{Span}(\mathbf{Graph})_*$ is, in this sense, the canonical such algebra.

Keywords: concurrency theory· category theory· transition systems · feedback · state · algebra.

1 Introduction

State from feedback. A remarkable fact from electronic circuit design is how data-storing components can be built out of a combination of *stateless components* and *feedback*. A famous example is the (set-reset) "NOR latch": a circuit with two stable configurations that stores one bit of information.

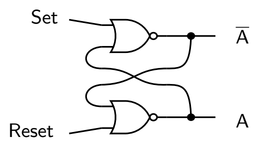


Fig. 1: NOR latch.

The NOR latch is controlled by two inputs, Set and Reset. Activating the first sets the output value to $\mathsf{A} = 1$; activating the second makes the output value return to $\mathsf{A} = 0$. This change is permanent: even when both Set and Reset are deactivated, the feedback loop maintains the last value the circuit was set to[4]—to wit, a bit of data has been conjured out

⋆ Elena Di Lavore, Mario Román and Paweł Sobociński were supported by the European Union through the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

[4] In its original description: *"the relay is designed to produce a large and* ***permanent*** *change in the current flowing in an electrical circuit by means of a small electrical stimulus received from the outside"* ([11], emphasis added).

of thin air. In this paper we show that this can be seen as an instance of a more abstract phenomenon: the universal way of adding feedback to a theory of processes consists of endowing each process with a *state space*.

Indeed, there is a natural weakening of the notion of traced monoidal categories called *categories with feedback* [28]. The construction of the *free* category with feedback coincides with a "state-bootstrapping" construction, $\mathsf{St}(\bullet)$, that appears in several different contexts in the literature [7,21,24]. We recall this construction and its mathematical status (Theorem 1), which can be summed up by the following intuition:

$$\text{Theory of Processes} + \text{Feedback} = \text{Theory of Stateful Processes.}$$

The algebra of transition systems. Our primary focus is the **Span**(**Graph**) model of concurrency, introduced in [25] as a categorical algebra of *communicating state machines*, or — equivalently — *open transition systems*. Open transition systems do not interact by input-output message passing, but by synchronization, producing a *simultaneous change of state*. This corresponds to a composition of spans, realized by taking a pullback in **Graph**. The dual algebra of **Cospan**(**Graph**) was introduced in [27]. It complements **Span**(**Graph**) by adding the operation of *communicating-sequential composition* [16].

Informally, a component of **Span**(**Graph**) is a state machine with states and transitions, i.e. a finite graph given by the 'head' of the span. The transition system is equipped with interfaces or *communication ports*, and every transition is labeled by the effect it produces in *all* its interfaces. We give examples below.

Stateful and stateless components. In Figure 2, we depict two open transition systems as arrows of **Span**(**Graph**). The first represents a NOR gate $\mathbb{B} \times \mathbb{B} \to \mathbb{B}$. The diagram below left is a graphical rendering of the corresponding span $\mathbb{B} \times \mathbb{B} \leftarrow N \to \mathbb{B}$, where $\mathbb{B}$ is considered as a single-vertex graph with two edges, corresponding to the signals $\{ 0, 1 \}$, N is the *unlabeled* graph depicted within the bubble, and the labels witness the action of two homomorphisms, respectively $N \to \mathbb{B} \times \mathbb{B}$ and $N \to \mathbb{B}$. Here each transition represents one of the valid input/output configurations of the gate. NOR gates are *stateless* components, since their transition graph N has a single vertex.

The second component is a span $L = \{\mathsf{Set}, \mathsf{Reset}, \mathsf{Idle}\} \to \{\mathsf{A}, \overline{\mathsf{A}}\} = R$ that models a set-reset latch. The diagram below right, again, is a convenient way of denoting the relevant span $L \leftarrow D \to R$. Latches store one bit of information, they are *stateful components*; consequently, their transition graph has two states.

In both cases, the boundaries on **Span**/**Cospan**(**Graph**) are stateless: indeed, they are determined by a mere set – the self-loops of a single-vertex graph. This is a restriction that occurs rather frequently: the important subcategory of **Span**(**Graph**), the one that we can clearly conceptually explain as *transition systems with interfaces*, is the full subcategory of **Span**(**Graph**) restricted

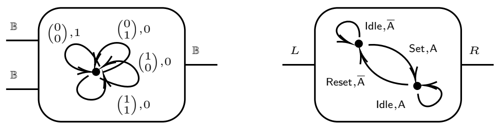


Fig. 2: A NOR gate and set-reset latch, in **Span**(**Graph**).

to objects that are single-vertex graphs, which we call **Span**(**Graph**)$_*$. Analogously, the relevant subcategory of **Cospan**(**Graph**) is **Cospan**(**Graph**)$_*$, the full subcategory on sets, or graphs with an empty set of edges.

Definition. **Span**(**Graph**)$_*$ is the full subcategory of **Span**(**Graph**) with objects the single-vertex graphs.

The problem with **Span**(**Graph**)$_*$ is that it is—at first sight—somewhat mysterious from the categorical point of view; the morphisms are graphs, but the boundaries are given by sets. *Decorated* and *structured* spans and cospans [13,3] were introduced as theoretical frameworks to capture such phenomena, which occur frequently when composing network structures. Nevertheless, they do not quite answer the question of *why* such examples do arise naturally.

Canonicity and our original contribution. Universal constructions, such as "state-bootstrapping" $\mathsf{St}(\bullet)$, characterize the object of interest up to equivalence, making it *the canonical object* satisfying some properties. This is the key to avoiding the problem outlined by Abramsky [1]: because of the lack of consensus about the intrinsic primitives of concurrency, we risk making our results too dependent on a specific syntax. It is thus important to characterize existing modeling formalisms for concurrent systems in terms of universal properties.

The main contribution of this paper is the characterization of **Span**(**Graph**)$_*$ in terms of a universal property: it is equivalent to the free category with feedback over the category of spans of functions. We now state this more formally:

Theorem. The free category with feedback over **Span**(**Set**) is equivalent to **Span**(**Graph**)$_*$, the full subcategory of **Span**(**Graph**) given by single-vertex graphs. That is, there is an equivalence of categories

$$\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*.$$

Given that **Span**(**Set**), the category of spans of functions, can be considered an *algebra of predicates* [4,10], the high level intuition that summarizes our main contribution (Theorem 2) can be given as:

$$\text{Algebra of Predicates} + \text{Feedback} = \text{Algebra of Transition Systems.}$$

We similarly prove (Section 3.4) that the free category with feedback over **Cospan**(**Set**) is equivalent to **Cospan**(**Graph**)$_*$, the full subcategory on discrete graphs of **Cospan**(**Graph**).

Related Work. $\mathbf{Span}/\mathbf{Cospan}(\mathbf{Graph})$ has been extensively used for the modeling of concurrent systems [25,27,39,40,9,36,16,14,15]. Similar approaches to compositional modeling of networks have used *decorated* and *structured cospans* [13,3]. Despite this, $\mathbf{Span}(\mathbf{Graph})_*$ has not previously been characterized in terms of a universal property.

In [28], the $\mathsf{St}(\bullet)$ construction (under a different name) is exhibited as the free *category with feedback*. Categories with feedback have been arguably underappreciated but, at the same time, the $\mathsf{St}(\bullet)$ construction has made multiple appearances as a "state bootstrapping" technique across the literature. The $\mathsf{St}(\bullet)$ construction is used to describe a string diagrammatic syntax for *concurrency theory* in [7]; a variant of it had been previously applied in the setting of *cartesian bicategories* in [24]; and it was again rediscovered to describe a *memoryful geometry of interaction* in [21]. However, a coherent account of both categories with feedback and their relation with these stateful extensions has not previously appeared. This motivates our extensive preliminaries in Sections 2.1 and 2.2.

Synopsis. Section 2 contains preliminary discussions on traced monoidal categories and categories with feedback; it explicitly describes $\mathsf{St}(\bullet)$, the free category with feedback. It collects mainly expository material. Section 3 exhibits a universal property for the $\mathbf{Span}(\mathbf{Graph})_*$ and $\mathbf{Cospan}(\mathbf{Graph})_*$ models of concurrency and Section 3.5 discusses a specific application.

Conventions. We write composition of morphisms in diagrammatic order, $(f\,;g)$. When describing morphisms in a symmetric monoidal category whose input and output are known, we omit the associators and unitors, implicitly using the coherence theorem for monoidal categories.

2 Preliminaries: categories with feedback

Categories with feedback were introduced in [28], and motivated by examples such as *Elgot automata* [12], *iteration theories* [6] and *continuous dynamical systems* [26]. We recall their definition below, contrast them with the stronger notion of *traced monoidal categories* in Section 2.2, discuss the relationship between feedback and delay in Section 2.3, recall the construction of a free category with feedback in Section 2.4 and conclude with some examples in Section 2.5.

2.1 Categories with feedback

A *feedback operator*, $\mathsf{fbk}(\bullet)$, takes a morphism $S \otimes A \to S \otimes B$ and *"feeds back"* one of its outputs to one of its inputs of the same type, yielding a morphism $A \to B$ (Figure 3, left). When using string diagrams, we depict the action of the feedback operator as a loop with a double arrowtip (Figure 3, right).

$$\frac{f\colon S\otimes A\to S\otimes B}{\mathsf{fbk}_S(f)\colon A\to B}$$

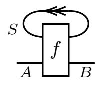


Fig. 3: Type and graphical notation for the operator $\mathsf{fbk}_S(\bullet)$.

Capturing a reasonable notion of feedback requires the operator to interact nicely with the flow imposed by the structure of a symmetric monoidal category. This interaction is expressed by a few straightforward axioms.

Definition 1. *A* category with feedback [28] *is a symmetric monoidal category* $\mathbf{C}$ *endowed with an operator*

$$\mathsf{fbk}_S\colon \mathbf{C}(S\otimes A, S\otimes B)\to\mathbf{C}(A,B),$$

which satisfies the following axioms (A1-A5, see also Figure 4*).*

(A1). Tightening, $u;\mathsf{fbk}_S(f);v=\mathsf{fbk}_S((\mathrm{id}\otimes u);f;(\mathrm{id}\otimes v))$.
(A2). Vanishing, $\mathsf{fbk}_I(f)=f$.
(A3). Joining, $\mathsf{fbk}_T(\mathsf{fbk}_S(f))=\mathsf{fbk}_{S\otimes T}(f)$.
(A4). Strength, $\mathsf{fbk}_S(f)\otimes g=\mathsf{fbk}_S(f\otimes g)$.
(A5). Sliding, $\mathsf{fbk}_T(f;(h\otimes\mathrm{id}))=\mathsf{fbk}_S((h\otimes\mathrm{id});f)$, *for* $h\colon S\to T$ *any isomorphism.*

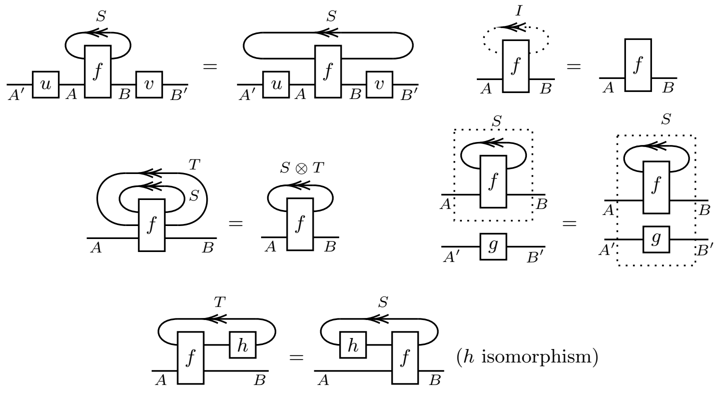


Fig. 4: Diagrammatic depiction of the axioms of feedback.

The natural notion of homomorphism between categories with feedback is that of a symmetric monoidal functor that moreover preserves the feedback structure. These are called *feedback functors*.

Definition 2. *A* feedback functor $F\colon\mathbf{C}\to\mathbf{D}$ *between two* categories with feedback $(\mathbf{C},\mathsf{fbk}^{\mathbf{C}})$ *and* $(\mathbf{D},\mathsf{fbk}^{\mathbf{D}})$ *is a* strong symmetric monoidal functor *such that*

$$F(\mathsf{fbk}^{\mathbf{C}}_S(f))=\mathsf{fbk}^{\mathbf{D}}_{F(S)}(\mu;Ff;\mu^{-1}),$$

where $\mu_{A,B} \colon F(A) \otimes F(B) \to F(A \otimes B)$ *is the structure morphism of the strong monoidal functor* F. *We call* $\mathsf{Feedback}$ *to the category of (small) categories with feedback and feedback functors between them. There exists a forgetful functor* $\mathcal{U} \colon \mathsf{Feedback} \to \mathsf{SymMon}$.

2.2 Traced monoidal categories

Categories with feedback are a weakening of the well known traced monoidal categories. Between them, there is an intermediate notion called *right traced category* [37] that strengthens the sliding axiom from isomorphisms to arbitrary morphisms. This first extension would be already too strong for our purposes later in Section 2.4: we would be unable to define a *state space* up to isomorphism. However, the more conceptual difference of traced monoidal categories is the "yanking axiom" (in Figure 5). Indeed, strengthening the sliding axiom and adding the yanking axiom yields the definition of traced monoidal category.

Traced monoidal categories are widely used in theoretical computer science. Since their conception [22] as an abstraction of the *trace* of a matrix in linear algebra, they have been used in linear logic and geometry of interaction [1,17,18], programming language semantics [19], automata theory [2] and fixed point operators [20,5].

Traces are thus undeniably important, but it is questionable whether we really want to always impose *all* of their axioms. Specifically, we will be concerned with the *yanking axiom* that states that $\mathsf{tr}(\sigma) = \mathrm{id}$. The yanking axiom is incontestably elegant from the geometrical point of view: strings are "pulled", and feedback (depicted as a loop with an arrowtip) disappears (Figure 5). However, if feedback can disappear without leaving any imprint, that must mean that it is *instantaneous*: its output necessarily mirrors its input.[5] Importantly for our purposes, this seems to imply that a feedback satisfying the yanking equation is "memoryless", or "stateless".

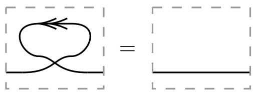


Fig. 5: The yanking axiom.

Consider again the NOR latch from Figure 1. We have seen how to model NOR gates in **Span**(**Graph**) in Figure 2, and the algebra of **Span**(**Graph**) does include a trace (Figure 6). However, imitating the real-world behavior of the NOR latch with *just* a trace is unsatisfactory: the trace of **Span**(**Graph**) is built out of stateless components, and tracing stateless components yields again a stateless component.

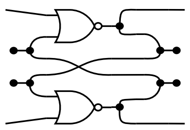


Fig. 6: Diagram for the NOR latch, modeled with a trace in **Span**(**Graph**).

In engineering and computer science, instantaneous feedback is actually a rare concept; a more common notion is that of *guarded feedback*. Consider *signal*

[5] In other words, traces are used to talk about processes in *equilibrium*, processes that have reached a *fixed point*. A theorem by Hasegawa [20] and Hyland [5] corroborates this interpretation: a trace in a cartesian category corresponds to a *fixpoint operator*.

flow graphs [38,32]: their categorical interpretation in [8] models feedback not by the usual trace, but by a trace "guarded by a register", that *delays the signal* and violates the yanking axiom (see Remark 7.8 in *loc.cit.*).

The component that trace misses in such examples is a *delay*.

2.3 Delay and feedback

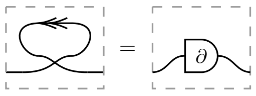


Fig. 7: Definition of *delay*.

The main difference between categories with feedback and traced monoidal categories is the failure of the yanking axiom. Consider the process that only "feeds back" its own input to itself then uses the "fed back" input to produce its output. We call this process, $\partial_A \coloneqq \mathsf{fbk}^{\mathbf{C}}_A(\sigma_{A,A})$, the *delay endomorphism*. The *yanking axiom* of traced monoidal categories states that the delay is equal to the identity, which is not necessarily true for *categories with feedback*. In that sense, a non-trivial delay is what sets apart categories with feedback from traced monoidal categories.

This interpretation of feedback as the combination of *trace* and *delay* can be made into a theorem when the category has enough structure. *Compact closed categories* are traced monoidal categories where every object A has a dual $A^\star$ and the trace is constructed from two pieces $\varepsilon\colon A \otimes A^\star \to I$ and $\eta\colon I \to A^\star \otimes A$. Even if not every traced monoidal category is compact closed, it is true that every traced monoidal category embeds fully faithfully into a compact closed category.[6] In a compact closed category, a feedback operator is necessarily a trace "guarded" by a *delay*.

Proposition 1 (Feedback from delay, [7]). *Let* $\mathbf{C}$ *be a compact closed category with* $\mathsf{fbk}^{\mathbf{C}}$ *a feedback operator that takes a morphism* $S \otimes A \to S \otimes B$ *to a morphism* $A \to B$*, satisfying the axioms of feedback (that we saw in Figure 4) but possibly failing to satisfy the yanking axiom of traced monoidal categories. Then the feedback operator is necessarily of the form*

$$\mathsf{fbk}^{\mathbf{C}}_S(f) \coloneqq (\varepsilon \otimes \mathrm{id}); (\mathrm{id} \otimes f); (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}); (\eta \otimes \mathrm{id})$$

where $\partial_A\colon A \to A$ *is a family of endomorphisms satisfying*

- $\partial_A \otimes \partial_B = \partial_{A\otimes B}$ *and* $\partial_I = \mathrm{id}$*, and*
- $\partial_A ; h = h ; \partial_B$ *for each isomorphism* $h\colon A \cong B$.

In fact, any family of morphisms ∂_A *satisfying these properties determines uniquely a feedback operator that has* ∂_A *as its delay endomorphisms.*

[6] This is the **Int** construction from [22].

Proof. Given a family ∂_S satisfying the two properties, we can define a feedback structure to be $\mathsf{fbk}_S^{\mathbf{C}}(f) \coloneqq (\varepsilon \otimes \mathrm{id}); (\mathrm{id} \otimes f); (\mathrm{id} \otimes \partial_S \otimes \mathrm{id}); (\eta \otimes \mathrm{id})$ and check that it satisfies all the axioms of feedback (Figure 4). Note here that, as expected, the yanking equation is satisfied precisely when delay endomorphisms are identities, $\partial_A = \mathrm{id}_A$.

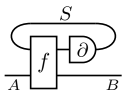


Fig. 8: Feedback from delay.

Let us now show that any feedback operator in a compact closed category is of this form. Indeed,

$$\begin{aligned}\mathsf{fbk}_S^{\mathbf{C}}(f) &= \mathsf{fbk}_S^{\mathbf{C}}((\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id}); (\sigma \otimes \sigma \otimes f); (\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id})) \\ &= (\mathrm{id} \otimes \varepsilon \otimes \varepsilon \otimes \mathrm{id}); (\mathsf{fbk}_S^{\mathbf{C}}(\sigma) \otimes \sigma \otimes f); (\mathrm{id} \otimes \eta \otimes \eta \otimes \mathrm{id}) \\ &= (\varepsilon \otimes \mathrm{id}); (\mathrm{id} \otimes f); (\mathrm{id} \otimes \mathsf{fbk}_S^{\mathbf{C}}(\sigma) \otimes \mathrm{id}); (\eta \otimes \mathrm{id}).\end{aligned}$$

Here we have used the fact that the trace is constructed by two separate pieces: ε and η; and then the fact that the feedback operator, like trace, can be applied "locally" (see the axioms in Figure 4). □

Consider one more time the NOR latch from Figure 1. The algebra of **Span**(**Graph**) does also include a feedback operator that is *not* a trace. This feedback operator is indeed canonical, in that it is the one that makes **Span**(**Graph**) the canonical category with feedback containing spans of functions. Imitating the real-world behavior of the NOR latch is finally possible: one of the components that builds up this feedback (and in fact, the only difference with the previous trace) is a *stateful* delay component. The emergence of state from feedback is witnessed by the St(•) construction.

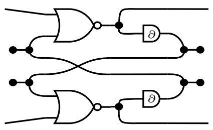


Fig. 9: NOR latch with feedback.

2.4 St(•), the free category with feedback

In this section, we identify the construction that yields the free category with feedback over a symmetric monoidal category. The St(•) construction is a general way of endowing a system with state. It appears multiple times across the literature in slightly different forms: it constructs a stateful resource calculus in [7]; a variant is used for geometry of interaction in [21]; it coincides with the free category with feedback presented in [28]; and yet another, slightly different formulation was given in [24].

Definition 3 (Category of stateful processes, [28]). *Let* $(\mathbf{C}, \otimes, I)$ *be a symmetric monoidal category. We call* St($\mathbf{C}$) *to the category having the same objects as* $\mathbf{C}$ *but where morphisms* $A \to B$ *are pairs* $(S \mid f)$*, consisting of a* state space $S \in \mathbf{C}$ *and a morphism* $f\colon S \otimes A \to S \otimes B$*. We consider morphisms up to isomorphism classes of their state space, and thus*

$$(S \mid f) = (T \mid (h^{-1} \otimes \mathrm{id}); f; (h \otimes \mathrm{id})), \quad \textit{for any isomorphism } h\colon S \cong T.$$

When depicting a stateful process, we explicitly mark the strings forming the space state*. That is, an equivalence class will be depicted as any of its representatives plus some strings marked.*

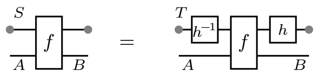


Fig. 10: We depict stateful processes by marking the space state.

We define the identity stateful process *on $A \in \mathbf{C}$ as $(I \mid \mathrm{id}_{I\otimes A})$.* Sequential composition *of the two stateful processes $(S \mid f)\colon A \to B$ and $(T \mid g)\colon B \to C$ is defined by*

$$(S \mid f);(T \mid g) = (S \otimes T \mid (\sigma \otimes \mathrm{id});(\mathrm{id} \otimes f);(\sigma \otimes \mathrm{id});(\mathrm{id} \otimes g)).$$

Parallel composition *of the two stateful processes $(S \mid f)\colon A \to B$ and $(S' \mid f')\colon A' \to B'$ is defined by*

$$(S \mid f) \otimes (S' \mid f') = (S \otimes S' \mid (\mathrm{id} \otimes \sigma \otimes \mathrm{id});(f \otimes f');(\mathrm{id} \otimes \sigma \otimes \mathrm{id})).$$

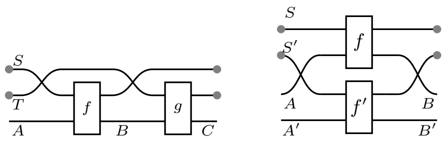


Fig. 11: Sequential and parallel composition of stateful processes.

This defines a symmetric monoidal category. Moreover, it is a category with feedback with the operator

$$\mathsf{store}_T(S \mid f) := (S \otimes T \mid f).$$

Theorem 1. [28] *The category $(\mathsf{St}(\mathbf{C}), \mathsf{store}(\bullet))$ is the free category with feedback over a symmetric monoidal category $\mathbf{C}$.*

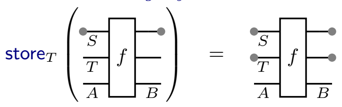


Fig. 12: The $\mathsf{store}(\bullet)$ operation, in diagrammatic terms.

2.5 Examples of categories with feedback

Our first source of examples is *traced monoidal categories*. The axioms of feedback are a strict weakening of the axioms of trace, and every traced category is automatically a category with feedback. A more interesting source of examples is the $\mathsf{St}(\bullet)$ construction we just defined.

Example 1. Consider $\mathsf{St}(\mathbf{Set})$, the free category with feedback over the monoidal structure of sets with the cartesian product. A *Mealy (or deterministic) transition system* with boundaries A and B, and state space S was originally defined [33, §2.1] to be just a function $f\colon S \times A \to S \times B$, which is a morphism of $\mathsf{St}(\mathbf{Set})$ up to isomorphism of the state space. Mealy transitions compose sequentially and in parallel following Definition 3, and they form a category with feedback $\mathbf{Mealy} \coloneqq \mathsf{St}(\mathbf{Set})$.

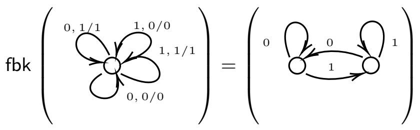


Fig. 13: Feedback of a Mealy transition system.

The feedback operator of Mealy transitions *internalizes* input/output pairs as states. Figure 13 is an example.

It is traditional to depict automata as state/transition graphs. The characterization $\mathbf{Span}(\mathbf{Graph})_* \cong \mathsf{St}(\mathbf{Span}(\mathbf{Set}))$ that we prove in Section 3 lifts the inclusion $\mathbf{Set} \to \mathbf{Span}(\mathbf{Set})$ to a feedback functor $\mathbf{Mealy} \to \mathbf{Span}(\mathbf{Graph})_*$. This inclusion embeds a deterministic transition system into the graph that determines it.

Similarly, when we consider $\mathbf{Set}$ to be the monoidal structure of sets with the disjoint union, the notion we recover is that of an *Elgot automaton* [12], given by a transition function $S + A \to S + B$. These categories of transition systems motivate the work in [24,28].

Example 2. A linear dynamical system with inputs in $\mathbb{R}^n$, outputs in $\mathbb{R}^m$ and state space in $\mathbb{R}^k$ is given by a matrix $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right) \in \mathbf{Mat}_{\mathbb{R}}(k+m, k+n)$ [23]. Two linear dynamical systems $\left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$ and $\left(\begin{smallmatrix} A' & B' \\ C' & D \end{smallmatrix}\right)$ are considered equal whenever there is an invertible matrix $H \in \mathbf{Mat}_{\mathbb{R}}(k,k)$ such that

$$A' = H^{-1}AH, \quad B' = BH, \quad C' = H^{-1}C.$$

Linear dynamical systems are morphisms of a category with feedback which coincides with $\mathsf{St}(\mathbf{Vect}_{\mathbb{R}}^{\oplus})$. The feedback operator is defined by

$$\mathsf{fbk}_l(k, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix}) = (k+l, \begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix})$$

where $\begin{pmatrix} A_1 & A_2 & B_1 \\ A_3 & A_4 & B_2 \\ C_1 & C_2 & D \end{pmatrix} \in \mathbf{Mat}_{\mathbb{R}}(k+l+m, k+l+n)$.

3 Span(Graph): an algebra of transition systems

$\mathbf{Span}(\mathbf{Graph})$ [25] is an algebra of "open transition systems". It has applications in *concurrency theory* and *verification* [24,25,27,29,16], but it has also been recently applied to biological systems [14,15]. Just as ordinary Petri nets have an underlying (firing) semantics in terms of transition systems, **Span**(**Graph**) is used as a semantic universe for a variant of open Petri nets, see [40,9].

An *open transition system* is a morphism of **Span**(**Graph**): it consists of a graph endowed with two *boundaries* or *communication ports*; each transition of the graph has an effect on each boundary, and this data is used to synchronize a network of multiple transition systems. This conceptual picture actually describes a subcategory, **Span**(**Graph**)$_*$, where boundaries are described by mere sets, accounting for the alphabets of signals that open transition systems synchronize on. In this section we recall the details of **Span**(**Graph**)$_*$ and show that it is universal in the following sense:

Span(**Graph**)$_*$ is the free category with feedback over **Span**(**Set**).

3.1 The algebra of Span(Graph).

Definition 4. *A span [4,10] from A to B, both objects of a category* **C***, is a pair of morphisms with a common domain, $A \leftarrow E \rightarrow B$. The object E is the "head" of the span, and the morphisms are the left and right "legs", respectively.*

When the category **C** has pullbacks, we can sequentially compose two spans $A \leftarrow E \rightarrow B$ and $B \leftarrow F \rightarrow C$ into a span $A \leftarrow E \times_B F \rightarrow C$. Here, $E \times_B F$ is the pullback of E and F along B: for instance, in the category **Set** of sets and functions, $E \times_B F$ is the subset of $E \times F$ given by pairs whose two components have the same image on B.

Definition 5. *Let* **C** *be a category with pullbacks.* **Span**(**C**) *is the category that has the same objects as* **C** *and isomorphism classes of spans between them as morphisms. That is, two spans are considered* equal *if there is an isomorphism between their heads that commutes with both legs. Dually, let* **C** *be a category with pushouts.* **Cospan**(**C**) *is the category* **Span**($\mathbf{C}^{op}$)*.*

Span(**C**) is a symmetric monoidal category when **C** has products. The parallel composition of $A \leftarrow E \rightarrow B$ and $A' \leftarrow E' \rightarrow B'$ is given by the componentwise product $A \times A' \leftarrow E \times E' \rightarrow B \times B'$. An example is again **Span**(**Set**).

Definition 6. *The category* **Graph** *of graphs has graphs $G = (s, t \colon E \rightrightarrows V)$ as objects. A morphism $G \rightarrow G'$ in this category is given by two functions $e \colon E \rightarrow E'$ and $v \colon V \rightarrow V'$ such that $e; s' = s; v$ and $e; t' = t; v$. In other words, it is the presheaf category on the diagram $(\bullet \rightrightarrows \bullet)$.*

Recall, however, that we are not interested in the whole **Span**(**Graph**) but only in **Span**(**Graph**)$_*$, the spans of graphs that have a graph of the form $A \rightrightarrows 1$ on the boundaries.

Definition 7. *An open transition system, a morphism of* **Span**(**Graph**)$_*$*, is a span of sets $A \leftarrow E \rightarrow B$ where the head is the set of transitions of a graph $E \rightrightarrows$*

$$\begin{tikzcd}
A \arrow[d, shift left] \arrow[d, shift right] & E \arrow[l, "a"'] \arrow[r, "b"] \arrow[d, shift right, "s"'] \arrow[d, shift left, "t"] & B \arrow[d, shift left] \arrow[d, shift right] \\
1 & V \arrow[l] \arrow[r] & 1
\end{tikzcd}$$

Fig. 14: A morphism of **Span**(**Graph**)$_*$.

V. Two open transition systems are considered equal *if there is an isomorphism between their graphs that commutes with the legs of the span.*

We have been calling "stateless" *to the open transition systems whose graph $E \rightrightarrows V$ has a single vertex, $V = 1$.*

Sequential composition (the *communicating-parallel operation* of [25]) of two open transition systems with spans $A \leftarrow E \rightarrow B$ and $B \leftarrow F \rightarrow C$ and graphs $E \rightrightarrows S$ and $F \rightrightarrows T$ yields the open transition system with span $A \leftarrow E \times_B F \rightarrow C$ and graph $E \times_B F \rightrightarrows S \times T$. This means that the only allowed transitions are those that synchronize E and F on the common boundary B.

$$\begin{tikzcd}
 & & E \times_B F \arrow[dl] \arrow[dr] \arrow[d, shift left] \arrow[d, shift right] & & \\
 & E \arrow[dl] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr] & S \times T \arrow[dl] \arrow[dr] & F \arrow[dl] \arrow[d, shift left] \arrow[d, shift right] \arrow[dr] & \\
A \arrow[d, shift left] \arrow[d, shift right] & S \arrow[dl] \arrow[dr] & B \arrow[d, shift left] \arrow[d, shift right] & T \arrow[dl] \arrow[dr] & C \arrow[d, shift left] \arrow[d, shift right] \\
1 & & 1 & & 1
\end{tikzcd}$$

Fig. 15: Sequential composition in **Span**(**Graph**)$_*$.

Parallel composition (the *non communicating-parallel operation* of [25]) of two open transition systems with spans $A \leftarrow E \rightarrow B$ and $A' \leftarrow E' \rightarrow B'$ and graphs $E \rightrightarrows V$ and $E' \rightrightarrows V'$ yields the open transition system with span $A \times A' \leftarrow E \times E' \rightarrow B \times B'$ and graph $E \times E' \rightrightarrows V \times V'$.

3.2 The components of Span(Graph)

Let us now detail some useful constants of the algebra of **Span**(**Graph**). We will illustrate how to use the algebra with an example in which we construct the NOR latch circuit from Figure 9.

Example 3. In this example, we model the circuit in Figure 9 in **Span**(**Graph**)$_*$. The connectivity of the circuit is modeled with a Frobenius algebra [10] (-●<, >●-, ●—, —●). The corresponding spans are constructed out of diagonals $A \to A \times A$ and units $A \to 1$.

$$(\text{-●<})_A = \{A \leftarrow A \rightarrow A \times A\} \qquad (\text{—●})_A = \{A \leftarrow A \rightarrow 1\}$$

$$(\text{>●-})_A = \{A \times A \leftarrow A \rightarrow A\} \qquad (\text{●—})_A = \{1 \leftarrow A \rightarrow A\}$$

These already induce a compact closed structure (and, therefore, a trace), given by the following spans.

$$(\bullet\!\!-\!\!\!\prec)_A = \{1 \leftarrow A \rightarrow A \times A\} \qquad (\succ\!\!\!-\!\!\bullet)_A = \{A \times A \leftarrow A \rightarrow 1\}$$

In general, any function $f\colon A \to B$ can be lifted covariantly to a span $A \leftarrow A \to B$ and contravariantly to a span $A \leftarrow B \to B$. Any span $A \leftarrow E \to B$ can be lifted to $\mathbf{Span}(\mathbf{Graph})_*$ by making the head represent the graph $E \rightrightarrows 1$. We use this to obtain the graph of the NOR gate (Figure 2). However, components created like this have a single-vertex: they are *stateless*.

We will need a single stateful component to model our circuit, the delay

$$(-\!\boxed{\partial}\!-)_A = \left\{ \begin{tikzcd} & A \times A \arrow[dl, "\pi_2"'] \arrow[d, bend right, "\pi_1"'] \arrow[d, bend left, "\pi_2"] \arrow[dr, "\pi_1"] & \\ A & A & A \end{tikzcd} \right\}.$$

This is *not* an arbitrary choice. This is the canonical delay obtained from the feedback structure[7] in $\mathbf{Span}(\mathbf{Graph})_*$ that gives its universal property.

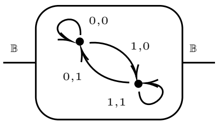


Fig. 16: Delay morphism over the set $\mathbb{B} \coloneqq \{0, 1\}$.

The NOR latch circuit of Figure 9 is the composition of two NOR gates where the outputs of each gate has been copied and fed back as input to the other gate (Figure 17). The algebraic expression, in $\mathbf{Span}(\mathbf{Graph})_*$, of this circuit is obtained by decomposing it into its components.

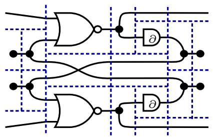


Fig. 17: Decomposing the circuit.

$$(\mathrm{id} \otimes \bullet\!\!-\!\!\!\prec \otimes \bullet\!\!-\!\!\!\prec \otimes \mathrm{id}) ; (\mathtt{NOR} \otimes \sigma \otimes \mathtt{NOR}) ; (-\!\!\!\prec \otimes \mathrm{id} \otimes -\!\!\!\prec)$$
$$; (\mathrm{id} \otimes \partial \otimes \mathrm{id} \otimes \partial \otimes \mathrm{id}) ; (\mathrm{id} \otimes \succ\!\!\!-\!\!\bullet \otimes \succ\!\!\!-\!\!\bullet \otimes \mathrm{id})$$

The graph obtained by the computation of this expression, together with its transitions, is shown in Figure 18. This time, our model is indeed stateful. It has four states: two states representing a correctly stored signal, $\overline{\mathsf{A}} = (1, 0)$ and $\mathsf{A} = (0, 1)$; and two states representing transitory configurations $\mathsf{T}_1 = (0, 0)$ and $\mathsf{T}_2 = (1, 1)$.

We will be controlling the *left boundary*: it can receive a *set* signal, $\mathsf{Set} = \binom{1}{0}$; a *reset* signal, $\mathsf{Reset} = \binom{0}{1}$; none of the two, $\mathsf{Idle} = \binom{0}{0}$; or both of them at the same time, $\mathsf{Unspec} = \binom{1}{1}$, which is known to cause unspecified behavior in a NOR latch. The signal on the *right boundary*, on the other hand, is always equal to the state the transition goes to and does not provide any additional information. Knowing this, we omit it from the drawing in Figure 18.

[7] As in Proposition 1.

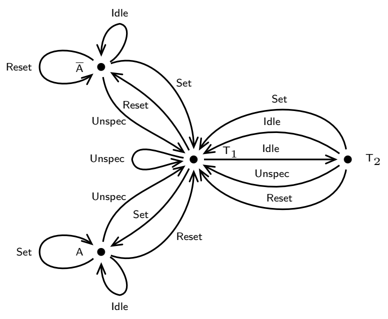


Fig. 18: Span of graphs representing the NOR latch

In normal functioning, activating the signal Set makes the latch transition to the state A in two transition steps. Analogously, activating Reset makes the latch transition to $\overline{\mathsf{A}}$ again in two transition steps. After any of these two cases, deactivating all signals, Idle, keeps the last state.

Moreover, the (real-world) NOR latch has some unspecified behavior that gets also reflected in the graph: activating both Set and Reset at the same time, what we call Unspec, causes the circuit to enter an unstable state where it bounces between the states T_1 and T_2. Our modeling has reflected this "unspecified behavior" as expected.

Feedback and trace. In terms of feedback, the circuit of Figure 18 is equivalently obtained as the result of taking feedback over the following stateless morphism in Figure 19. We know that it is stateless because it is the composition of stateless morphisms.

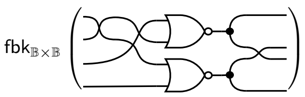


Fig. 19: Applying $\mathsf{fbk}(\bullet)$ over the circuit gives the NOR latch.

But **Span**(**Graph**)$_*$ is also canonically traced: it is actually compact closed. What changes in the modeling if, over the same morphism, we would have used trace instead? As we argued back for Figure 6, we obtain a stateless transition system: it is given by a graph with a single edge. The valid transitions can be now computed explicitly to be

$$\{(\mathsf{Unspec}, \mathsf{T}_1), (\mathsf{Idle}, \mathsf{A}), (\mathsf{Idle}, \overline{\mathsf{A}}), (\mathsf{Set}, \mathsf{A}), (\mathsf{Reset}, \overline{\mathsf{A}})\}$$

These encode important information: they are the *equilibrium* states of the circuit. However, unlike the previous graph, this one would not get us the correct allowed transitions: under this modeling, our circuit could freely bounce between $(\mathsf{Idle}, \mathsf{A})$ and $(\mathsf{Idle}, \overline{\mathsf{A}})$, which is not the expected behavior of a NOR latch.

The fundamental piece making our modeling succeed the previous time was feedback derived from the stateful *delay*. The next section explains in which sense that feedback is canonical.

3.3 Span(Graph) as a category with feedback

This section introduces our main theorem. We start by introducing the mapping that associates to each stateful span of sets the corresponding span of graphs. This mapping is well-defined and lifts to a functor $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$. Finally, we prove that it gives an equivalence $\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*$.

Lemma 1. *The following assignment of stateful processes over* $\mathbf{Span}(\mathbf{Set})$ *to morphisms of* $\mathbf{Span}(\mathbf{Graph})$ *is well defined.*

$$K\left(S \;\middle|\; \begin{tikzcd} & E \arrow[dl, "{(s,a)}"'] \arrow[dr, "{(t,b)}"] & \\ S \times A & & S \times B \end{tikzcd} \right) := \left(\begin{tikzcd} A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "a"'] \arrow[r, "b"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & B \arrow[d, bend right] \arrow[d, bend left] \\ 1 & S \arrow[l] \arrow[r] & 1 \end{tikzcd} \right)$$

Proof. We first check that two *isomorphic* spans are sent to *isomorphic* spans of graphs. Let $S \times A \leftarrow E \rightarrow S \times B$ and $S \times A \leftarrow E' \rightarrow S \times B$ be two spans that are isomorphic with $h \colon E \cong E'$. Then (h, id) is an isomorphism of spans of graphs, also making the relevant diagram commute (Figure 20).

$$\begin{tikzcd} & E \arrow[dl, "{(s,a)}"'] \arrow[dr, "{(t,b)}"] \arrow[dd, "h"] & \\ S \times A & & S \times B \\ & E' \arrow[ul, "{(s',a')}"] \arrow[ur, "{(t',b')}"'] & \end{tikzcd} \qquad \begin{tikzcd} A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "a"'] \arrow[r, "h"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] \arrow[rrr, bend left, "b"] & E' \arrow[r, "b'"] \arrow[d, bend right, "s'"'] \arrow[d, bend left, "t'"] \arrow[ll, bend right, "a'"'] & B \arrow[d, bend right] \arrow[d, bend left] \\ 1 & S \arrow[l] \arrow[r, "\mathrm{id}"] \arrow[rr, bend right] & S \arrow[r] \arrow[ll, bend left] & 1 \end{tikzcd}$$

Fig. 20: Isomorphic spans result in isomorphic spans of graphs.

We show now that the assignment preserves the equivalence relation of stateful processes. Isomorphisms in a category of spans are precisely spans whose two legs are isomorphisms (Proposition 3). This means that an isomorphism in $\mathbf{Span}(\mathbf{Set})$ can be always rewritten as $S \leftarrow S \rightarrow T$, where the left leg is an identity and the right leg is $h \colon S \to T$, some isomorphism. Its inverse can be written analogously as $T \leftarrow S \rightarrow S$. In order to prove that the quotient relation induced by the feedback is preserved, we need to check that equivalent spans of sets are sent to isomorphic spans of graphs. If two spans are equivalent with the isomorphism $h \colon S \cong T$, then the corresponding graphs are isomorphic with the isomorphism of graphs (id, h). □

$$\begin{tikzcd}
 & \cdot \arrow[dl, "h\times \mathrm{id}"'] \arrow[dr, no head] & & E \arrow[dl, "{(s,a)}"'] \arrow[dr, "{(t,b)}"] & & \cdot \arrow[dl, no head] \arrow[dr, "h\times \mathrm{id}"] & \\
T\times A & & S\times A & & S\times B & & T\times B
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "a"'] \arrow[r, "\mathrm{id}"] \arrow[rr, bend left, "b"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & E \arrow[r, "b"] \arrow[ll, bend right, "a"'] \arrow[d, bend right, "s;h"'] \arrow[d, bend left, "t;h"] & B \arrow[d, bend right] \arrow[d, bend left] \\
1 & S \arrow[l] \arrow[r, "h"] \arrow[rr, bend right] & T \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

Fig. 21: Equivalent spans result in isomorphic spans of graphs.

Theorem 2. *There exists an equivalence of categories*

$$\mathsf{St}(\mathbf{Span}(\mathbf{Set})) \cong \mathbf{Span}(\mathbf{Graph})_*.$$

The free category with feedback over **Span**(**Set**) *is equivalent to the full subcategory of* **Span**(**Graph**) *given by single-vertex graphs.*

Proof. We prove that there is a fully faithful functor $K\colon \mathsf{St}(\mathbf{Span}(\mathbf{Set})) \to \mathbf{Span}(\mathbf{Graph})$ defined on objects as $K(A) = (A \rightrightarrows 1)$ and defined on morphisms as in Lemma 1. We have shown there that this assignation is well-defined.

We now show that it is functorial, preserving composition and identities. We can directly check that the identity morphism in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$, as a span $A \leftarrow A \to A$ is sent to the identity span of the graph $A \rightrightarrows 1$. Let us now show that composition is also preserved. The sequential composition of two stateful spans is computed as follows. Let the two stateful spans be given by $S \times A \leftarrow E \to S \times B$ and $T \times B \leftarrow F \to T \times C$, then the composite stateful span is given by $S \times T \times A \leftarrow E \times_B F \to S \times T \times C$. We check that this span is sent to the corresponding composition in **Span**(**Graph**). As **Graph** is a functor category, limits are computed pointwise. Thus, the pullback of two graph morphisms is given by taking the pullbacks of both the vertices (where the pullback on 1 is a product) and the edges; this was shown in Figure 15.

The final step is to show that the original assignment is fully-faithful. We can see that it is full: every span of single-vertex graphs given by $A \leftarrow E \to B$ and $E \rightrightarrows S$ does arise from some span, namely $S \times A \leftarrow E \to S \times B$. Let us check it is also faithful. Suppose that two morphisms in $\mathsf{St}(\mathbf{Span}(\mathbf{Set}))$, $S \times A \leftarrow E \to S \times B$ and $S' \times A \leftarrow E' \to S' \times B$, are sent to equivalent spans of graphs, i.e. there exist $h\colon E \cong E'$ and $k\colon S' \cong S$ making the following diagrams commute.

$$\begin{tikzcd}
A \arrow[d, bend right] \arrow[d, bend left] & E \arrow[l, "a"'] \arrow[r, "h"] \arrow[rr, bend left, "b"] \arrow[d, bend right, "s"'] \arrow[d, bend left, "t"] & E' \arrow[r, "b'"] \arrow[ll, bend right, "a'"'] \arrow[d, bend right, "s'"'] \arrow[d, bend left, "t'"] & B \arrow[d, bend right] \arrow[d, bend left] \\
1 & S \arrow[l] \arrow[r, "k"] \arrow[rr, bend right] & S' \arrow[r] \arrow[ll, bend left] & 1
\end{tikzcd}$$

In this case, we know that $S \times A \leftarrow E \to S \times B$ is equivalent to $S' \times A \leftarrow E \to S' \times B$ because of the equivalence relation on stateful processes. Finally, $S' \times A \leftarrow E \to S' \times B$ is equivalent as a span to $S' \times A \leftarrow E' \to S' \times B$.

We have shown that there exists a fully-faithful functor from the free category with feedback over **Span**(**Set**) to the category **Span**(**Graph**) of spans of graphs. The functor induces an equivalence between St(**Span**(**Set**)) and the full subcategory of **Span**(**Graph**) on single-vertex graphs. □

3.4 Cospan(Graph) as a category with feedback

The previous results can be generalized to any category **C** with all finite limits. By taking **Graph**(**C**) to be the presheaf category of the diagram $(\bullet \rightrightarrows \bullet)$ in **C** and $\mathbf{Span}(\mathbf{Graph}(\mathbf{C}))_*$ the full subcategory on objects of the form $A \rightrightarrows 1$, we can prove the following result.

Theorem 3. *There exists an equivalence of categories*

$$\mathsf{St}(\mathbf{Span}(\mathbf{C})) \cong \mathbf{Span}(\mathbf{Graph}(\mathbf{C})).$$

The free category with feedback over **Span**(**C**) *is equivalent to the full subcategory on* **Span**(**Graph**(**C**)) *given by single-vertex graphs.*

$\mathbf{Cospan}(\mathbf{Graph})_*$ can be also characterized as a free category with feedback. We know that $\mathbf{Cospan}(\mathbf{Set}) \cong \mathbf{Span}(\mathbf{Set}^{op})$, we note that $\mathbf{Graph}(\mathbf{Set}^{op}) \cong \mathbf{Graph}$ (which has the effect of flipping edges and vertices), and we can use Theorem 3 because **Set** has all finite colimits. The explicit assignment is similar to the one shown in Lemma 1.

$$K\left(S \;\middle|\; \begin{tikzcd} & S & \\ E+A \arrow[ur, "{[t|a]}"] & & E+B \arrow[ul, "{[s|b]}"'] \end{tikzcd}\right) := \left(\begin{tikzcd} A \arrow[r, "a"] & S & B \arrow[l, "b"'] \\ 0 \arrow[u, bend left] \arrow[u, bend right] \arrow[r] & E \arrow[u, bend left, "t"] \arrow[u, bend right, "s"'] & 0 \arrow[l] \arrow[u, bend left] \arrow[u, bend right] \end{tikzcd}\right)$$

Corollary 1. *There exists an equivalence of categories* $\mathsf{St}(\mathbf{Cospan}(\mathbf{Set})) \cong \mathbf{Cospan}(\mathbf{Graph})_*$.

Two feedback structures on Cospan(Graph). **Cospan**(**Graph**) is also compact closed and, in particular, traced. As in the case of **Span**(**Graph**), the feedback structure given by the universal property is different from the trace. While the trace identifies the vertices that are the images of the same element on the boundaries, the feedback puts an additional edge between them.

3.5 Syntactical presentation of Cospan(FinGraph)

The observation in Proposition 1 has an important consequence in the case of finite sets. Let us call **FinGraph** to **Graph**(**FinSet**). **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [30], meaning that any morphism written out of the operations of a special commutative Frobenius algebra and the structure of a symmetric monoidal category is precisely a cospan of finite sets (or, in other words, symmetric monoidal functors out of **Cospan**(**FinSet**) correspond to special commutative Frobenius algebras). But we also know that **Cospan**(**FinSet**) with an added generator to its PROP structure [7] (the delay, with the conditions given in Proposition 1) is St(**Cospan**(**FinSet**), or, equivalently, **Cospan**(**FinGraph**). This means that any morphism written out of the operations of a special commutative Frobenius algebra plus a freely added generator of type $(\text{-}\boxed{\partial}\text{-}) \colon 1 \to 1$ is a morphism in **Cospan**(**FinGraph**)$_*$. This is a direct proof of a fact that already appeared in [35].

Proposition 2 ([35], Proposition 3.2). *The category* **Cospan**(**FinGraph**)$_*$ *is the generic special commutative Frobenius monoid with an added generator.*

Proof. It is known that the category **Cospan**(**FinSet**) is the generic special commutative Frobenius algebra [30]. Adding a free generator $(\text{-}\boxed{\partial}\text{-}) \colon 1 \to 1$ to its PROP structure corresponds to adding a family $(\text{-}\boxed{\partial}\text{-})_n \colon n \to n$ with the conditions on Proposition 1. Now, Proposition 1 implies that adding such a generator to **Cospan**(**FinSet**) results in St(**Cospan**(**FinSet**)). Finally, we can use again Theorem 2 to conclude that St(**Cospan**(**FinSet**)) $\cong$ **Cospan**(**FinGraph**)$_*$. □

4 Conclusions and further work

We have characterized **Span**(**Graph**)$_*$, an algebra of open transition systems, as equivalent to the free category with feedback over the category of spans of functions. The St(●) constuction is well-known as a technique of adding state to processes. In [28], it had been characterized as the free category with feedback under a different name. What was missing was a coherent and explicit connection between the two.

We have seen how the St(●) construction creates categories of *transition systems* out of symmetric monoidal categories. As future work, we plan to study how to expand our investigation from mere transition systems to automata, which have initial and accepting states. Initial states are particularly important when providing semantics: for instance, we would like to interpret a transition system $S \times A \to S \times B$ as a stream transducer **Stream**$(A) \to$ **Stream**(B), but this is not possible without an initial state $s_0 \in S$. In future work, we discuss an elegant way of accomodating initial states by considering a more general definition of feedback. This generalized definition modifies the sliding axiom: instead of sliding isomorphisms, we can slide arbitrary classes of morphisms under the image of a functor. We conjecture that sliding point-preserving homomorphisms of pointed sets recovers a well-behaved notion of initial state without modifying the framework of categories with feedback any further.

References

1. Samson Abramsky. What are the fundamental structures of concurrency? We still don't know! *CoRR*, abs/1401.4973, 2014.

2. Jiří Adámek, Stefan Milius, and Jiří Velebil. Elgot algebras. *Electronic Notes in Theoretical Computer Science*, 155:87–109, 2006.
3. John C. Baez and Kenny Courser. Structured cospans. *CoRR*, abs/1911.04630, 2019.
4. Jean Bénabou. Introduction to bicategories. In *Reports of the Midwest Category Seminar*, pages 1–77. Springer, 1967.
5. Nick Benton and Martin Hyland. Traced premonoidal categories. *RAIRO Theor. Informatics Appl.*, 37(4):273–299, 2003.
6. Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes.* EATCS Monographs on Theoretical Computer Science. Springer, 1993.
7. Filippo Bonchi, Joshua Holland, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. Diagrammatic algebra: from linear to concurrent systems. *Proc. ACM Program. Lang.*, 3(POPL):25:1–25:28, 2019.
8. Filippo Bonchi, Paweł Sobociński, and Fabio Zanasi. The Calculus of Signal Flow Diagrams I: Linear Relations on Streams. *Information and Computation*, 252:2–29, 2017.
9. 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.
10. Aurelio Carboni and Robert F. C. Walters. Cartesian Bicategories I. *Journal of pure and applied algebra*, 49(1-2):11–32, 1987.
11. William Henry Eccles and Frank Wilfred Jordan. Improvements in ionic relays. *British patent number: GB*, 148582:704, 1918.
12. Calvin C. Elgot. Monadic computation and iterative algebraic theories. In *Studies in Logic and the Foundations of Mathematics*, volume 80, pages 175–230. Elsevier, 1975.
13. Brendan Fong. Decorated cospans. *Theory and Applications of Categories*, 30(33):1096–1120, 2015.
14. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, Filippo Schiavio, and Simone Tini. CospanSpan(Graph): a compositional description of the heart system. *Fundam. Informaticae*, 171(1-4):221–237, 2020.
15. Alessandro Gianola, Stefano Kasangian, Desiree Manicardi, Nicoletta Sabadini, and Simone Tini. Compositional modeling of biological systems in CospanSpan(Graph). In *Proc. of ICTCS 2020.* CEUR-WS, To appear.
16. Alessandro Gianola, Stefano Kasangian, and Nicoletta Sabadini. Cospan/Span(Graph): an Algebra for Open, Reconfigurable Automata Networks. In Filippo Bonchi and Barbara König, editors, *7th Conference on Algebra and Coalgebra in Computer Science, CALCO 2017, June 12-16, 2017, Ljubljana, Slovenia*, volume 72 of *LIPIcs*, pages 2:1–2:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.
17. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
18. Jean-Yves Girard. Towards a geometry of interaction. *Contemporary Mathematics*, 92(69-108):6, 1989.
19. Masahito Hasegawa. Recursion from cyclic sharing: Traced monoidal categories and models of cyclic lambda calculi. pages 196–213. Springer Verlag, 1997.
20. Masahito Hasegawa. The uniformity principle on traced monoidal categories. In Richard Blute and Peter Selinger, editors, *Category Theory and Computer Science, CTCS 2002, Ottawa, Canada, August 15-17, 2002*, volume 69 of *Electronic Notes in Theoretical Computer Science*, pages 137–155. Elsevier, 2002.

21. Naohiko Hoshino, Koko Muroya, and Ichiro Hasuo. Memoryful geometry of interaction: from coalgebraic components to algebraic effects. In Thomas A. Henzinger and Dale Miller, editors, *Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), CSL-LICS '14, Vienna, Austria, July 14 - 18, 2014*, pages 52:1–52:10. ACM, 2014.
22. André Joyal, Ross Street, and Dominic Verity. Traced monoidal categories. *Mathematical Proceedings of the Cambridge Philosophical Society*, 119:447 – 468, 04 1996.
23. Rudolf Emil Kalman, Peter L. Falb, and Michael A. Arbib. *Topics in mathematical system theory*, volume 1. McGraw-Hill New York, 1969.
24. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Bicategories of processes. *Journal of Pure and Applied Algebra*, 115(2):141–178, 1997.
25. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Span(Graph): A Categorial Algebra of Transition Systems. In Michael Johnson, editor, *Algebraic Methodology and Software Technology, 6th International Conference, AMAST '97, Sydney, Australia, December 13-17, 1997, Proceedings*, volume 1349 of *Lecture Notes in Computer Science*, pages 307–321. Springer, 1997.
26. 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.
27. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A formalization of the IWIM model. In *International Conference on Coordination Languages and Models*, pages 267–283. Springer, 2000.
28. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. Feedback, trace and fixed-point semantics. *ITA*, 36(2):181–194, 2002.
29. Piergiulio Katis, Nicoletta Sabadini, and Robert F. C. Walters. A Process Algebra for the Span(Graph) Model of Concurrency. *arXiv preprint arXiv:0904.3964*, 2009.
30. Stephen Lack. Composing PROPs. *Theory and Applications of Categories*, 13(9):147–163, 2004.
31. Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1978.
32. S. J. Mason. Feedback theory-some properties of signal flow graphs. *Proceedings of the IRE*, 41(9):1144–1156, 1953.
33. George H. Mealy. A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5):1045–1079, 1955.
34. Kate Ponto and Michael Shulman. Traces in symmetric monoidal categories. *Expositiones Mathematicae*, 32(3):248–273, 2014.
35. Robert Rosebrugh, Nicoletta Sabadini, and Robert F. C. Walters. Generic commutative separable algebras and cospans of graphs. *Theory and applications of categories*, 15(6):164–177, 2005.
36. Nicoletta Sabadini, Filippo Schiavio, and Robert F. C. Walters. On the geometry and algebra of networks with state. *Theor. Comput. Sci.*, 664:144–163, 2017.
37. Peter Selinger. A survey of graphical languages for monoidal categories. In *New structures for physics*, pages 289–355. Springer, 2010.
38. Claude E. Shannon. *The Theory and Design of Linear Differential Equation Machines.* Bell Telephone Laboratories, 1942.
39. Paweł Sobociński. A non-interleaving process calculus for multi-party synchronisation. In *2nd Interaction and Concurrency Experience: Structured Interactions, (ICE 2009)*, volume 12 of *EPTCS*, 2009.

40. Paweł Sobociński. Representations of Petri net interactions. In *Concurrency Theory, 21th International Conference, (CONCUR 2010)*, number 6269 in LNCS, pages 554–568. Springer, 2010.

Appendix

Remarks

Remark 1. An alternative definition of feedback, $\mathsf{fbk}(\bullet)$, declares it to be an operator taking instead a morphism $S \otimes A \to B \otimes S$, yielding a morphism $A \to B$.

$$\frac{f \colon S \otimes A \to B \otimes S}{\mathsf{fbk}_S(f) \colon A \to B}$$

Borrowing the names from the exposition of traces in [34], we will call this *twisted feedback*, and contrast it with the *aligned feedback* we have decided to define instead. Let us explain the rationale behind this decision. The advantage of using *twisted feedback* is that we can define sequential composition of stateful processes without ever requiring symmetry of the underlying monoidal category (as in [24]). However, the parallel composition *does* require symmetry in any case; and diagrams seem easier to draw and read when using *aligned feedback*.

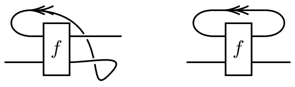


Fig. 22: Twisted vs. aligned feedback

Omitted definitions: Monoidal categories

Definition 8. *A* monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho)$ *is a category* $\mathbf{C}$ *equipped with a functor* $\otimes \colon \mathbf{C} \times \mathbf{C} \to \mathbf{C}$*, a unit* $I \in \mathbf{C}$*, and three natural isomorphisms: the associator* $\alpha_{A,B,C} \colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$*, the left unitor* $\lambda_A \colon I \otimes A \cong A$ *and the right unitor* $\rho_A \colon A \otimes I \cong A$*; such that* $\alpha_{A,I,B} ; (\mathrm{id}_A \otimes \lambda_B) = \rho_A \otimes \mathrm{id}_B$ *and* $(\alpha_{A,B,C} \otimes \mathrm{id}) ; \alpha_{A,B \otimes C,D} ; (\mathrm{id}_A \otimes \alpha_{B,C,D}) = \alpha_{A \otimes B,C,D} ; \alpha_{A,B,C \otimes D}$*. A monoidal category is* strict *if* α*,* λ *and* ρ *are identities.*

Definition 9. *Let* $(\mathbf{C}, \otimes, I, \alpha^{\mathbf{C}}, \lambda^{\mathbf{C}}, \rho^{\mathbf{C}})$ *and* $(\mathbf{D}, \boxtimes, J, \alpha^{\mathbf{D}}, \lambda^{\mathbf{D}}, \rho^{\mathbf{D}})$ *be monoidal categories. A* monoidal functor *(or* strong monoidal functor*) is a triple* (F, ε, μ) *consisting of a functor* $F \colon \mathbf{C} \to \mathbf{D}$ *and two natural isomorphisms* $\varepsilon \colon J \cong F(I)$ *and* $\mu \colon F(A \otimes B) \cong F(A) \boxtimes F(B)$*; such that the associators satisfy* $\alpha^{\mathbf{D}}_{FA,FB,FC} ; (\mathrm{id}_{FA} \otimes \mu_{B,C}) ; \mu_{A,B \otimes C} = (\mu_{A,B} \otimes \mathrm{id}_{FC}) ; \mu_{A \otimes B,C} ; F(\alpha^{\mathbf{C}}_{A,B,C})$*, the left unitor satisfies* $(\varepsilon \otimes \mathrm{id}_{FA}) ; \mu_{I,A} ; F(\lambda^{\mathbf{C}}_A) = \lambda^{\mathbf{D}}_{FA}$ *and the right unitor satisfies* $(\mathrm{id}_{FA} \otimes \varepsilon) ; \mu_{A,I} ; F(\rho^{\mathbf{C}}_{FA}) = \rho^{\mathbf{D}}_{FA}$*. A monoidal functor is a* monoidal equivalence *if it is moreover an equivalence of categories. Two monoidal categories are monoidally equivalent if there exists a monoidal equivalence between them.*

Theorem 4 (Coherence theorem, [31]). *Every monoidal category is monoidally equivalent to a strict monoidal category.*

Let us comment further on how we use the coherence theorem. Each time we have a morphism $f \colon A \to B$ in a monoidal category, we have a corresponding morphism $A \to B$ in its strictification. This morphism can be lifted to the original category to uniquely produce, say, a morphism $(\lambda_A; f; \lambda_B{}^{-1})\colon I \otimes A \to I \otimes B$. Each time the source and the target are clearly determined, we simply write f again for this new morphism.

As an example, consider the statement of the vanishing axiom, which says that for $f\colon A \to B$, we have $\mathsf{fbk}_I(f) = f$. However, the feedback operator needs to be applied to a morphism $I \otimes A \to I \otimes B$, and the only such morphism that is mapped again to $f\colon A \to B$ in the equivalent strict monoidal category is $(\lambda_A; f; \lambda_B{}^{-1})\colon I \otimes A \to I \otimes B$. Thus, we are really stating that

$$\mathsf{fbk}_I(\lambda_A; f; \lambda_B{}^{-1}) = f.$$

In the same vein, the joining axiom really states that

$$\mathsf{fbk}_S(\mathsf{fbk}_T(f)) = \mathsf{fbk}_{S\otimes T}(\alpha_{S,T,A}; f; \alpha^{-1}_{S,T,B}).$$

The reason to avoid this explicit notation on our definitions and proofs is that it would quickly become verbose and distractive. Equations seem conceptually easier to understand when written assuming the coherence theorem. And we are following this criterion anyway when employing string diagrams. In fact, in [28], strictness is assumed since the beginning to facilitate the reading, even when we have shown it is not a necessary assumption.

There are cases where we do need to be careful about the correct use of associators and unitors. For instance, when writing stateful processes, we could be tempted to conclude that, for any $f\colon ((S \otimes T) \otimes R) \otimes A \to ((S \otimes T) \otimes R) \otimes B$, the following equation holds $((S \otimes T) \otimes R \mid f) = (S \otimes (T \otimes R) \mid f)$ without needing to invoke the equivalence relation. This would allow us to construct the category $\mathsf{St}(\bullet)$ of stateful processes without having to quotient them by the equivalence relation. However, this equality is only enabled by the fact that $\alpha_{S,T,R}$ is an isomorphism: we have

$$((S \otimes T) \otimes R \mid f) = (S \otimes (T \otimes R) \mid \alpha_{S,R,T}; f; \alpha^{-1}_{S,R,T}),$$

even if we write the equation omitting the coherence maps. This is also what allows us to notate stateful processes diagramatically. We only mark the wires forming the space state; the order in which they are tensored does not matter thanks again to the equivalence relation that we are imposing.

Definition 10. *A symmetric monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho, \sigma)$ is a monoidal category $(\mathbf{C}, \otimes, I, \alpha, \lambda, \rho)$ equipped with a braiding $\sigma_{A,B}\colon A \otimes B \to B \otimes A$, which satisfies the hexagon equation $\alpha_{A,B,C}; \sigma_{A,B\otimes C}; \alpha_{B,C,A} = (\sigma_{A,B} \otimes \mathrm{id}); \alpha_{B,A,C}; (\mathrm{id} \otimes \sigma_{A,C})$ and additionally satisifes $\sigma_{A,B}; \sigma_{B,A} = \mathrm{id}$.*

Definition 11. *A symmetric monoidal functor $F\colon \mathbf{C} \to \mathbf{D}$ between two symmetric monoidal categories $(\mathbf{C}, \sigma^{\mathbf{C}})$ and $(\mathbf{D}, \sigma^{\mathbf{D}})$ is a strong monoidal functor such that $\sigma^{\mathbf{D}}; \psi = \psi; F(\sigma^{\mathbf{C}})$.*

Definition 12. *A traced monoidal category [22,37] is a category with feedback that additionally satisfies the* yanking axiom $\mathsf{fbk}(\sigma) = \mathrm{id}$ *and that additionally satisfies the* sliding axiom, $\mathsf{fbk}_T(f; (h \otimes \mathrm{id})) = \mathsf{fbk}_S((h \otimes \mathrm{id}); f)$*, for an arbitrary morphism $h\colon S \to T$.*

Omitted proofs: spans

Proposition 3. *Let* $\mathbf{C}$ *be a category with all finite limits. In its category of spans,* $\mathbf{Span}(\mathbf{C})$*, an isomorphism* $A \cong B$ *is always of the form* $A \leftarrow A \rightarrow B$*, where the left leg is an identity and the right leg is an isomorphism in* $\mathbf{C}$*.*

Proof. Let $A \leftarrow E \rightarrow B$ and $B \leftarrow E' \rightarrow A$ be inverses. This means that the following pullback diagrams commute.

$$\begin{tikzcd}
& & A \arrow[dl, "x"'] \arrow[dr, "x'"] \arrow[ddll, bend right, "\mathrm{id}"'] \arrow[ddrr, bend left, "\mathrm{id}"] & & \\
& E \arrow[dl, "a"'] \arrow[dr, "b"] & & E' \arrow[dl, "b'"'] \arrow[dr, "a'"] & \\
A & & B & & A
\end{tikzcd}
\qquad
\begin{tikzcd}
& & B \arrow[dl, "y'"'] \arrow[dr, "y"] \arrow[ddll, bend right, "\mathrm{id}"'] \arrow[ddrr, bend left, "\mathrm{id}"] & & \\
& E' \arrow[dl, "b'"'] \arrow[dr, "a'"] & & E \arrow[dl, "a"'] \arrow[dr, "b"] & \\
B & & A & & B
\end{tikzcd}$$

Fig. 23: Pullback diagrams repreesnting compositions in $\mathbf{Span}(\mathbf{C})$.

We know then that a, a', b and b' are split epimorphisms, whereas x, x', y and y' are their corresponding split monomorphisms. Let us prove that they are also isomorphisms.

The span $E \leftarrow E \rightarrow E'$, given by $b; y' \colon E \rightarrow E'$ and the identity, is a cone over the diagram $E \rightarrow B \leftarrow E'$, because $y'; b' = \mathrm{id}_B$. By the universal property of the pullback, there exists a unique $h \colon E \rightarrow A$ such that $h; x = \mathrm{id}_E$ and $h; x' = b; y'$. This proves that x is a split epimorphism and, hence, an isomorphism. The same reasoning can be repeated for x', y and y'. It follows that a, b, a', b' are isomorphisms as well.

Once we have shown that the original $A \leftarrow E \rightarrow B$ and $B \leftarrow E' \rightarrow A$ have pairs of isomorphisms as legs, we can rewrite them as

$$A \xleftarrow{\mathrm{id}} A \xrightarrow{a^{-1};b} B \qquad \text{and} \qquad A \xleftarrow{a'^{-1};b'} B \xrightarrow{\mathrm{id}} B,$$

where one leg is an identity and the other is an isomorphism. □

8. Profunctor Optics, a Categorical Update

Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, Mario Román
Compositionality, 2022

Abstract: Optics are bidirectional data accessors that capture data transformation patterns such as accessing subfields or iterating over containers. Profunctor optics are a particular choice of representation supporting modularity, meaning that we can construct accessors for complex structures by combining simpler ones. Profunctor optics have previously been studied only in an unenriched and non-mixed setting, in which both directions of access are modelled in the same category. However, functional programming languages are arguably better described by enriched categories; and we have found that some structures in the literature are actually mixed optics, with access directions modelled in different categories. Our work generalizes a classic result by Pastro and Street on Tambara theory and uses it to describe mixed V-enriched profunctor optics and to endow them with V-category structure. We provide some original families of optics and derivations, including an elementary one for traversals. Finally, we discuss a Haskell implementation.

Declaration: *Hereby I declare that my contribution to this manuscript was to: write most of the manuscript as main author, write the code, prove the main results. Jeremy Gibbons was my supervisor on previous work on the same topic. Bartosz Milewski proposed the research question. Derek Elkins identified multiple improvements on the mathematics. Fosco Loregian aided the exposition of coend calculus. Emily Pillmore helped preparing the Haskell code. Bryce Clarke identified multiple problems with a previous draft of the paper and helped correcting them.*

PROFUNCTOR OPTICS: A CATEGORICAL UPDATE

BRYCE CLARKE, DEREK ELKINS, JEREMY GIBBONS, FOSCO LOREGIAN, BARTOSZ MILEWSKI, EMILY PILLMORE, AND MARIO ROMÁN

Abstract. Optics are bidirectional data accessors that capture data transformation patterns such as accessing subfields or iterating over containers. Profunctor optics are a particular choice of representation supporting modularity, meaning that we can construct accessors for complex structures by combining simpler ones. Profunctor optics have previously been studied only in an unenriched and non-mixed setting, in which both directions of access are modelled in the same category. However, functional programming languages are arguably better described by enriched categories; and we have found that some structures in the literature are actually *mixed* optics, with access directions modelled in different categories. Our work generalizes a classic result by Pastro and Street on Tambara theory and uses it to describe *mixed* $\mathcal{V}$-enriched profunctor optics and to endow them with $\mathcal{V}$-category structure. We provide some original families of optics and derivations, including an elementary one for *traversals*. Finally, we discuss a Haskell implementation.

Keywords: lens, profunctor, Tambara module, coend calculus.

Contents

1. Introduction 3
1.1. Optics 3
1.2. Modularity 4
1.3. Profunctor optics 5
1.4. Mixed profunctor optics 6
1.5. Coend Calculus 6
1.6. Contributions 8
1.7. Synopsis 8
1.8. Setting 8
2. Optics 9
3. Examples of optics 10
3.1. Lenses and prisms 10
3.2. Traversals 16
3.3. Grates 21
3.4. Getters, reviews and folds 22
3.5. Setters and adapters 23
3.6. Optics for (co)free 24
4. Tambara theory 25
4.1. Generalized Tambara modules 26
4.2. Pastro-Street's "double" comonad 27
4.3. Pastro-Street's "double" promonad 32
4.4. Profunctor representation theorem 34
5. Haskell implementation 36
5.1. Concepts of enriched category theory 36
5.2. Mixed profunctor optics 37
5.3. Combinators 38
5.4. Table of optics 39
6. Conclusions 40
6.1. Van Laarhoven encoding 40
6.2. Related work 41
6.3. Further work 42
Acknowledgements 42
References 43

1. Introduction

1.1. **Optics.** *Optics* are an abstract representation of some common patterns in bidirectional data accessing. The most widely known optics are *lenses*: 'focussing' on a subfield A of a larger data structure S through a pair of functions *view* $(S \to A)$ and *update* $(S \times A \to S)$ that respectively retrieve and modify the field. *Lenses* have been used in functional programming as a compositional solution to the problem of accessing fields of nested data structures [10, 40] (Figure 1).

```
data Address = Address
  { street'  :: String
  , town'    :: String
  , country' :: String }

viewStreet :: Address -> String
viewStreet (Address str twn ctr) = str

updateStreet :: Address -> String -> Address
updateStreet addr str = addr {street' = str}

example :: Address
example = Address
  { street'  = "221b Baker Street"
  , town'    = "London"
  , country' = "UK" }

>>> example
Address { street'  = "221b Baker Street"
        , town'    = "London"
        , country' = "UK"}

>>> viewStreet example
"221b Baker Street"

>>> updateStreet example "4 Marylebone Road"
Address { street'  = "4 Marylebone Road"
        , town'    = "London"
        , country' = "UK"}
```

Figure 1. Lenses are pairs of 'view' and 'update' functions that capture the repeating pattern of accessing subfields. Here, `viewStreet` extracts a field from a record, and `updateStreet` updates that field.

As the understanding of these data accessors grew, different *families of optics* were introduced for a variety of different types (e.g. *prisms* for disjoint unions and *traversals* for containers), each one of them capturing a particular data accessing pattern (Figure 2).

1.2. **Modularity.** It is straightforward to compose two lenses, one given by $S \to A$ and $S \times A \to S$ and the other given by $A \to X$ and $A \times X \to A$, in order to access nested subfields. However, explicitly writing down this composition (or explaining it to a computer) can be tedious. Intercomposability only becomes increasingly difficult as other data accessors enter the stage: composing a *lens* given by $S \to A$ and $S \times A \to S$ with a *prism* given by $A \to X + A$ and $X \to A$ can produce a function $S \to X \times (X \to S) + S$, which is neither a *lens* nor a *prism* but a different optic known as *affine traversal*. Implementing explicitly a composition like this for every possible pair of optics would be prone to errors and result in a large codebase. However, we would like optics to behave *modularly*; in the sense that, given two optics, it should be possible to join them into a composite optic that directly accesses the innermost subfield.

```
buildString :: Address -> String
buildString (Address str twn ctr) =
  str ++ ", " ++ twn ++ ", " ++ ctr

verifyAddress :: String -> Either String Address
verifyAddress a = case splitOn ", " a of
      [str, twn, ctr] -> Right (Address str twn ctr)
      failure -> Left a

asAddress :: Prism Address String
asAddress = mkPrism verifyAddress buildString

>>> "221b Baker Street, London, UK" ?. asAddress
Just (Address
  { street'  = "221b Baker Street"
  , town'    = "London"
  , country' = "UK" })
```

FIGURE 2. A prism is given by a pair of functions `match` and `build` that account for the possiblity that the pattern matching does not succeed (for instance, when the string does not have the correct format). In the figure, we verify whether a string can be parsed as an address. The combinator `(?.)` returns the results using the `Maybe` monad (see §5.3.1).

Perhaps surprisingly, many implementations allow the programmer to wrap optics into a different representation and then use *ordinary function composition* to construct composite optics.

How is it possible to compose two constructs, that are not functions, using ordinary function composition? Implementations provided by popular libraries such as lens [20], mezzolens [26] in Haskell, or profunctor-optics [12] in Purescript, achieve this effect by using different representations of optics in terms of polymorphic functions and the Yoneda lemma. This paper focuses on the encoding known as *profunctor representation*, which is based on the isomorphism between lenses (and optics in general) and functions that are polymorphic over profunctors with a particular algebraic structure called a *Tambara module*. Optics under this encoding are called *profunctor optics*.

1.3. **Profunctor optics.** Profunctor optics are a compositional solution to the problem of easily combining data accessors for the fields of nested data structures [10, 40].

Different families of profunctor optics are intercomposable. When we use the profunctor representation of optics, composing optics of different kinds becomes also a particular case of ordinary function composition. Together, all *families of profunctor optics* form a powerful language for modular data access. Consider the example of Figure 3, where a lens and a prism are used in conjunction to manipulate parts of a string.

```
>>> let place = "221b Baker St, London, UK"

>>> place ?. asAddress . street
Just "221b Baker St"

>>> place & asAddress . street .~ "4 Marylebone Rd"
  "4 Marylebone Rd, London, UK"
```

FIGURE 3. The composition of a prism (`asAddress`) and a lens (`street`) produces a composite optic (`asAddress . street`). This optic (an "affine traversal", see §3.24) is used to parse a string and then access and modify one of its subfields.

Moreover, optics can be used to entirely change not only the value but the type of the focus, and propagate that change back to the original data structure. These are called *type-variant* optics, in contrast with the *type-invariant* optics we have introduced so far (Figure 4). In that case, the functions defining the optic need to account for that type change (commonly, by also introducing polymorphism), but the internal representation will work the same. The optics we discuss in this paper are assumed to be type-variant, with type-invariant optics being a special case.

```
data Timestamped a = Timestamped
  { created' :: UTCTime
  , modified' :: UTCTime
  , contents' :: a }

viewContents :: Timestamped a -> a
viewContents = contents'

updateContents :: Timestamped a -> b -> Timestamped b
updateContents x b = x {contents' = b}

contents :: Lens' a b (Timestamped a) (Timestamped b)
contents = mkLens' viewContents updateContents
```

Figure 4. A type-variant lens that targets the contents of a value paired with creation and modification timestamps. The lens is constructed from two functions `viewContents` and `updateContents`.

In its profunctor representation, each optic is written as a single function that is polymorphic over profunctors with a certain algebraic structure. For instance, *lenses* can be written as functions polymorphic over *cartesian* profunctors, whereas *prisms* can be written as functions polymorphic over *cocartesian* profunctors [30, §3]. Milewski [24] identified these algebraic structures (cartesian profunctors, cocartesian profunctors, ...) as *Tambara modules* [39] and used a result by Pastro and Street [29] to propose a unified definition of optic. This definition has been later extended by Boisseau and Gibbons [3] and Riley [33], both using different techniques and proposing laws for optics.

1.4. **Mixed profunctor optics.** However, the original result by Pastro and Street cannot be used directly to unify all the optics that appear in practice. Our work generalizes this result, going beyond the previous definitions of optic to cover *mixed* [33, §6.1] and *enriched optics*.

Our generalized profunctor representation theorem captures optics already present in the literature and makes it possible to upgrade them to more sophisticated definitions. For instance, many generalizations of *lenses* in functional programming are shown to be particular cases of a more refined definition that uses mixed optics (Definition 3.1). We also show derivations for some new optics that were not present in the literature (Definitions 3.8, 3.26 and 3.31). Finally, Milewski [24] posed the problem of fitting the three basic optics (lenses, prisms and traversals) into an elementary pattern; lenses and prisms had been covered in his work, but traversals were missing. We present a new description of *traversals* in terms of power series functors (Proposition 3.22) whose derivation is more direct than the ones based on *traversables* as studied by Jaskelioff and Rypacek [17].

1.5. **Coend Calculus.** *Coend calculus* is a branch of category theory that describes the behaviour of *ends* and *coends*, certain universal objects associated with profunctors $P : \mathbf{C}^{op} \times \mathbf{C} \to \mathcal{V}$. Ends can be thought of as some form of universal quantifier, whereas coends can be thought of as their existential counterparts.

Ends are subobjects of the product $\prod_{X \in \mathbf{C}} P(X,X)$, whereas coends result from quotienting the coproduct $\coprod_{X \in \mathbf{C}} P(X,X)$. Both take into account the fact that P depends on two "terms", covariantly on the second, and contravariantly on the first. Explicitly, the *end* is the equalizer of the action of morphisms on both arguments of the profunctor, whereas the *coend* is dually defined as a coequalizer.

Definition 1.1 (Ends and coends)**.**

$$\mathrm{end}(P) \coloneqq \mathrm{eq}\left(\textstyle\prod_{X \in \mathbf{C}} P(X,X) \rightrightarrows \prod_{f \colon A \to B} P(A,B) \right),$$

$$\mathrm{coend}(P) \coloneqq \mathrm{coeq}\left(\textstyle\coprod_{f \colon B \to A} P(A,B) \rightrightarrows \coprod_{X \in \mathbf{C}} P(X,X) \right).$$

Ends are usually denoted with a subscripted integral, whereas *coends* use a superscripted integral.

$$\int_{X \in \mathbf{C}} P(X,X) \coloneqq \mathrm{end}(P), \qquad \int^{X \in \mathbf{C}} P(X,X) \coloneqq \mathrm{coend}(P).$$

In both cases, X is a dummy variable, and we consider $\int_{X \in \mathbf{C}} P(X,X)$ and $\int_{Y \in \mathbf{C}} P(Y,Y)$ 'equivalent modulo α-conversion'. The notation draws on an analogy with elementary calculus. An integral $\int f(x)\ dx$ depends "covariantly" on the variable x defined, say, on $\mathbb{R}^n$, whereas the differential "dx" can be regarded as an element of the *dual* space $(\mathbb{R}^n)^*$. A more striking analogy is that co/ends satisfy a form of 'Fubini rule' and a 'Dirac delta' integration rule (see Proposition 1.3 and Proposition 1.2).

Theorems involving ends and coends can be proved using their universal properties. Here, we offer a terse account of coend calculus [4, 21]. Using the calculus based on the following rules, it is possible to construct isomorphisms between objects of a category by means of a chain of 'deduction steps'.

Proposition 1.2. *Yoneda and coYoneda reductions.*

$$\int_{X \in \mathbf{C}} \mathcal{V}(\mathbf{C}(A,X), FX) \cong FA. \qquad \int^{X \in \mathbf{C}} \mathbf{C}(X,A) \otimes FX \cong FA,$$

where $\otimes$ is the tensor product in the monoidal category $\mathcal{V}$ (the base of the enrichment for $\mathbf{C}$) and $F \colon \mathbf{C} \to \mathcal{V}$ is a co-presheaf (there are analogous identities for presheaves).

Proposition 1.3. *Fubini rule.*

$$\int_{X_1 \in \mathbf{C}} \int_{X_2 \in \mathbf{C}} P(X_1,X_2,X_1,X_2) \cong \int_{X_2 \in \mathbf{C}} \int_{X_1 \in \mathbf{C}} P(X_1,X_2,X_1,X_2).$$

$$\int^{X_1 \in \mathbf{C}} \int^{X_2 \in \mathbf{C}} P(X_1,X_2,X_1,X_2) \cong \int^{X_2 \in \mathbf{C}} \int^{X_1 \in \mathbf{C}} P(X_1,X_2,X_1,X_2).$$

Proposition 1.4. *Continuity and cocontinuity.*

$$\mathcal{V}\left(A, \int_{X \in \mathbf{C}} P(X,X)\right) \cong \int_{X \in \mathbf{C}} \mathcal{V}(A, P(X,X)).$$

$$\mathcal{V}\left(\int^{X\in\mathbf{C}} P(X,X), A\right) \cong \int_{X\in\mathbf{C}} \mathcal{V}(P(X,X),A).$$

Proposition 1.5. Natural transformations are ends.

$$\int_{X\in\mathbf{C}} \mathbf{D}(FX,GX) \cong [\mathbf{C},\mathbf{D}](F,G)$$

Finally, coend calculus expressions are commonly simplified using *adjunctions* ($F \dashv G$) as $\mathbf{D}(FX,Y) \cong \mathbf{C}(X,GY)$.

1.6. **Contributions.** Our first contribution is the derivation and partial classification of mixed optics, covering both optics existing in the literature and some novel ones, all following a unified definition (Definition 2.1). Our work completes and extends the classification of (non-mixed) optics in [3, 33].

Explicitly, we present a new family of optics in Definition 3.8, that unifies new examples with some optics already present in the literature, such as *achromatic lenses* [2, §5.2]. We introduce an original derivation showing that *monadic lenses* [1] are mixed optics in Proposition 3.6. Similarly, in Proposition 3.4, we present a novel derivation showing that the appropiate generalization of lenses to an arbitrary monoidal category [37, §2.2] is not an optic but a mixed optic. We give a unified definition of *lens* in Definition 3.1 that for the first time can be specialized to all of these previous examples. Finally, we present a new derivation of the optic known as *traversal* in Proposition 3.22.

Our second contribution is the definition of the enriched category of mixed profunctor optics. The construction requires a generalization of the *Tambara modules* of [29] that had been used to define categories of profunctor optics [3, 33] to *generalized Tambara modules.* This is done in Section 4. As a corollary, we extend the result that justifies the use of the profunctor representation of optics in functional programming to the case of enriched and mixed optics (Theorem 4.14), endowing them with $\mathcal{V}$-category structure.

1.7. **Synopsis.** We introduce the definition of *mixed optic* in Section 2. Section 3 describes some practical examples from functional programming and shows how they are captured by the definition. Section 4 describes how the theory of Tambara modules can be applied to obtain a profunctor representation for optics. Section 6 contains concluding remarks. The Appendix (Section 5) introduces the details of a full Haskell implementation.

1.8. **Setting.** We shall work with categories enriched over a Bénabou cosmos $(\mathcal{V},\otimes,I)$; that is, a (small)-complete and cocomplete symmetric monoidal closed category. In particular, $\mathcal{V}$ is enriched over itself, and we write the internal hom-object between $A,B\in \mathrm{Obj}(\mathcal{V})$ as $[A,B]$ or just $\mathcal{V}(A,B)$ when it does not cause ambiguity. Our intention is to keep a close eye on the applications in functional programming: the enriching category $\mathcal{V}$ should be thought of as the category whose objects model the types of an idealized programming language and whose morphisms model the programs. Because of this, $\mathcal{V}$ will be cartesian in many of the examples. We can, however, remain agnostic as to which specific $\mathcal{V}$ we are addressing.

For calculations, we make significant use of coend calculus as described, for instance, by Loregian [21]. The proofs in this paper can be carried out without assuming choice or excluded middle, but there is an important set-theoretical issue: in some of the examples, we compute coends over non-small categories. We implicitly fix a suitable Grothendieck universe and our categories are to be considered small with respect to that universe. As Riley [33, §2] notes, this will not be a problem in general: even if some sets of optics are indexed by large categories and we cannot argue their existence using the cocompleteness of **Sets**, we will still find them represented by objects in the category.

2. Optics

Our first goal is to give a unified definition that captures what it means to be an **optic**. We have seen so far how *lenses* and *prisms* work (Figures 1 and 2). A common pattern can be extracted from these two cases. *Lenses* can be constructed when we have a function $S \to M \times A$ that splits some data structure of type S into something of the form $M \times A$ and then a second function $M \times A \to S$ recombines it back. Here, A is the type of the field we want to focus on, and M is the type containing the remaining information that constitutes S. From this split, we can extract the pair of functions $\mathbf{C}(S,A)\times\mathbf{C}(S\times A,S)$ that define a lens. *Prisms* can be constructed when we have a function that can split some data structure of type S into something of the form $M + A$ and put the pieces together again.

The structure that is common to all optics is that they split a bigger data structure of type $S \in \mathbf{C}$ into $M \textcircled{\scriptsize L} A$, some *focus* of type $A \in \mathbf{C}$ and some *context* or *residual* $M \in \mathbf{M}$ around it. We cannot access the context directly, but we can still use its shape to update the original data structure, replacing the current focus by a new one, $M \textcircled{\scriptsize R} B$. The definition will capture this fact imposing a quotient relation on the possible contexts; this quotient is expressed by the dinaturality condition of a coend. The category of contexts $\mathbf{M}$ will be monoidal, allowing us to compose optics with contexts M and N into an optic with context $M \otimes N$. Finally, as we want to capture *type-variant* optics, we leave open the possibility of the new focus being of a different type $B \in \mathbf{D}$, possibly in a different category, which yields a new data structure of type $T \in \mathbf{D}$. This is summarized in Figure 5.

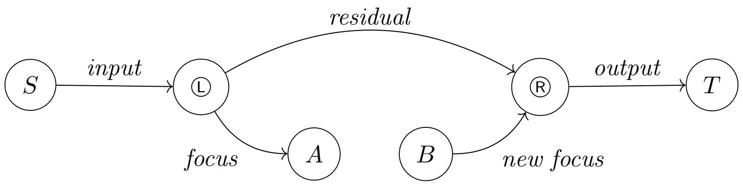


Figure 5. The common structure of an optic. Profunctors give semantics for diagrams of this kind [36].

Multiple definitions of optics of increasing generality have been given in [3, 24, 33]. We encompass all of them under an abstract definition in terms of monoidal actions.

Let $(\mathbf{M}, \otimes, I, a, \lambda, \rho)$ be a monoidal $\mathcal{V}$-category [6]. Let it act on two arbitrary $\mathcal{V}$-categories $\mathbf{C}$ and $\mathbf{D}$ by means of strong monoidal $\mathcal{V}$-functors $(\text{Ⓛ})\colon \mathbf{M} \to [\mathbf{C}, \mathbf{C}]$ and $(\text{Ⓡ})\colon \mathbf{M} \to [\mathbf{D}, \mathbf{D}]$; and let us write

$$\phi_A\colon A \cong I \text{ Ⓛ } A, \qquad \phi_{M,N,A}\colon M \text{ Ⓛ } N \text{ Ⓛ } A \cong (M \otimes N) \text{ Ⓛ } A,$$
$$\varphi_B\colon B \cong I \text{ Ⓡ } B, \qquad \varphi_{M,N,B}\colon M \text{ Ⓡ } N \text{ Ⓡ } B \cong (M \otimes N) \text{ Ⓡ } B,$$

for the structure isomorphisms of the strong monoidal actions Ⓛ and Ⓡ, which we use in infix notation.

Definition 2.1. Let $S, A \in \mathbf{C}$ and $T, B \in \mathbf{D}$. An **(Ⓛ, Ⓡ)-optic** from (S, T) with the focus on (A, B) is a (generalized) element of the following object described as a coend:

$$\mathbf{Optic}_{\text{Ⓛ},\text{Ⓡ}}((A, B), (S, T)) \coloneqq \int^{M \in \mathbf{M}} \mathbf{C}(S, M \text{ Ⓛ } A) \otimes \mathbf{D}(M \text{ Ⓡ } B, T).$$

The two strong monoidal actions Ⓛ and Ⓡ represent the two different ways in which the context interacts with the focus: one when the data structure is decomposed and another one, possibly different, when it is reconstructed. By varying these two actions we will recover many examples from the literature and introduce some new ones, as the table in Figure 6 summarizes.

Name	Description	Actions	Base
Adapter	$\mathbf{C}(S, A) \otimes \mathbf{D}(B, T)$	$(\mathbf{Optic}_{\mathrm{id},\mathrm{id}})$	$\mathcal{V}, \otimes$
Lens	$\mathbf{C}(S, A) \times \mathbf{D}(S \bullet B, T)$	$(\mathbf{Optic}_{\times,\bullet})$	$\mathcal{W}, \times$
Monoidal lens	$\mathbf{CCom}(S, A) \times \mathbf{C}(\mathcal{U}S \otimes B, T)$	$(\mathbf{Optic}_{\otimes,\mathcal{U}\times})$	$\mathcal{W}, \times$
Algebraic lens	$\mathbf{C}(S, A) \times \mathbf{D}(\Psi S \bullet B, T)$	$(\mathbf{Optic}_{\mathcal{U}\times,\mathcal{U}\bullet})$	$\mathcal{W}, \times$
Monadic lens	$\mathcal{W}(S, A) \times \mathcal{W}(S \times B, \Psi T)$	$(\mathbf{Optic}_{\times,\rtimes})$	$\mathcal{W}, \times$
Linear lens	$\mathbf{C}(S, [B, T] \bullet A)$	$(\mathbf{Optic}_{\bullet,\otimes})$	$\mathcal{V}, \otimes$
Prism	$\mathbf{C}(S, T \bullet A) \times \mathbf{D}(B, T)$	$(\mathbf{Optic}_{\bullet,+})$	$\mathcal{W}, \times$
Coalg. prism	$\mathbf{C}(S, \Theta T \bullet A) \times \mathbf{D}(B, T)$	$(\mathbf{Optic}_{\mathcal{U}\bullet,\mathcal{U}+})$	$\mathcal{W}, \times$
Grate	$\mathbf{D}([S, A] \bullet B, T)$	$(\mathbf{Optic}_{\{\ ,\ \},\bullet})$	$\mathcal{V}, \otimes$
Glass	$\mathbf{C}(S \times [[S, A], B], T)$	$(\mathbf{Optic}_{\times[\ ,\ ],\times[\ ,\ ]})$	$\mathcal{W}, \times$
Affine traversal	$\mathbf{C}(S, T + A \otimes \{B, T\})$	$(\mathbf{Optic}_{+\otimes,+\otimes})$	$\mathcal{W}, \times$
Traversal	$\mathcal{V}(S, \sum^n A^n \otimes [B^n, T])$	$(\mathbf{Optic}_{\mathrm{Pw},\mathrm{Pw}})$	$\mathcal{V}, \otimes$
Kaleidoscope	$\sum_n \mathcal{V}([A^n, B], [S^n, T])$	$(\mathbf{Optic}_{\mathrm{App},\mathrm{App}})$	$\mathcal{V}, \otimes$
Setter	$\mathcal{V}([A, B], [S, T])$	$(\mathbf{Optic}_{\mathrm{ev},\mathrm{ev}})$	$\mathcal{V}, \otimes$
Fold	$\mathcal{V}(S, \mathcal{L}A)$	$(\mathbf{Optic}_{\mathrm{Foldable},*})$	$\mathcal{V}, \otimes$

FIGURE 6. Table of optics, together with their explicit description and their generating monoidal actions.

The purpose of an abstract unified definition is twofold: firstly, it provides a framework to classify existing optics and explore new ones, as we do in Section 3; and secondly, it enables a unified profunctor representation, which we present in Section 4.

3. EXAMPLES OF OPTICS

3.1. **Lenses and prisms.** *Lenses* are as accessors for a particular subfield of a data structure. In its basic form, they are given by a pair of functions:

the `view` function that accesses the subfield; and the `update` function that overwrites its contents.

The basic definition of *lens* [10, 27, 28] has been generalized in many different directions. *Monadic lenses* [1], *lenses in a symmetric monoidal category* [37, §2.2], *linear lenses* [33, §4.8] or *achromatic lenses* [2, §5.2] are some of them. These generalizations were not meant to be mutually compatible. They use the monoidal structure in different ways and introduce monadic effects in different parts of the signature. Some were not presented as *optics*, and a profunctor representation for them was not considered. We present a unified description.

We provide two derivations of lenses as mixed optics that capture all of the variants mentioned before, together with new ones, and endow them with a unified profunctor representation (Theorem 4.14).

The first derivation is based on cartesian structure. It generalizes the original derivation [24] and captures *lenses in a symmetric monoidal category* (Definition 3.3) and *monadic lenses* (Definition 3.5). It is then refined to cover *achromatic lenses* and describe some new variants of optics missing in the literature. The second derivation slightly generalizes *linear lenses* [33, §4.8] and uses the closed structure instead.

3.1.1. *Lenses.* Most variants of lenses rely on a cartesian monoidal structure. Throughout this section, we take a cartesian closed category $(\mathcal{W}, \times, 1)$ as our base for enrichment. During the rest of the paper we will explicitly use $\mathcal{W}$ to refer to a cartesian monoidal base of enrichment and use $\mathcal{V}$ to refer to a not-necessarily-cartesian base of enrichment. A monoidal $\mathcal{W}$-category $(\mathbf{C}, \times, 1)$ is said to be *cartesian* if there exist $\mathcal{W}$-natural isomorphisms $\mathbf{C}(Z, X \times Y) \cong \mathbf{C}(Z, X) \times \mathbf{C}(Z, Y)$ and $\mathbf{C}(X, 1) \cong 1$.

Definition 3.1. Let $\mathbf{C}$ be a cartesian $\mathcal{W}$-category with a monoidal $\mathcal{W}$-action $(\bullet)\colon \mathbf{C} \times \mathbf{D} \to \mathbf{D}$ to an arbitrary $\mathcal{W}$-category $\mathbf{D}$. A *lens* is an element of

$$\mathbf{Lens}((A, B), (S, T)) \coloneqq \mathbf{C}(S, A) \times \mathbf{D}(S \bullet B, T).$$

Proposition 3.2. *Lenses are mixed optics (as in Definition 2.1) for the actions of the cartesian product* $(\times)\colon \mathbf{C} \times \mathbf{C} \to \mathbf{C}$ *and the given action* $(\bullet)\colon \mathbf{C} \times \mathbf{D} \to \mathbf{D}$*. That is,* $\mathbf{Lens} \cong \mathbf{Optic}_{(\times,\bullet)}$.

Proof. We apply the universal property of the product, which can be summarized as it being right adjoint to the diagonal functor $(\Delta)\colon \mathbf{C} \to \mathbf{C}^2$.

$$\begin{aligned} & \int^{C \in \mathbf{C}} \mathbf{C}(S, C \times A) \times \mathbf{D}(C \bullet B, T) \\ \cong \quad & \{\text{Adjunction } \Delta \dashv (\times)\} \\ & \int^{C \in \mathbf{C}} \mathbf{C}(S, C) \times \mathbf{C}(S, A) \times \mathbf{D}(C \bullet B, T) \\ \cong \quad & \{\text{Coyoneda}\} \\ & \mathbf{C}(S, A) \times \mathbf{D}(S \bullet B, T). \end{aligned}$$ □

This definition can be specialized to the pair $\mathbf{C}(S, A) \times \mathbf{C}(S \times B, T)$ if we take $\mathbf{C} = \mathbf{D}$ and we let $(\bullet)$ be the cartesian product. It is, however, more general than that, and it captures the following examples (Definitions 3.3 and 3.5).

3.1.2. *Lenses in a symmetric monoidal category.*

Definition 3.3 ([37, §2.2])**.** A **monoidal lens** in a symmetric monoidal category $\mathbf{C}$ is a *view and update* pair where the view is a commutative comonoid homomorphism,

$$\mathbf{MonLens}_{\otimes}((A,B),(S,T)) \coloneqq \mathbf{CCom}(S,A) \times \mathbf{C}(\mathcal{U}S \otimes B, T).$$

Here $\mathcal{U}$ represents the forgetful functor $\mathcal{U}\colon \mathbf{CCom} \to \mathbf{C}$.

Proposition 3.4. *Monoidal lenses in a symmetric monoidal category are a particular case of Definition 3.1.*

Proof. The category of cocommutative comonoids $\mathbf{CCom}$ over a category $\mathbf{C}$ can be given a cartesian structure in such a way that the forgetful functor $\mathcal{U}\colon \mathbf{CCom} \to \mathbf{C}$ is strict monoidal (a consequence of Fox's theorem [11]). By Proposition 3.2, we can show $\mathbf{MonLens}_{\otimes} \cong \mathbf{Optic}_{(\otimes,\bullet)}$ where $(\bullet)$ is given by $S \bullet A \coloneqq \mathcal{U}S \otimes A$ in this case. □

3.1.3. *Monadic lenses.*

Definition 3.5. Monadic lenses [1, §2.3] allow for monadic effects in the `update` function (an example is Figure 7). For $\Psi\colon \mathcal{W} \to \mathcal{W}$ a $\mathcal{W}$-monad,

$$\mathbf{MndLens}_{\Psi}((A,B),(S,T)) \coloneqq \mathcal{W}(S,A) \times \mathcal{W}(S \times B, \Psi T).$$

Proposition 3.6. *Monadic lenses are a particular case of Definition 3.1.*

Proof. Every $\mathcal{W}$-endofunctor is *strong*; thus, the $\mathcal{W}$-monad Ψ comes with a $\mathcal{W}$-natural family $\theta_{X,Y}\colon X \times \Psi(Y) \to \Psi(X \times Y)$. This induces a $\mathcal{W}$-action $(\rtimes)\colon \mathcal{W} \times \mathrm{Kl}(\Psi) \to \mathrm{Kl}(\Psi)$, with $\mathrm{Kl}(\Psi)$ the Kleisli category of the monad; defined, on morphisms, as the composite

$$\begin{aligned} \mathcal{W}(X,Y) \times \mathcal{W}(A,\Psi B) &\xrightarrow{(\times)} \mathcal{W}(X \times A, Y \times \Psi B) \\ &\xrightarrow{\theta} \mathcal{W}(X \times A, \Psi(Y \times B)). \end{aligned}$$

Using that $\mathrm{Kl}_{\Psi}(S \rtimes B, T) \coloneqq \mathcal{W}(S \times B, \Psi T)$, monadic lenses are lenses (as in Definition 3.1), $\mathbf{MndLens}_{\Psi} \cong \mathbf{Optic}_{(\times,\rtimes)}$, where $(\bullet)$ is given by $(\rtimes)\colon \mathcal{W} \times \mathrm{Kl}(\Psi) \to \mathrm{Kl}(\Psi)$. □

Remark 3.7. This technique is similar to the one used by Riley [33, §4.9] to describe a non-mixed variant called *effectful lenses*.

3.1.4. *Algebraic lenses.* We can further generalize Definition 3.1 if we allow the *context* over which we take the coend to be an algebra for a fixed monad. The motivation is that lenses with a context like this appear to have direct applications in programming; for instance, the *achromatic lenses* of [3] are a particular case of this definition. These *algebraic lenses* should not be confused with the previous *monadic lenses* in Definition 3.5.

```
 stamp :: MonadicLens IO a b (Timestamped a) (Timestamped b)
 stamp = mkMonadicLens @IO viewContents updateContentsAndStamp
   where
     viewContents :: Timestamped a -> a
     viewContents = contents'

     updateContentsAndStamp :: Timestamped a -> b -> IO (Timestamped b)
     updateContentsAndStamp x b = do
       currentTime <- getCurrentTime
       return (x {contents' = b , modified' = currentTime})

ultimateQuestion :: IO (Timestamped String)
ultimateQuestion = do
  t <- getCurrentTime
  x <- pure (Timestamped t t "What is the answer?")
  threadDelay (2500000) -- microseconds
  x & stamp .! "42"

 >> ultimateQuestion
 Contents: "42",
 Created:  2020-02-02 12:24:55.119075225 UTC,
 Modified: 2020-02-02 12:24:57.621826372 UTC.
```

FIGURE 7. A polymorphic family of type-changing monadic lenses for the IO monad is used to track how a data holder (Timestamped) is accessed 2.5 seconds after creation.

Definition 3.8. Let $\Psi\colon \mathbf{C} \to \mathbf{C}$ be a $\mathcal{W}$-monad in a cartesian $\mathcal{W}$-category $\mathbf{C}$. Let $(\bullet)\colon \mathbf{C}\times\mathbf{D} \to \mathbf{D}$ be a monoidal $\mathcal{W}$-action to an arbitrary $\mathcal{W}$-category $\mathbf{D}$. An **algebraic lens** is an element of

$$\mathbf{AlgLens}_{\Psi}((A,B),(S,T)) \coloneqq \mathbf{C}(S,A) \times \mathbf{D}(\Psi S \bullet B, T).$$

Proposition 3.9. *Algebraic lenses are mixed optics for the actions of the product by the carrier of an algebra* $({}_{\mathcal{U}}\times)\colon \mathrm{EM}_{\Psi}\times\mathbf{C} \to \mathbf{C}$ *and some given tensor* $({}_{\mathcal{U}}\bullet)\colon \mathrm{EM}_{\Psi}\times\mathbf{D} \to \mathbf{D}$*. That is,* $\mathbf{AlgLens}_{\Psi} \cong \mathbf{Optic}_{({}_{\mathcal{U}}\times, {}_{\mathcal{U}}\bullet)}$.

Proof. The $\mathcal{W}$-category of algebras is cartesian, making the forgetful functor $\mathcal{U}\colon \mathrm{EM}_{\Psi} \to \mathbf{C}$ monoidal; $\mathcal{U}C \times A$ defines a strong monoidal action. The forgetful $\mathcal{U}$ has a left adjoint $\mathcal{F}^{\Psi}$ such that $\mathcal{U}\circ\mathcal{F}^{\Psi} = \Psi$.

$$\begin{aligned}
&\int^{C\in\mathrm{EM}_{\Psi}} \mathbf{C}(S,\mathcal{U}C\times A)\times\mathbf{D}(\mathcal{U}C\bullet B,T)\\
\cong\quad&\{\text{Adjunction } (\times)\dashv\Delta\}\\
&\int^{C\in\mathrm{EM}_{\Psi}} \mathbf{C}(S,\mathcal{U}C)\times\mathbf{C}(S,A)\times\mathbf{D}(\mathcal{U}C\bullet B,T)\\
\cong\quad&\{\text{Adjunction } \mathcal{F}^{\Psi}\dashv\mathcal{U}\}\\
&\int^{C\in\mathrm{EM}_{\Psi}} \mathrm{EM}_{\Psi}(\mathcal{F}^{\Psi}S,C)\times\mathbf{C}(S,A)\times\mathbf{D}(\mathcal{U}C\bullet B,T)
\end{aligned}$$

$$\cong \quad \{\text{Coyoneda}\}$$
$$\mathbf{C}(S, A) \times \mathbf{D}(\Psi S \bullet B, T). \qquad \square$$

Remark 3.10. Algebraic lenses are a new optic. When $(\bullet)$ is the cartesian product, algebraic lenses are given by the usual `view` function that accesses the subfield, and a variant of the `update` function that takes the original source as a computation rather than a value.

Example 3.11. The algebraic lens for the list monad $\mathcal{L} \colon \mathcal{V} \to \mathcal{V}$ is a new kind of optic that we dub a *classifying lens*. This is given by two functions, the usual `view` function that accesses the focus, and a `classify` function that takes a list of examples together with some piece of data and produces a new example. A classifying lens can be trained with a dataset (Figure 8) to classify a new focus into a complete data structure. A learning algorithm (in this case, a naive version of *nearest neighbor*) defines an algebraic lens that can be used to classify foci into full data structures.

```
let iris =
  [ Iris Setosa 4.9 3.0 1.4 0.2
  , Iris Setosa 4.7 3.2 1.3 0.2
  , ...
  , Iris Virginica 5.9 3.0 5.1 1.8 ]

measure :: AlgebraicLens [] Measurements Flower
measure = mkAlgebraicLens @[] measurements learn
 where
  distance :: Measurements -> Measurements -> Float
  distance (Measurements a b c d) (Measurements x y z w) =
     (sqrt . sum . fmap (**2)) [a-x,b-y,c-z,d-w]

  learn :: [Flower] -> Measurements -> Flower
  learn l m = Flower m (species (minimumBy
    (compare `on` (distance m . measurements)) l))

>>> (iris !! 4) ^. measure
(5.0, 3.6, 1.4, 0.2)

>>> iris & measure .? Measurements 4.8 3.1 1.5 0.1
Iris Versicolor (4.8, 3.1, 1.5, 0.1)
```

FIGURE 8. Fisher's `iris` dataset [8], samplying lengths and widths of sepals and petals of three species of iris. A classifying lens (`measure`) is used both for accessing the measurements of a point in the `iris` dataset and to classify *new* measurements (not already in the dataset) into a species (`Versicolor`).

Remark 3.12. The algebraic lens for the *maybe monad* $\mathcal{M} \colon \mathcal{V} \to \mathcal{V}$ was studied in [2, §5.2] under the name **achromatic lens**. It is motivated by the fact that, sometimes in practice, lenses come naturally equipped with a `create` function [10, §3] along the usual `view` and `update`. For this implementation, we note that

$$\mathbf{C}(S,A) \times \mathbf{C}(\mathcal{M}S \times B, T) \cong \mathbf{C}(S,A) \times \mathbf{C}(S \times B, T) \times \mathbf{C}(B,T).$$

Remark 3.13. The name *achromatic lens* can also refer to a different proposal in [33, §4.10],

$$\mathbf{C}(S,[B,T]+1) \times \mathbf{C}(S,A) \times \mathbf{C}(B,T).$$

This is the optic for the action $\odot \colon \mathbf{C} \to [\mathbf{C},\mathbf{C}]$ defined by $M \odot A \coloneqq (M+1) \times A$. It is not exactly equivalent to the *achromatic lens* in [2], which is the optic for the action $(\times)\colon \mathcal{M}\text{-Alg} \to [\mathbf{C},\mathbf{C}]$ defined by $\mathcal{U}M \times A$. It can be implemented by `view` and `create` functions, this time together with a `maybeUpdate` that is allowed to fail.

3.1.5. *Lenses in a closed category.* Linear lenses are a different generalization of lenses which relies on a closed monoidal structure. Their advantage is that we do not need to require our enriching category to be cartesian anymore.

Definition 3.14 (33, §4.8)**.** Let $(\mathbf{D}, \otimes, [])$ be a right closed $\mathcal{V}$-category with a monoidal $\mathcal{V}$-action $(\bullet)\colon \mathbf{D} \otimes \mathbf{C} \to \mathbf{C}$ to an arbitrary $\mathcal{V}$-category $\mathbf{C}$. A **linear lens** is an element of

$$\mathbf{LinearLens}_{\otimes,\bullet}((A,B),(S,T)) \cong \mathbf{C}(S,[B,T] \bullet A).$$

Proposition 3.15. *Linear lenses are mixed optics (as in Definition 2.1) for the actions of the monoidal product* $(\otimes)\colon \mathbf{D} \times \mathbf{D} \to \mathbf{D}$ *and* $(\bullet)\colon \mathbf{D} \times \mathbf{C} \to \mathbf{C}$*. That is,* $\mathbf{LinearLens}_{\otimes,\bullet} \cong \mathbf{Optic}_{\bullet,\otimes}$

Proof. The monoidal product has a right adjoint given by the exponential.

$$\begin{aligned}
& \int^{D\in\mathbf{D}} \mathbf{C}(S, D \bullet A) \otimes \mathbf{D}(D \otimes B, T) \\
\cong\ & \{\text{Adjunction } (- \otimes B) \dashv [B,-]\} \\
& \int^{D\in\mathbf{D}} \mathbf{C}(S, D \bullet A) \otimes \mathbf{D}(D, [B,T]) \\
\cong\ & \{\text{Coyoneda}\} \\
& \mathbf{C}(S, [B,T] \bullet A).
\end{aligned}$$

□

3.1.6. *Prisms.* Prisms pattern-match on data structures and handle a possible failure to match. They are given by a `match` function that tries to access the matched structure and a `build` function that constructs an abstract type from one of its possible matchings. Prisms happen to be lenses in the opposite category. However, they can also be described as optics in the original category for a different pair of actions. We will provide a derivation of prisms that is dual to our derivation of lenses for a cartesian structure. This derivation specializes to the pair $\mathbf{C}(S, T+A) \times \mathbf{C}(B,T)$.

Definition 3.16. Let $\mathbf{D}$ be a cocartesian $\mathcal{W}$-category with a monoidal $\mathcal{W}$-action $(\bullet)\colon \mathbf{D} \times \mathbf{C} \to \mathbf{C}$ to an arbitrary category $\mathbf{C}$. A **prism** is an element of

$$\mathbf{Prism}((A,B),(S,T)) \coloneqq \mathbf{C}(S, T \bullet A) \times \mathbf{D}(B,T).$$

In other words, a prism from (S,T) to (A,B) is a lens from (T,S) to (B,A) in the opposite categories $\mathbf{D}^{op}$ and $\mathbf{C}^{op}$. However, they can also be seen as optics from (A,B) to (S,T).

Proposition 3.17. *Prisms are mixed optics (as in Definition 2.1) for the actions of the coproduct* $(+)\colon \mathbf{D}\times\mathbf{D}\to\mathbf{D}$ *and* $(\bullet)\colon \mathbf{C}\times\mathbf{D}\to\mathbf{D}$. *That is,* $\mathbf{Prism}\cong\mathbf{Optic}_{(\bullet,+)}$.

Proof. The coproduct $(+)\colon \mathbf{D}\times\mathbf{D}\to\mathbf{D}$ is left adjoint to the diagonal functor.

$$\begin{aligned}
&\int^{D\in\mathbf{D}} \mathbf{C}(S, D\bullet A)\times\mathbf{D}(D+B,T)\\
\cong\;& \{\text{Adjunction } (+)\dashv\Delta\}\\
&\int^{D\in\mathbf{D}} \mathbf{C}(S, D\bullet A)\times\mathbf{D}(D,T)\times\mathbf{D}(B,T)\\
\cong\;& \{\text{Coyoneda}\}\\
&\mathbf{C}(S,T\bullet A)\times\mathbf{D}(B,T).
\end{aligned}$$
□

Remark 3.18 (Prisms in a symmetric monoidal category). A prism in a symmetric monoidal category $\mathbf{C}$ is a match and build pair where the build is a monoid homomorphism,

$$\mathbf{MonPrism}((A,B),(S,T)) \coloneqq \mathbf{C}(S,\mathcal{U}T\otimes A)\times\mathbf{CMon}(B,T).$$

Remark 3.19 (Prisms with coalgebraic context). Let Θ be a $\mathcal{W}$-comonad in a cocartesian $\mathcal{W}$-category $\mathbf{D}$ and $(\bullet)\colon \mathbf{D}\times\mathbf{C}\to\mathbf{C}$ a monoidal $\mathcal{W}$-action. A **coalgebraic prism** is an element of

$$\mathbf{AlgPrism}_\Theta((A,B),(S,T)) \coloneqq \mathbf{C}(S,\Theta T\bullet A)\times\mathbf{D}(B,T).$$

The coalgebraic variant is given by a `cMatch` function, that captures the failure into a comonad, and the usual `build` function.

3.2. **Traversals.** Traversals extract the elements of a container into an ordered list, allowing us to iterate over the elements without altering the container (Figure 9). Traversals are constructed from a single `extract :: s -> ([a], [b] -> t)` function that both outputs the elements and takes a new list to update them. Usually, we require the length of the input to the function of type `[b] -> t` to be the same as the length of `[a]`. This restriction can be also encoded into the type when dependent types are available.

```
let places =
  [ "43 Adlington Rd, Wilmslow, United Kingdom"
  , "26 Westcott Rd, Princeton, USA"
  , "St James's Square, London, United Kingdom" ]

each :: Traversal a [a]
each = mkTraversal (\x -> (x , id))

>>> places & each.asAddress.street %~ uppercase
 [ "43 ADLINGTON RD, Wilmslow, United Kingdom"
 , "26 WESTCOTT RD, Princeton, USA"
 , "ST JAMES'S SQUARE, London, United Kingdom"
 ]
```

FIGURE 9. The composition of a traversal (`each`) with a prism (`asAddress`) and a lens (`street`) is used to parse a collection of strings and modify one of the resulting subfields.

Definition 3.20. Let $\mathbf{C}$ be a symmetric monoidal closed $\mathcal{V}$-category with countably infinite (and smaller) coproducts. A **traversal** is an element of

$$\mathbf{Traversal}((A,B),(S,T)) := \mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes [B^n, T]\right).$$

Remark 3.21. Let $(\mathbb{N},+)$ be the free strict monoidal $\mathcal{V}$-category on one object. Ends and coends indexed by $\mathbb{N}$ coincide with products and coproducts, respectively. Here $A^{(-)}\colon \mathbb{N} \to \mathbf{C}$ is the unique monoidal $\mathcal{V}$-functor sending the generator of $\mathbb{N}$ to $A \in \mathbf{C}$. Each functor $X\colon \mathbb{N} \to \mathbf{C}$ induces a *power series*

$$\mathrm{Pw}_X(A) = \int^{n\in\mathbb{N}} A^n \otimes X_n.$$

This defines an action $\mathrm{Pw}\colon [\mathbb{N},\mathbf{C}] \to [\mathbf{C},\mathbf{C}]$ sending the indexed family to its power series (see Remark 3.23). We propose a derivation of the *traversal* as the optic for power series.

Proposition 3.22. *Traversals are optics (as in Definition 2.1) for power series. That is,* $\mathbf{Traversal} \cong \mathbf{Optic}_{\mathrm{Pw},\mathrm{Pw}}$.

Proof. The derivation generalizes that of linear lenses (Definition 3.14).

$$\begin{aligned}
&\int^{X\in[\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes X_n\right) \otimes \mathbf{C}\left(\int^{n\in\mathbb{N}} B^n \otimes X_n, T\right)\\
\cong\quad & \{\text{Continuity}\}\\
&\int^{X\in[\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes X_n\right) \otimes \int_{n\in\mathbb{N}} \mathbf{C}\left(X_n \otimes B^n, T\right)\\
\cong\quad & \{\text{Adjunction } (-\otimes B^n) \dashv [B^n, -]\}\\
&\int^{X\in[\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes X_n\right) \otimes \int_{n\in\mathbb{N}} \mathbf{C}\left(X_n, [B^n, T]\right)\\
\cong\quad & \{\text{Natural transformation}\}\\
&\int^{X\in[\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes X_n\right) \otimes [\mathbb{N},\mathbf{C}]\left(X_{(-)}, [B^{(-)}, T]\right)\\
\cong\quad & \{\text{Coyoneda}\}\\
&\mathbf{C}\left(S, \int^{n\in\mathbb{N}} A^n \otimes [B^n, T]\right). \qquad \square
\end{aligned}$$

The derivation from the general definition of optic to the concrete description of lenses and prisms in functional programming was first described by [30] and [24], but finding a derivation of traversals like the one presented here, fitting the same elementary pattern as lenses or prisms, was left as an open problem. It should be noted, however, that derivations of the traversal as the optic for a certain kind of functor called **Traversables** (which should not be confused with traversals themselves) have been previously described by [3, 33]. For a derivation using Yoneda, [33] recalls a parameterised adjunction that has an equational proof in the work of [16]. These two derivations do not contradict each other: two different classes of functors can generate the same optic; if, for instance, the adjunction describing both of them gives rise to the same monad. This seems to be the case here: traversables are coalgebras for a comonad and power series are the cofree coalgebras for the same comonad [35, §4.2].

In the **Sets**-based case, the relation between traversable functors, applicative functors [23] and these power series functors has been studied by [17].

Remark 3.23. The $\mathcal{V}$-functor $\mathrm{Pw}\colon [\mathbb{N},\mathbf{C}] \to [\mathbf{C},\mathbf{C}]$ is actually a strong monoidal action thanks to the fact that two power series functors compose into a power series functor.

$$\begin{aligned}
&\int^{m\in\mathbb{N}} \left(\int^{n\in\mathbb{N}} A^n \otimes C_n\right)^m \otimes D_m\\
\cong\quad & \{\text{Product distributes over colimits}\}\\
&\int^{m}\int^{n_1,\dots,n_m} A^{n_1}\otimes\cdots\otimes A^{n_m}\otimes C_{n_1}\otimes\cdots\otimes C_{n_m}\otimes D_m\\
\cong\quad & \{\text{Rearranging terms}\}\\
&\int^{k\in\mathbb{N}} A^k \otimes \left(\sum_{n_1+\cdots+n_m=k} C_{n_1}\otimes\cdots\otimes C_{n_m}\otimes D_m\right).
\end{aligned}$$

We are then considering an implicit non-symmetric monoidal structure where the monoidal product $(\bullet)\colon [\mathbb{N}, \mathbf{C}] \otimes [\mathbb{N}, \mathbf{C}] \to [\mathbb{N}, \mathbf{C}]$ of C_n and D_n can be written as follows; and the relevant copowers exist because of $\mathbf{C}$ having an initial object.

$$\int^{m} \int^{n_1,\dots,n_m} \mathbb{N}(n_1 + \dots + n_m, k) \cdot C_{n_1} \otimes \dots \otimes C_{n_m} \otimes D_m.$$

This is precisely the structure described by Kelly [19, §8] for the study of non-symmetric operads. A similar monoidal structure is described there when we substitute $\mathbb{N}$ by $(\mathbb{P}, +)$, the $\mathcal{V}$*-category of permutations* defined as the free strict symmetric monoidal category on one object. The same derivation can be repeated with this new structure to obtain an optic similar to the traversal, with the difference that elements are not ordered explicitly, and given by

$$\mathbf{C}\left(S, \int^{n \in \mathbb{P}} A^n \otimes [B^n, T]\right).$$

3.2.1. *Affine traversals. Affine traversals* strictly generalize prisms and linear lenses in the non-mixed case allowing a *lens-like* accessing pattern to fail; they are used to give a concrete representation of the composition between a lens and a prism. An affine traversal is implemented by a single `access` function.

Definition 3.24. Let $\mathcal{W}$ be cartesian closed and let $\mathbf{C}$ be a symmetric monoidal closed $\mathcal{W}$-category that is also cocartesian. An **affine traversal** is an element of

$$\mathbf{Affine}_{\otimes}((A,B),(S,T)) := \mathbf{C}(S, T + A \otimes [B,T]).$$

Proposition 3.25. *Let* $(\mathbf{Aff}, \odot)$ *[33] be the monoidal structure in* $\mathbf{C}^2$ *representing composition of affine polynomials,*

$$(C, D) \odot (C', D') = (C + D \otimes C', D \otimes D').$$

Affine traversals are optics (as in Definition 3.1) for the action $(+\otimes)\colon \mathbf{Aff} \to [\mathbf{C}, \mathbf{C}]$ *that sends* $C, D \in \mathbf{C}$ *to the functor* $C + D \otimes (-)$*. That is,* $\mathbf{Affine}_{\otimes} \cong \mathbf{Optic}_{(+\otimes),(+\otimes)}$.

Proof. The action uses that the monoidal product, which is in this case a left adjoint, distributes over the coproduct.

$$\begin{aligned}
&\int^{C,D} \mathbf{C}(S, C + D \otimes A) \times \mathbf{C}(C + D \otimes B, T) \\
\cong\ & \{\text{Adjunction } (+) \dashv \Delta\} \\
&\int^{C,D} \mathbf{C}(S, C + D \otimes A) \times \mathbf{C}(C, T) \times \mathbf{C}(D \otimes B, T) \\
\cong\ & \{\text{Coyoneda}\} \quad \{\text{Adjunction } (- \otimes B) \dashv [B, -]\} \\
&\int^{D} \mathbf{C}(S, T + D \otimes A) \times \mathbf{C}(D, [B,T]) \\
\cong\ & \{\text{Coyoneda}\} \\
&\mathbf{C}(S, T + A \otimes [B,T]).
\end{aligned}$$

□

3.2.2. *Kaleidoscopes.* *Applicative* functors are commonly used in functional programming as they provide a convenient generalization to monads with better properties for composition; they form the $\mathcal{V}$-category **App** of monoids with respect to Day convolution $(*)\colon [\mathcal{V},\mathcal{V}]\times[\mathcal{V},\mathcal{V}] \to [\mathcal{V},\mathcal{V}]$, which is defined as

$$(F * G)(A) = \int^{X,Y\in\mathcal{V}} \mathcal{V}(X\otimes Y, A)\otimes FX \otimes GY.$$

Alternatively, they are lax monoidal $\mathcal{V}$-functors for the cartesian structure. It is natural to ask what is the optic associated with applicative functors. We know from a basic result in category theory [22, §VII, Theorem 2] that, as the category $[\mathcal{V},\mathcal{V}]$ has coproducts indexed by the natural numbers, and Day convolution distributes over them, the free applicative functor can be computed as the colimit $(I + F + F^{*2} + F^{*3} + \dots)$. Having characterized free applicative functors, computing their associated optic amounts to an application of coend calculus.

Definition 3.26. A **kaleidoscope** is an element of

$$\mathbf{Kaleidoscope}\left((A,B),(S,T)\right) := \prod_{n\in\mathbb{N}} \mathcal{V}\left([A^n,B],[S^n,T]\right).$$

Proposition 3.27. *Kaleidoscopes are optics for the action of applicative functors.*

Proof. Let $\mathcal{U}\colon \mathbf{App} \to [\mathcal{V},\mathcal{V}]$ be the forgetful functor from the category of applicatives.

$$\begin{aligned}
&\int^{F\in\mathbf{App}} \mathcal{V}(S,\mathcal{U}FA)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Yoneda}\}\\
&\int^{F\in\mathbf{App}} \mathcal{V}\left(S, \int_{C\in\mathcal{V}} [[A,C],\mathcal{U}FC]\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Continuity}\}\\
&\int^{F\in\mathbf{App}} \int_C \mathcal{V}\left(S,[[A,C],\mathcal{U}FC]\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Adjunction } ([A,C]\otimes -)\dashv [[A,C],-]\}\\
&\int^{F\in\mathbf{App}} \left(\int_C \mathcal{V}\left([A,C]\otimes S,\mathcal{U}FC\right)\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Natural transformation}\}\\
&\int^{F\in\mathbf{App}} [\mathcal{V},\mathcal{V}]\left([A,-]\otimes S,\mathcal{U}F\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Adjunction of free applicatives}\}\\
&\int^{F\in\mathbf{App}} \mathbf{App}\left(\sum_{n\in\mathbb{N}}[A^n,-]\otimes S^n, F\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ & \{\text{Coyoneda}\}
\end{aligned}$$

$$\begin{aligned} & \mathcal{V}\left(\sum_{n\in\mathbb{N}} S^n \otimes [A^n, B], T\right) \\ \cong \quad & \{\text{Continuity}\} \quad \{\text{Adjunction } (S^n \otimes -) \dashv [S^n, -]\} \\ & \prod_{n\in\mathbb{N}} \mathcal{V}([A^n, B], [S^n, T]) . \end{aligned}$$ □

Remark 3.28. The free applicative we construct here is known in Haskell programming as the `FunList` applicative [41]. In the same way that traversables are written in terms of lists, we can approximate Kaleidoscopes as a single function `aggregate :: ([a] -> b) -> ([s] -> t)` that takes a folding for the foci and outputs a folding for the complete data structure. Kaleidoscopes are a new optic, and we propose to use them as accessors for pointwise foldable data structures.

Kaleidoscopes pose a problem on the side of applications: they cannot be composed with lenses to produce new kaleidoscopes. This is because the constraint defining them (`Applicative`) is not a superclass of the constraint defining lenses: a functor given by a product is not applicative, in general. However, a functor given by a product *by a monoid* is applicative. This means that applicatives can be composed with lenses whose residual is a monoid, which are precisely the newly defined *classifying lenses* (Definition 3.8).

```
representative :: Kaleidoscope Float Measurements
representative = mkKaleidoscope aggregate
 where
  aggregate f l = Measurements
   (f (fmap sepalLe l)) (f (fmap sepalWi l))
   (f (fmap petalLe l)) (f (fmap petalWi l))

iris & measure.representative >- mean
 >>> Iris Versicolor; Sepal (5.843, 3.054); Petal (3.758, 1.198)
```

FIGURE 10. Following the previous Example 8, a kaleidoscope (`representative`) is composed with an algebraic lens to create a new point in the dataset by aggregating measurements with some function (`mean`, `maximum`) and then classifying it.

3.3. Grates.

Grates create a new structure when provided with a way of creating a new focus from a view function. They are given by a single `grate` function with the form of a nested continuation.

Definition 3.29. Let $\mathbf{C}$ be a symmetric closed $\mathcal{V}$-category (as in [7]). Let $(\ominus)\colon \mathbf{C}^{op} \times \mathbf{D} \to \mathbf{D}$ be an arbitrary action. A **grate** is an element of

$$\mathbf{Grate}\,((A,B),(S,T)) := \mathbf{D}([S,A] \ominus B, T).$$

Proposition 3.30. *Grates are mixed optics for the action of the exponential and* $(\ominus)\colon \mathbf{C}^{op} \times \mathbf{D} \to \mathbf{D}$*. That is,* $\mathbf{Grate} \cong \mathbf{Optic}_{[\ ,\ ],\ominus}$.

Proof. We know from [7, Proposition 1.2] that the adjunction $[-, A]^{op} \dashv [-, A]$ holds in any symmetric monoidal category.

$$\begin{aligned}
&\int^{C \in \mathbf{C}} \mathbf{C}(S, [C, A]) \otimes \mathbf{D}(C \ominus B, T) \\
\cong\ & \{\text{Adjunction } [-, A]^{op} \dashv [-, A]\} \\
&\int^{C \in \mathbf{C}} \mathbf{C}(C, [S, A]) \otimes \mathbf{D}(C \ominus B, T) \\
\cong\ & \{\text{Coyoneda}\} \\
&\mathbf{D}([S, A] \ominus B, T)
\end{aligned}$$

□

The description of a grate and its profunctor representation in terms of "closed" profunctors was first reported by Deikun and O'Connor [25]; it can be seen as a consequence of the profunctor representation theorem (Theorem 4.14).

3.3.1. *Glasses. Glasses* are a new optic that strictly generalizes grates and lenses for the cartesian case. In functional programming, glasses can be implemented by a single function `glass` that takes a way of transforming *views* into new foci and uses it to update the data structure. We propose to use them as a concrete representation of the composition of lenses and grates.

Definition 3.31. A **glass** in a cartesian closed $\mathcal{W}$-category is an element of

$$\mathbf{Glass}\,((A, B), (S, T)) := \mathbf{C}(S \times [[S, A], B], T).$$

Proposition 3.32. *Glasses are optics for the action* $(\times[\ ,\ ]) \colon \mathbf{C}^{op} \times \mathbf{C} \to [\mathbf{C}, \mathbf{C}]$ *that sends* $C, D \in \mathbf{C}$ *to the* $\mathcal{W}$*-functor* $D \times [C, -]$.

Proof.

$$\begin{aligned}
&\int^{C,D} \mathbf{C}(S, C \times [D, A]) \times \mathbf{C}([D, B] \times C, T) \\
\cong\ & \{\text{Adjunction } \Delta \dashv (\times)\} \\
&\int^{C,D} \mathbf{C}(S, C) \times \mathbf{C}(S, [D, A]) \times \mathbf{C}([D, B] \times C, T) \\
\cong\ & \{\text{Coyoneda}\} \\
&\int^{D} \mathbf{C}(S, [D, A]) \times \mathbf{C}([D, B] \times S, T) \\
\cong\ & \{\text{Adjunction } [-, A]^{op} \dashv [-, A]\} \\
&\int^{D} \mathbf{C}(D, [S, A]) \times \mathbf{C}([D, B] \times S, T) \\
\cong\ & \{\text{Coyoneda}\} \\
&\mathbf{C}([[S, A], B] \times S, T).
\end{aligned}$$

□

3.4. **Getters, reviews and folds.** Some constructions, such as plain morphisms, can be regarded as degenerate cases of optics. We will describe these constructions (*getters*, *reviews* and *folds* [20]) as mixed optics. All of them set one of the two base categories to be the terminal category and, contrary to most optics, they act only unidirectionally. We will derive the usual implementations of getters as `s -> a`, of reviews as `b -> t`, and of folds as `s -> [a]`. Their profunctor representation can be seen as a covariant or contravariant application of the Yoneda lemma.

Definition 3.33. Let $\mathbf{C}$ be an arbitrary $\mathcal{V}$-category. Getters (morphisms $\mathbf{C}(S, A)$) are degenerate optics for the trivial action on the covariant side. Reviews (morphisms $\mathbf{C}(B, T)$) are degenerate optics for the trivial action on the contravariant side. In other words, we can obtain plain morphisms as a particular case of optic when $\mathbf{M} = \mathbf{1}$ and $\mathbf{D} = \mathbf{1}$ or $\mathbf{C} = \mathbf{1}$, respectively.

The category of **foldable** functors is the slice category on the list functor $\mathcal{L}\colon \mathcal{V} \to \mathcal{V}$, also known as the free monoid functor. Using the fact that $\mathcal{L}$ is a monad, the slice category $[\mathcal{V}, \mathcal{V}]/\mathcal{L}$ can be made monoidal in such a way that the forgetful functor $[\mathcal{V}, \mathcal{V}]/\mathcal{L} \to [\mathcal{V}, \mathcal{V}]$ becomes strong monoidal.

Definition 3.34. Folds are optics for the action of foldable functors and the trivial action on the contravariant side.

$$\mathbf{Fold}((A, *), (S, *)) = \int^{F \in \mathbf{Foldable}} \mathcal{V}(S, FA) \cong \mathcal{V}(S, \mathcal{L}A).$$

Folds admit a concrete description, $\mathcal{V}(S, \mathcal{L}A)$, which can be reached from the definition of coends as colimits. A coend from a diagram category with a terminal object is determined by the value at the terminal object. The same technique can be used to prove that the optic for the slice category over a monad $\mathcal{G}\colon \mathcal{V} \to \mathcal{V}$ has a concrete form given by $\mathbf{C}(S, \mathcal{G}A)$.

3.5. **Setters and adapters.**

Definition 3.35. We finish our examples of optics with two extremes. A setter [20] is an element of

$$\mathbf{Setter}((A, B), (S, T)) := \mathcal{V}([A, B], [S, T]).$$

Proposition 3.36. *Setters are optics for identity action* $\mathrm{id}\colon [\mathcal{V}, \mathcal{V}] \to [\mathcal{V}, \mathcal{V}]$. *That is,* $\mathbf{Setter} \cong \mathbf{Optic}_{\mathrm{id},\mathrm{id}}$.

Proof. The concrete derivation is described by Riley [33, §4.5.2], and his strength requirement can be transferred directly to the enriched case.

$$\begin{aligned}
& \int^{F \in [\mathcal{V}, \mathcal{V}]} \mathcal{V}(S, FA) \otimes \mathcal{V}(FB, T) \\
\cong \quad & \{\text{Yoneda}\} \\
& \int^{F \in [\mathcal{V}, \mathcal{V}]} \left(\int_{C \in \mathcal{V}} [\mathcal{V}(A, C), \mathcal{V}(S, FC)] \right) \otimes \mathcal{V}(FB, T) \\
\cong \quad & \{\text{Copower}\} \\
& \int^{F \in [\mathcal{V}, \mathcal{V}]} \left(\int_{C \in \mathcal{V}} \mathcal{V}(S \otimes [A, C], FC) \right) \otimes \mathcal{V}(FB, T) \\
\cong \quad & \{\text{Natural transformation}\} \\
& \int^{F \in [\mathcal{V}, \mathcal{V}]} [\mathcal{V}, \mathcal{V}](S \otimes [A, -], F) \otimes \mathcal{V}(FB, T)
\end{aligned}$$

$$\begin{aligned}
&\cong \quad \{\text{Coyoneda}\} \\
&\quad \mathcal{V}(S \otimes [A,B], T) \\
&\cong \quad \{\text{Adjunction } (S \otimes -) \dashv [S,-]\} \\
&\quad \mathcal{V}([A,B],[S,T]).
\end{aligned}$$

□

Remark 3.37. In functional programming, we implicitly restrict to the case where $\mathcal{V}$ is a cartesian category, and we curry this description to obtain the usual representation of setters as a single `over` function.

Definition 3.38. Adapters [20] are morphisms in $\mathbf{C}^{op} \otimes \mathbf{D}$. They are optics for the action $\mathrm{Id}\colon \mathbf{1} \to [\mathbf{C},\mathbf{C}]$.

$$\mathbf{Adapter}((A,B),(S,T)) \coloneqq \mathbf{C}(S,A) \otimes \mathbf{D}(B,T).$$

3.6. **Optics for (co)free.** A common pattern that appears across many optic derivations is that of computing the optic associated with a class of functors using an adjunction to allow for an application of the Yoneda lemma. This observation can be generalized to a class of concrete optics.

Consider some $\mathcal{V}$-endofunctor $H\colon \mathcal{V} \to \mathcal{V}$. Any objects $A,B,S,T \in \mathcal{V}$ can be regarded as functors from the unit $\mathcal{V}$-category. The following isomorphisms are the consequence of the fact that left and right global Kan extensions are left and right adjoints to precomposition, respectively.

$$\mathcal{V}(S,HA) \cong [\mathcal{V},\mathcal{V}](\mathsf{Lan}_A S, H),$$
$$\mathcal{V}(HB,T) \cong [\mathcal{V},\mathcal{V}](H, \mathsf{Ran}_B T).$$

These extensions exist in $\mathcal{V}$ and they are given by the formulas

$$\mathsf{Lan}_A S \cong [A,-] \otimes S, \quad \mathsf{Ran}_B T \cong \big[[-,B],T\big].$$

Proposition 3.39 ([35, §3.4.7]). *Let the monoidal $\mathcal{V}$-action $\mathcal{U}\colon \mathbf{M} \to [\mathcal{V},\mathcal{V}]$ have a left adjoint $\mathcal{L}\colon [\mathcal{V},\mathcal{V}] \to \mathbf{M}$, or, dually, let it have a right adjoint $\mathcal{R}\colon [\mathcal{V},\mathcal{V}] \to \mathbf{M}$. In both of these cases the optic determined by that monoidal action has a concrete form, given by*

$$\mathcal{V}(\mathcal{U}\mathcal{L}([A,-] \otimes S)(B), T) \quad \text{or} \quad \mathcal{V}(S, \mathcal{U}\mathcal{R}\big[[-,B],T\big](A)),$$

respectively.

Proof. We prove the first case. The second one is dual.

$$\begin{aligned}
&\int^{M \in \mathbf{M}} \mathcal{V}(S, \mathcal{U}MA) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong &\quad \{\text{Kan extension}\} \\
&\int^{M \in \mathbf{M}} [\mathcal{V},\mathcal{V}]\,(\mathsf{Lan}_A S, \mathcal{U}M) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong &\quad \{\text{Adjunction } \mathcal{L} \dashv \mathcal{U}\} \\
&\int^{M \in \mathbf{M}} \mathbf{M}\,(\mathcal{L}\mathsf{Lan}_A S, M) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong &\quad \{\text{Coyoneda}\} \\
&\mathcal{V}\,(\mathcal{U}\mathcal{L}\mathsf{Lan}_A S(B), T)\,.
\end{aligned}$$

□

4. Tambara theory

A fundamental feature of optics is that they can be represented as a single polymorphic function. Optics in this form are called *profunctor optics* and we say this function is their *profunctor representation*. Profunctor optics can be easily composed, even if they are from different families: composition of optics as polymorphic functions becomes ordinary function composition. Figure 3 shows an example of this phenomenon. Profunctor optics are functions polymorphic over profunctors endowed with some extra algebraic structure. This extra structure depends on the family of the optic they represent. For instance, *lenses* are represented by functions polymorphic over *cartesian* profunctors, while *prisms* are represented by functions polymorphic over *cocartesian* profunctors [30, §3]. Milewski [24] notes that the algebraic structures accompanying these profunctors are precisely *Tambara modules*, a particular kind of profunctor that has been used to characterize the monoidal centre of convolution monoidal categories [39]. Because of this correspondence, categories of lenses or prisms can be obtained as particular cases of the "Doubles for monoidal categories" construction defined by Pastro and Street [29, §6]. The *double*[1] of an arbitrary monoidal $\mathcal{V}$-category $(\mathcal{A}, \otimes, I)$, is a promonoidal[2] $\mathcal{V}$-category $\mathcal{DA}$ whose hom-objects are defined as

$$\mathcal{DA}((A, B), (S, T)) \coloneqq \int^{C \in \mathcal{A}} \mathcal{A}(S, C \otimes A) \otimes \mathcal{A}(C \otimes B, T).$$

In the particular case where $\mathcal{A}$ is cartesian or cocartesian, the $\mathcal{V}$-category $\mathcal{DA}$ is precisely the category of lenses or prisms over $\mathcal{A}$, respectively. Moreover, one of the main results of [29, Proposition 6.1] declares that the $\mathcal{V}$-category $[\mathcal{D}\mathbf{C}, \mathcal{V}]$ of copresheaves over these $\mathcal{V}$-categories is equivalent to the $\mathcal{V}$-category of Tambara modules on $\mathbf{C}$. In the case of lenses or prisms, these Tambara modules are precisely cartesian and cocartesian profunctors, and this correspondence justifies their profunctor representation.

We will see how the results of Pastro and Street can be directly applied to the theory of optics, although they are not general enough for our purposes. Milewski [24] already proposed a unified description of optics, later extended by [3] and [33], that requires a generalization of the original result by Pastro and Street from monoidal products to arbitrary monoidal actions. In order to capture $\mathcal{V}$-enriched mixed optics, we need to go even further and generalize the definition of Tambara module in two directions. The monoidal category $\mathcal{A}$ in their definition needs to be substituted by a pair of arbitrary categories $\mathbf{C}$ and $\mathbf{D}$, and the monoidal product $\otimes \colon \mathcal{A} \to [\mathcal{A}, \mathcal{A}]$ needs to be substituted by a pair of arbitrary monoidal actions $(Ⓛ) \colon \mathbf{M} \otimes \mathbf{C} \to \mathbf{C}$ and $(Ⓡ) \colon \mathbf{M} \otimes \mathbf{D} \to \mathbf{D}$, from a common monoidal category $\mathbf{M}$.

[1]Double, as used by Pastro and Street [29], should not be confused with *double* in the sense of *double category*.

[2]A promonoidal category $\mathcal{A}$ is a generalization of a monoidal category with functors replaced by profunctors. For instance, the tensor product is a profunctor $P \colon \mathcal{A}^{op} \otimes \mathcal{A} \otimes \mathcal{A} \to \mathcal{V}$ and the unit is a presheaf $J \colon \mathcal{A}^{op} \to \mathcal{V}$.

This section can be seen both as a partial exposition of one of the main results of the work of Pastro and Street [29, Proposition 6.1] and a generalization of their definitions and propositions to the setting that is most useful for applications in functional programming.

4.1. **Generalized Tambara modules.**

Originally, *Tambara modules* [39] were conceived of as a structure on top of certain profunctors that made them play nicely with some monoidal action in both their covariant and contravariant components.

For our purposes, Tambara modules represent the different ways in which we can use an optic. If a profunctor P has Tambara module structure for the monoidal actions defining an optic, we can use that optic to lift the profunctor applied to the foci, $P(A, B)$, to the full structures, $P(S, T)$. For instance, the profunctor $(-) \times B \to (-)$ can be used to lift the projection $A \times B \to B$ into the `update` function $S \times B \to T$. In other words, this profunctor is a Tambara module compatible with all the families of optics that admit an `update` function, such as *lenses*. In programming libraries, that profunctor can be used to define a polymorphic `update` combinator that works across different families of optics.

Formally, we want to prove that Tambara modules for the actions Ⓛ and Ⓡ are copresheaves over the category $\mathbf{Optic}_{Ⓛ,Ⓡ}$. This will also justify the profunctor representation of optics in terms of Tambara modules (Theorem 4.14).

Definition 4.1. Let $(\mathbf{M}, \otimes, I)$ be a monoidal $\mathcal{V}$-category with two monoidal actions $(Ⓛ)\colon \mathbf{M} \otimes \mathbf{C} \to \mathbf{C}$ and $(Ⓡ)\colon \mathbf{M} \otimes \mathbf{D} \to \mathbf{D}$. A generalized **Tambara module** consists of a $\mathcal{V}$-profunctor $P\colon \mathbf{C}^{op} \otimes \mathbf{D} \to \mathcal{V}$ endowed with a family of morphisms

$$\alpha_{M,A,B}\colon P(A, B) \to P(M Ⓛ A, M Ⓡ B)$$

$\mathcal{V}$-natural in $A \in \mathbf{C}$ and $B \in \mathbf{D}$ and $\mathcal{V}$-dinatural in $M \in \mathbf{M}$, which additionally satisfies the two equations

$$\alpha_{I,A,B} \circ P(\phi_A, \varphi_B^{-1}) = \mathrm{id},$$
$$\alpha_{M\otimes N,A,B} \circ P(\phi_{M,N,A}, \varphi^{-1}_{M,N,B}) = \alpha_{M,NⓁA,NⓇB} \circ \alpha_{N,A,B},$$

for every $M, N \in \mathbf{M}$, every $A \in \mathbf{C}$ and every $B \in \mathbf{D}$. In other words, a family of morphisms making the following two diagrams commute.

$$\begin{tikzcd}
P(A,B) \arrow[r, "\alpha_{M\otimes N,A,B}"] \arrow[d, "\alpha_{N,A,B}"'] & P((M\otimes N) Ⓛ A, (M\otimes N) Ⓡ B) \arrow[d, "P(\phi_{M,N,A}{,}\varphi^{-1}_{M,N,B})"] \\
P(N Ⓛ A, N Ⓡ B) \arrow[r, "\alpha_{M,NⓁA,NⓇB}"'] & P(M Ⓛ N Ⓛ A, M Ⓡ N Ⓡ B)
\end{tikzcd}$$

$$\begin{tikzcd}
P(A,B) \arrow[r, "\alpha_{I,A,B}"] \arrow[dr, "\mathrm{id}"'] & P(I Ⓛ A, I Ⓡ B) \arrow[d, "P(\phi_A{,}\varphi_B^{-1})"] \\
& P(A,B)
\end{tikzcd}$$

When $\mathcal{V} = \mathbf{Sets}$, we can define a morphism of Tambara modules as a natural transformation $\eta_{A,B} \colon P(A,B) \to Q(A,B)$ satisfying

$$\eta_{M \circledL A, M \circledR B} \circ \alpha_{M,A,B} = \alpha'_{M,A,B} \circ \eta_{A,B}.$$

For an arbitrary $\mathcal{V}$, Tambara modules are the objects of a $\mathcal{V}$-category **Tamb** whose hom-object from (P, α) to (Q, α') is computed as the intersection of the equalizers of

$$\begin{aligned} &\int_{X,Y} \mathcal{V}(P(X,Y), Q(X,Y)) \\ &\xrightarrow{\pi_{A,B}} \mathcal{V}(P(A,B), Q(A,B)) \\ &\xrightarrow{\mathcal{V}(\mathrm{id}, \alpha'_{A,B})} \mathcal{V}(P(A,B), Q(M \circledL A, M \circledR B)) \end{aligned}$$

and

$$\begin{aligned} &\int_{X,Y} \mathcal{V}(P(X,Y), Q(X,Y)) \\ &\xrightarrow{\pi_{M \circledL A, M \circledR B}} \mathcal{V}(P(M \circledL A, M \circledR B), Q(M \circledL A, M \circledR B)) \\ &\xrightarrow{\mathcal{V}(\alpha_{A,B}, \mathrm{id})} \mathcal{V}(P(A,B), Q(M \circledL A, M \circledR B)) \end{aligned}$$

for each $A \in \mathbf{C}$, $B \in \mathbf{D}$ and $M \in \mathbf{M}$ [29, §3.2].

Remark 4.2. Pastro and Street [29] follow the convention of omitting the unitors and the associators of the monoidal category when defining Tambara modules. These appear in Definition 4.1 replaced by the structure isomorphisms of the two monoidal actions.

4.2. **Pastro-Street's "double" comonad.** Tambara modules are coalgebras for a particular comonad [29, §5]. That comonad has a left adjoint that must therefore be a monad, and then Tambara modules can be equivalently described as algebras for that monad. We will describe the $\mathcal{V}$-category of *generalized* Tambara modules **Tamb** as an Eilenberg-Moore category first for a comonad and then for its left adjoint monad. This will be the main lemma (Lemma 4.6) towards the profunctor representation theorem (Theorem 4.14).

Definition 4.3. We start by constructing the underlying $\mathcal{V}$-functor of the comonad $\Theta \colon \mathbf{Prof}(\mathbf{C}, \mathbf{D}) \to \mathbf{Prof}(\mathbf{C}, \mathbf{D})$. Consider first the $\mathcal{V}$-functor

$$T \colon \mathbf{M}^{op} \otimes \mathbf{M} \otimes \mathbf{C}^{op} \otimes \mathbf{D} \otimes \mathbf{Prof}(\mathbf{C}, \mathbf{D}) \to \mathcal{V},$$

given by the composition of the actions $(\circledL)$ and $(\circledR)$ with the evaluation $\mathcal{V}$-functor $\mathbf{C}^{op} \otimes \mathbf{D} \otimes \mathbf{Prof}(\mathbf{C}, \mathbf{D}) \to \mathcal{V}$. On objects, this is given by

$$T(M, N, A, B, P) \coloneqq P(M \circledL A, N \circledR B).$$

By the universal property of the end, this induces a $\mathcal{V}$-functor $\mathbf{C}^{op} \otimes \mathbf{D} \otimes \mathbf{Prof}(\mathbf{C}, \mathbf{D}) \to \mathcal{V}$ given by

$$T(A, B, P) = \int_{M \in \mathbf{M}} P(M \circledL A, M \circledR B),$$

which can be curried into the $\mathcal{V}$-functor

$$\Theta P(A,B) \coloneqq \int_{M\in\mathbf{M}} P(M \textcircled{L} A, M \textcircled{R} B).$$

Proposition 4.4. *The $\mathcal{V}$-functor Θ can be endowed with a comonad structure. Its counit is the $\mathcal{V}$-natural family of morphisms $\varepsilon_{P,A,B}\colon \Theta P(A,B) \to P(A,B)$ obtained by projecting on the monoidal unit and applying the unitors,*

$$\Theta P(A,B) \xrightarrow{\pi_I} P(I \textcircled{L} A, I \textcircled{R} B) \xrightarrow{P(\phi_A,\varphi_B^{-1})} P(A,B).$$

Its comultiplication is given by $\delta_{P,A,B}\colon \Theta P(A,B) \to \Theta\Theta P(A,B)$, the $\mathcal{V}$-natural family of transformations obtained as the unique morphisms factorizing

$$\Theta P(A,B) \xrightarrow{P\left(\phi_{M,N,A},\varphi_{M,N,B}^{-1}\right)\circ\pi_{M\otimes N}} P(M \textcircled{L} N \textcircled{L} A, M \textcircled{R} N \textcircled{R} B)$$

through the projection

$$\Theta\Theta P(A,B) \xrightarrow{\pi_M\circ\pi_N} P(M \textcircled{L} N \textcircled{L} A, M \textcircled{R} N \textcircled{R} B)$$

for every $M,N \in \mathbf{M}$.

Remark 4.5. Before the proof, let us recall the axioms for a strong monoidal $\mathcal{V}$-action $\oslash\colon \mathbf{M}\otimes\mathbf{C} \to \mathbf{C}$ of a monoidal $\mathcal{V}$-category $(\mathbf{M},\otimes,I)$ with coherence isomorphisms (a,λ,ρ) to an arbitrary category $\mathbf{C}$. Let

$$\phi_A\colon A \cong I\oslash A, \quad \phi_{M,N,A}\colon M\oslash(N\oslash A) \cong (M\otimes N)\oslash A,$$

be the structure $\mathcal{V}$-natural isomorphisms of the strong monoidal action. Note that the following are precisely the axioms for a strong monoidal functor $\mathbf{M} \to [\mathbf{C},\mathbf{C}]$ written as $\mathbf{M}\otimes\mathbf{C} \to \mathbf{C}$; they are simplified by the fact that $[\mathbf{C},\mathbf{C}]$ is strict.

$$\begin{tikzcd}
I\oslash M\oslash A \arrow[d, "\phi_{I,M,A}"'] & M\oslash A \arrow[l, "\phi_{M\oslash A}"'] \arrow[r, "M\oslash\phi_A"] \arrow[d, "\mathrm{id}"] & M\oslash I\oslash A \arrow[d, "\phi_{M,I,A}"] \\
(I\otimes M)\oslash A \arrow[r, "\lambda_M\oslash A"'] & M\oslash A & (M\otimes I)\oslash A \arrow[l, "\rho_M\oslash A"]
\end{tikzcd}$$

$$\begin{tikzcd}
M\oslash N\oslash K\oslash A \arrow[r, "\mathrm{id}"] \arrow[d, "\phi_{M,N,K\oslash A}"'] & M\oslash N\oslash K\oslash A \arrow[d, "M\oslash\phi_{N,K,A}"] \\
(M\otimes N)\oslash K\oslash A \arrow[d, "\phi_{M\otimes N,K,A}"'] & M\oslash(N\otimes K)\oslash A \arrow[d, "\phi_{M,N\otimes K,A}"] \\
((M\otimes N)\otimes K)\oslash A \arrow[r, "a_{M,N,K}\oslash A"'] & (M\otimes(N\otimes K))\oslash A
\end{tikzcd}$$

Proof. In order to keep the diagrams in this proof manageable, we introduce the notation $P[M](A,B) \coloneqq P(M \textcircled{L} A, M \textcircled{R} B)$. We will show that this construction satisfies the comonad axioms.

We first prove left counitality, $\Theta\varepsilon_P \circ \delta_P = \mathrm{id}_{\Theta P}$, which follows from the commutativity of the following diagram.

$$\begin{tikzcd}
\Theta P(A,B) \arrow[r, "\delta_P"] \arrow[dd, "\pi_{I\otimes M}"'] & \Theta\Theta P(A,B) \arrow[r, "(\Theta\varepsilon)_P"] \arrow[d, "\pi_M"] & \Theta P(A,B) \arrow[d, "\pi_M"] \\
 & \Theta P[M](A,B) \arrow[r, "\varepsilon_{P[M]}"] \arrow[d, "\pi_I"] & P[M](A,B) \arrow[d, "\mathrm{id}"] \\
P[I\otimes M](A,B) \arrow[r, "P(\phi_{I,M,A},\varphi^{-1}_{I,M,B})"'] & P[M][I](A,B) \arrow[r, "P(\phi_{M\text{Ⓛ}A},\varphi^{-1}_{M\text{Ⓡ}B})"'] & P[M](A,B)
\end{tikzcd}$$

The left pentagon of this diagram commutes because of the definition of δ. The upper right square commutes because of functoriality of ends and naturality of π_M. The lower right square commutes because of the definition of ε. By the axioms of the monoidal actions (Remark 4.5), the bottom edge of the whole diagram can be rewritten as

$$P(\phi_{M\text{Ⓛ}A},\varphi^{-1}_{M\text{Ⓡ}B}) \circ P(\phi_{I,M,A},\varphi^{-1}_{I,M,B}) = P(\lambda^{-1}_M \oslash A, \lambda_M \oslash B).$$

Now, by the wedge condition of the end, the left-bottom side of the previous diagram is just the projection π_M.

$$\begin{tikzcd}
 & \Theta P(A,B) \arrow[dl, "\pi_{I\otimes M}"'] \arrow[dr, "\pi_M"] & \\
P[I\otimes M](A,B) \arrow[dr, "P(\mathrm{id},\lambda_M\oslash B)"'] & & P[M](A,B) \arrow[dl, "P(\lambda_M\oslash A,\mathrm{id})"] \\
 & P(I\otimes M \text{Ⓛ} A, M \text{Ⓡ} B) &
\end{tikzcd}$$

Finally, by the universal property of the end, that implies that $\Theta\varepsilon_P \circ \delta_P$ must coincide with the identity.

Let us now prove right counitality, $\varepsilon_{\Theta P} \circ \delta_P = \mathrm{id}_{\Theta P}$, which follows from the commutativity of the following diagram.

$$\begin{tikzcd}
\Theta P(A,B) \arrow[r, "\delta_P"] \arrow[dd, "\pi_{I\otimes M}"'] & \Theta\Theta P(A,B) \arrow[r, "\varepsilon_{\Theta P}"] \arrow[d, "\pi_I"] & \Theta P(A,B) \arrow[d, "\mathrm{id}"] \\
 & \Theta P[I](A,B) \arrow[r, "\Theta P(\phi_I,\varphi_I^{-1})"] \arrow[d, "\pi_M"] & \Theta P(A,B) \arrow[d, "\pi_M"] \\
P[M\otimes I](A,B) \arrow[r, "P(\phi_{M,I,A},\varphi^{-1}_{M,I,B})"'] & P[I][M](A,B) \arrow[r, "P(\phi_A,\varphi_B^{-1})"'] & P[M](A,B)
\end{tikzcd}$$

Again, the definition of δ makes the left pentagon commute. The upper right square commutes now because of the definition of ε, whereas the lower right square commutes because of functoriality of ends and naturality of π. By the axioms of the monoidal actions (Remark 4.5), the bottom edge of the diagram can be rewritten as

$$P(\phi_A,\varphi_B^{-1}) \circ P(\phi_{M,I,A},\varphi^{-1}_{M,I,B}) = P(\rho^{-1}_M \oslash A, \rho_M \oslash B).$$

Now, by the wedge condition of the end, the left-bottom side of the previous diagram is just the projection π_M.

$$\begin{tikzcd}
 & \Theta P(A,B) \arrow[dl, "\pi_{I\otimes M}"'] \arrow[dr, "\pi_M"] & \\
P[I\otimes M](A,B) \arrow[dr, "P(\mathrm{id},\lambda_M \oslash B)"'] & & P[M](A,B) \arrow[dl, "P(\lambda_M \oslash A,\mathrm{id})"] \\
 & P(I\otimes M \mathbin{\text{Ⓛ}} A, M \mathbin{\text{Ⓡ}} B) &
\end{tikzcd}$$

Finally, by the universal property of the end, $\varepsilon_{\Theta P} \circ \delta_P$ must coincide with the identity.

Coassociativity, $\Theta\delta_P \circ \delta_P = \delta_{\Theta P} \circ \delta_P$, follows from commutativity of the following diagram in Figure 11.

$$\begin{tikzcd}
 & \Theta P(A,B) \arrow[dl, "\delta_P"'] \arrow[dr, "\delta_P"] & \\
\Theta\Theta P(A,B) \arrow[dr, "\Theta\delta_P"'] \arrow[ddd, "\pi_K"'] & & \Theta\Theta P(A,B) \arrow[dl, "\delta_{\Theta P}"] \arrow[ddd, "\pi_{N\otimes K}"] \\
 & \Theta\Theta\Theta P(A,B) \arrow[d, "\pi_K"] & \\
 & \Theta\Theta P[K](A,B) \arrow[dd, "\pi_N"] & \\
\Theta P[K](A,B) \arrow[ur, "\delta_{P[K]}"] \arrow[ddd, "\pi_{M\otimes N}"'] & & \Theta P[N\otimes K](A,B) \arrow[dl, "\Theta P(\phi_{N,K,A},\varphi^{-1}_{N,K,B})"] \arrow[ddd, "\pi_M"] \\
 & \Theta P[K][N](A,B) \arrow[d, "\pi_M"] & \\
 & P[K][N][M](A,B) & \\
P[K][M\otimes N](A,B) \arrow[ur, "P(\phi_{M,N,K \text{Ⓛ} A},\varphi^{-1}_{M,N,K \text{Ⓡ} B})"] \arrow[d, "P(\phi_{M\otimes N,K,A},\varphi^{-1}_{M\otimes N,K,B})"] & & P[N\otimes K][M](A,B) \arrow[ul, "P(\phi_{N,K,A},\varphi^{-1}_{N,K,B})"'] \arrow[d, "P(\phi_{M,N\otimes K,A},\varphi^{-1}_{M,N\otimes K,B})"'] \\
P[(M\otimes N)\otimes K](A,B) \arrow[rr, "P(a^{-1}_{M,N,K},a_{M,N,K})"] & & P[M\otimes(N\otimes K)](A,B)
\end{tikzcd}$$

Figure 11. Diagram for the coassociativity axiom.

We need to show that the upper diamond commutes; by the universal property of the ends, this amounts to showing that it commutes when followed by $\pi_M \circ \pi_N \circ \pi_K$. The lower pentagon is made of isomorphisms, and it commutes by the axioms of the monoidal actions (Remark 4.5). The two upper degenerate pentagons commute by definition of δ. The two trapezoids commute by functoriality of ends and naturality of the projections.

Finally, the two outermost morphisms of the diagram correspond to two projections from $\Theta P(A,B)$, namely $\pi_{(M\otimes N)\otimes K}$ and $\pi_{M\otimes(N\otimes K)}$. The wedge condition for the associator $a_{M,N,K}$ makes the external diagram commute. □

Lemma 4.6. *Tambara modules are precisely coalgebras for this comonad. There exists an isomorphism of $\mathcal{V}$-categories* $\mathbf{Tamb} \cong EM(\Theta)$ *from the category of Tambara modules to the Eilenberg-Moore category of* Θ.

Proof. Note that the object of $\mathcal{V}$-natural transformations from P to ΘP is precisely

$$\begin{aligned}
&\mathbf{Prof}(\mathbf{C},\mathbf{D})(P,\Theta P)\\
\cong\ &\{\text{Natural transformation}\}\\
&\int_{A,B} \mathcal{V}\left(P(A,B), \int_{M\in\mathbf{M}} P(M \circledL A, M \circledR B)\right)\\
\cong\ &\{\text{Continuity}\}\\
&\int_{A,B,M} \mathcal{V}\left(P(A,B), P(M \circledL A, M \circledR B)\right)
\end{aligned}$$

whose elements can be seen as a family of morphisms that is natural in both $M \in \mathbf{M}$ and $(A,B) \in \mathbf{C}^{op} \otimes \mathbf{D}$. The two conditions in the definition of Tambara module can be rewritten as the axioms of the coalgebra. □

Proposition 4.7. *The* Θ *comonad has a left $\mathcal{V}$-adjoint* Φ, *which must therefore be a monad. On objects, it is given by the following formula.*

$$\Phi Q(X,Y) = \int^{M,U,V} Q(U,V) \otimes \mathbf{C}(X, M \circledL U) \otimes \mathbf{D}(M \circledR V, Y).$$

That is, there exists a $\mathcal{V}$-natural isomorphism $\mathrm{Nat}(\Phi Q, P) \cong \mathrm{Nat}(Q, \Theta P)$.

Proof. We can explicitly construct the $\mathcal{V}$-natural isomorphism using coend calculus.

$$\begin{aligned}
&\int_{A,B} \mathcal{V}\left(Q(A,B), \int_{M} P(M \circledL A, M \circledR B)\right)\\
\cong\ &\{\text{Continuity}\}\\
&\int_{M,A,B} \mathcal{V}(Q(A,B), P(M \circledL A, M \circledR B))\\
\cong\ &\{\text{Yoneda}\}\\
&\int_{M,A,B} \mathcal{V}\left(Q(A,B), \int_{X,Y} \mathcal{V}\Big(\mathbf{C}(X, M \circledL A) \otimes \mathbf{D}(M \circledR B, Y), P(X,Y)\Big)\right)\\
\cong\ &\{\text{Continuity}\}\\
&\int_{M,A,B,X,Y} \mathcal{V}\left(Q(A,B), \mathcal{V}\Big(\mathbf{C}(X, M \circledL A) \otimes \mathbf{D}(M \circledR B, Y), P(X,Y)\Big)\right)\\
\cong\ &\{\text{Copower}\}\\
&\int_{M,A,B,X,Y} \mathcal{V}(Q(A,B) \otimes \mathbf{C}(X, M \circledL A) \otimes \mathbf{D}(M \circledR B, Y), P(X,Y))\\
\cong\ &\{\text{Continuity}\}
\end{aligned}$$

$$\int_{X,Y} \mathcal{V}\left(\int^{M,A,B} Q(A,B) \otimes \mathbf{C}(X, M ⓛ A) \otimes \mathbf{D}(M ⓡ B, Y), P(X,Y)\right).$$

Alternatively, the adjunction can be deduced from the definition of the comonad Θ and the characterization of global Kan extensions as adjoints to precomposition. □

4.3. **Pastro-Street's "double" promonad.** The second part of this proof occurs in the bicategory of $\mathcal{V}$-profunctors. In this bicategory, there exists a formal analogue of the Kleisli construction that, when applied to the Pastro-Street monad Φ, yields a category whose morphisms are the optics from Definition 2.1. This is the crucial step of the proof, as the universal property of that Kleisli construction will imply that copresheaves over the category of optics there defined are Tambara modules (Lemma 4.10). After that, the last step will be a relatively straightforward application of the Yoneda lemma (Lemma 4.12).

Let **Prof** be the bicategory of $\mathcal{V}$-profunctors that has as 0-cells the $\mathcal{V}$-categories $\mathbf{C}, \mathbf{D}, \mathbf{E}, \ldots$; as 1-cells $P\colon \mathbf{C} \nrightarrow \mathbf{D}$ the $\mathcal{V}$-profunctors given as $P\colon \mathbf{C}^{op} \otimes \mathbf{D} \to \mathcal{V}$; and as 2-cells the natural transformations between them. The composition of two $\mathcal{V}$-profunctors $P\colon \mathbf{C}^{op} \otimes \mathbf{D} \to \mathcal{V}$ and $Q\colon \mathbf{D}^{op} \otimes \mathbf{E} \to \mathcal{V}$ is the $\mathcal{V}$-profunctor $Q \diamond P\colon \mathbf{C}^{op} \otimes \mathbf{E} \to \mathcal{V}$ defined on objects by the coend[3]

$$(Q \diamond P)(C,E) = \int^{D \in \mathbf{D}} P(C,D) \otimes Q(D,E).$$

There is, however, an equivalent way of defining profunctor composition if we interpret each $\mathcal{V}$-profunctor $\mathbf{C}^{op} \otimes \mathbf{D} \to \mathcal{V}$ as a $\mathcal{V}$-functor $\mathbf{C}^{op} \to [\mathbf{D}, \mathcal{V}]$ to the category of copresheaves. In this case, the composition of two profunctors $P\colon \mathbf{C}^{op} \to [\mathbf{D}, \mathcal{V}]$ and $Q\colon \mathbf{D}^{op} \to [\mathbf{E}, \mathcal{V}]$ is the $\mathcal{V}$-functor $(Q \diamond P)\colon \mathbf{C}^{op} \to [\mathbf{E}, \mathcal{V}]$ defined by taking a left Kan extension $(Q \diamond P) \coloneqq \mathsf{Lan}_y Q \circ P$ along the Yoneda embedding $y\colon \mathbf{D}^{op} \to [\mathbf{D}, \mathcal{V}]$. The unit profunctor for composition is precisely the Yoneda embedding.

$$\begin{tikzcd}
 & \mathbf{D}^{op} \arrow[r, "Q"] \arrow[d, "y"'] & {[\mathbf{E}, \mathcal{V}]} \\
\mathbf{C}^{op} \arrow[r, "P"] & {[\mathbf{D}, \mathcal{V}]} \arrow[ur, "\mathsf{Lan}_y Q \circ P"'] &
\end{tikzcd}$$

In the same way that we can construct a Kleisli category over a monad, we will perform a Kleisli construction over the monoids of the bicategory **Prof**, which are called **promonads**. Promonads over the base category $\mathcal{V}$ that are also Tambara modules for the product appear frequently in the literature on functional programming languages under the name of *arrows* [14, 15, 34].

Definition 4.8. A **promonad** is given by a $\mathcal{V}$-category $\mathbf{A}$, an endoprofunctor $T\colon \mathbf{A}^{op} \otimes \mathbf{A} \to \mathcal{V}$, and two $\mathcal{V}$-natural families $\eta_{X,Y}\colon \mathbf{C}(X,Y) \to T(X,Y)$

[3]Although in general the composition of two profunctors can fail to exist for size reasons or when $\mathcal{V}$ lacks certain colimits, we only ever need these composites in a formal sense. This perspective can be formalized with the notion of *virtual equipment* [5].

and $\mu_{X,Y} \colon (T \diamond T)(X,Y) \to T(X,Y)$ satisfying the following unitality and associativity axioms.

$$\begin{tikzcd}
T \arrow[r, "\eta\diamond\mathrm{id}"] \arrow[dr, "\mathrm{id}"'] & T\diamond T \arrow[d, "\mu"] & T \arrow[l, "\mathrm{id}\diamond\eta"'] \arrow[dl, "\mathrm{id}"] \\
 & T &
\end{tikzcd}
\qquad
\begin{tikzcd}
T\diamond T\diamond T \arrow[r, "\mu\diamond\mathrm{id}"] \arrow[d, "\mathrm{id}\diamond\mu"'] & T\diamond T \arrow[d, "\mu"] \\
T\diamond T \arrow[r, "\mu"] & T
\end{tikzcd}$$

A module for the promonad is a $\mathcal{V}$-profunctor $P \colon \mathbf{X}^{op} \otimes \mathbf{A} \to \mathcal{V}$, together with a $\mathcal{V}$-natural transformation $\rho \colon T \diamond P \to P$ making the following diagrams commute.

$$\begin{tikzcd}
P \arrow[r, "\eta\diamond\mathrm{id}"] \arrow[dr, "\mathrm{id}"'] & T\diamond P \arrow[d, "\rho"] \\
 & P
\end{tikzcd}
\qquad
\begin{tikzcd}
T\diamond T\diamond P \arrow[r, "\mu\diamond\mathrm{id}"] \arrow[d, "\mathrm{id}\diamond\rho"'] & T\diamond P \arrow[d, "\rho"] \\
T\diamond P \arrow[r, "\rho"] & P
\end{tikzcd}$$

An algebra is a module structure on a $\mathcal{V}$-copresheaf $P \colon \mathbf{A} \to \mathcal{V}$, interpreted as a profunctor $\mathbf{I}^{op} \otimes \mathbf{A} \to \mathcal{V}$ from the unit $\mathcal{V}$-category.

Lemma 4.9. *The bicategory* **Prof** *admits the Kleisli construction [29, §6]. The Kleisli $\mathcal{V}$-category $\mathrm{Kl}(T)$ for a promonad (T, μ, η) over $\mathbf{A}$ is constructed as having the same objects as $\mathbf{A}$ and letting the hom-object between $X, Y \in \mathbf{A}$ be precisely $T(X,Y) \in \mathcal{V}$.*

Proof. The multiplication of the promonad is a $\mathcal{V}$-natural transformation whose components can be taken as the definition for the composition of the $\mathcal{V}$-category $\mathrm{Kl}(T)$.

$$\begin{aligned}
&\mathcal{V}\left(\int^{Z\in\mathbf{C}} T(X,Z)\otimes T(Z,Y), T(X,Y)\right) \\
\cong\ & \{\text{Continuity}\} \\
&\int_{Z\in\mathbf{C}} \mathcal{V}\left(T(X,Z)\otimes T(Z,Y), T(X,Y)\right)
\end{aligned}$$

Let us show now that this $\mathcal{V}$-category satisfies the universal property of the Kleisli construction. Let $P \colon \mathbf{X}^{op} \otimes \mathbf{A} \to \mathcal{V}$ be a $\mathcal{V}$-profunctor. A module structure $\rho \colon T \diamond P \to P$ corresponds to a way of making the profunctor P functorial over $\mathrm{Kl}(T)$ in the second argument

$$\begin{aligned}
&\int_{X\in\mathbf{X}, Z\in\mathbf{A}} \mathcal{V}\left(\int^{Y\in\mathbf{A}} P(X,Y)\otimes T(Y,Z), P(X,Z)\right) \\
\cong\ & \{\text{Continuity}\} \\
&\int_{X,Y,Z} \mathcal{V}(P(X,Y)\otimes T(Y,Z), P(X,Z)) \\
\cong\ & \{\text{Exponential}\} \\
&\int_{X,Y,Z} \mathcal{V}(T(Y,Z), [P(X,Y), P(X,Z)]).
\end{aligned}$$

Functoriality of this family follows from the monad-algebra axioms. □

Lemma 4.10. *The category of algebras for a promonad is equivalent to the copresheaf category over its Kleisli object.*

Proof. Let $\mathbf{X}$ be any category and $\Phi\colon \mathbf{Y} \to \mathbf{Y}$ a promonad. By the universal property of the Kleisli construction (see Lemma 4.9), $\mathbf{Prof}(\mathbf{X}, \mathrm{Kl}(\Phi))$ is equivalent to the category of modules on $\mathbf{X}$ for the promonad. In particular, $\mathcal{V}$-profunctors from the unit $\mathcal{V}$-category to the Kleisli object form precisely the category $\mathrm{EM}(\Phi)$ of algebras for the promonad; thus

$$[\mathrm{Kl}(\Phi), \mathcal{V}] \cong [\mathbf{I}^{op} \otimes \mathrm{Kl}(\Phi), \mathcal{V}] \cong \mathbf{Prof}(\mathbf{I}, \mathrm{Kl}(\Phi)) \cong \mathrm{EM}(\Phi). \qquad \square$$

Proposition 4.11. *Let $T\colon [\mathbf{A}, \mathcal{V}] \to [\mathbf{A}, \mathcal{V}]$ be a cocontinuous monad. The profunctor $\check{T}\colon \mathbf{A}^{op} \to [\mathbf{A}, \mathcal{V}]$ defined by $\check{T} \coloneqq T \circ y$ can be given a promonad structure. Moreover, algebras for T are precisely algebras for the promonad $\check{T}$.*

Proof. First, the fact that T is cocontinuous means that it preserves left Kan extensions and thus,

$$\mathsf{Lan}_y \check{T} \cong \mathsf{Lan}_y(T \circ y) \cong T \circ \mathsf{Lan}_y(y) \cong T.$$

This means that the composition of the profunctor $\check{T}$ with itself is

$$\check{T} \diamond \check{T} = \mathsf{Lan}_y \check{T} \circ \check{T} \cong \mathsf{Lan}_y \check{T} \circ T \circ y \cong T \circ T \circ y.$$

The unit and multiplication of the promonad are then obtained by whiskering the unit and multiplication of the monad with the Yoneda embedding; that is, $(\eta \circ y)\colon y \to T \circ y$ and $(\mu \circ y)\colon T \circ T \circ y \to T \circ y$. The diagrams for associativity and unitality for the promonad are the whiskering by the Yoneda embedding of the same diagrams for the monad. In fact, the same reasoning yields that, for any $P\colon \mathbf{D}^{op} \to [\mathbf{A}, \mathcal{V}]$,

$$\check{T} \diamond P \cong (T \circ y) \diamond P \cong \mathsf{Lan}_y(T \circ y) \circ P \cong T \circ P.$$

As a consequence of this for the case $P\colon \mathbf{I} \to [\mathbf{A}^{op}, \mathcal{V}]$, any T-algebra can be seen as a $\check{T}$-algebra and vice versa. The axioms for the promonad structure on $\check{T}$ coincide with the axioms for the corresponding monad on T. $\square$

In particular, the Pastro-Street monad Φ is a left adjoint. That implies that it is cocontinuous and, because of Proposition 4.11, it induces a promonad $\check{\Phi} = \Phi \circ y$, having Tambara modules as algebras. We can compute by the Yoneda lemma that

$$\check{\Phi}(A, B, S, T) = \int^{M} \mathbf{C}(S, M \, ⓛ \, A) \otimes \mathbf{D}(M \, ⓡ \, B, T).$$

This coincides with Definition 2.1. We now define **Optic** to be the Kleisli $\mathcal{V}$-category for the promonad $\check{\Phi}$.

4.4. **Profunctor representation theorem.** Let us zoom out to the big picture again. It has been observed that optics can be composed using their profunctor representation; that is, profunctor optics can be endowed with a natural categorical structure. On the other hand, we have generalized the double construction in [29] to abstractly obtain the category **Optic**. The final missing piece that makes both coincide is the *profunctor representation theorem*, which will justify the profunctor representation of optics and their composition in profunctor form being the usual function composition.

The profunctor representation theorem for the case $\mathcal{V} = \mathbf{Sets}$ and non-mixed optics has been discussed in [3, Theorem 4.2]. Although our statement is more general and the proof technique is different, the main idea is the same. In both cases, the key insight is the following lemma, already described by [24].

Lemma 4.12 ("Double Yoneda" from [24])**.** *For any $\mathcal{V}$-category $\mathbf{A}$, the hom-object between X and Y is $\mathcal{V}$-naturally isomorphic to the object of $\mathcal{V}$-natural transformations between the functors that evaluate copresheaves in X and Y; that is,*

$$\mathbf{A}(X,Y) \cong [[\mathbf{A},\mathcal{V}],\mathcal{V}](-(X),-(Y)).$$

The isomorphism is given by the canonical maps $\mathbf{A}(X,Y) \to \mathcal{V}(FX,FY)$ for each $F \in [\mathbf{A},\mathcal{V}]$. Its inverse is given by computing its value on the identity on the $\mathbf{A}(X,-)$ component.

Proof. In the functor $\mathcal{V}$-category $[\mathbf{A},\mathcal{V}]$, we can apply the Yoneda embedding to two representable functors $\mathbf{A}(Y,-)$ and $\mathbf{A}(X,-)$ to get

$$[\mathbf{A},\mathcal{V}](\mathbf{A}(Y,-),\mathbf{A}(X,-)) \cong \int_F \mathcal{V}\Big([\mathbf{A}(X,-),F],[\mathbf{A}(Y,-),F]\Big).$$

Here reducing by Yoneda lemma on both the left hand side and the two arguments of the right hand side, we get the desired result. □

Remark 4.13. As a very simple special case of the Double Yoneda construction, the Haskell type

```
forall f . Functor f => f a -> f b
```

is isomorphic to the simple function type `a -> b` [24]. It is straightforward for a functional programmer to construct the two witnesses to the isomorphism: the functorial action in one direction, and instantiation to the identity functor in the other.

Theorem 4.14 (Profunctor representation theorem)**.**

$$\mathbf{Optic}((A,B),(S,T)) \cong \int_{P\in\mathbf{Tamb}} \mathcal{V}(P(A,B),P(S,T)).$$

Proof. We apply Double Yoneda (Lemma 4.12) to the $\mathcal{V}$-category **Optic** and then use that copresheaves over it are precisely Tambara modules (Proposition 4.10). □

Remark 4.15. The immediate practical application of this theorem is to justify the following *profunctor representation* commonly employed in Haskell libraries.

```
Optic a b s t = forall p . Tambara p => p a b -> p s t
```

For all the optics we've been discussing in this paper, where the *Tambara* constraint is replaced by the class of profunctors preserving the appropriate monoidal action. For instance, the standard lens

```
type Lens a b s t = forall p . Cartesian p => p a b -> p s t
```

is defined by the class of profunctors preserving the action defined by the cartesian product.

```
class Profunctor p => Cartesian p where
  first'  :: p a b -> p (a, c) (b, c)
  second' :: p a b -> p (c, a) (c, b)
```

5. Haskell implementation

Let $\mathcal{V}$ be a cartesian closed category whose objects model the types of our programming language and whose points $1 \to X$ represent programs of type X. The following is an informal translation of the concepts of enriched category theory to a Haskell implementation where a single abstract definition of optic is used for a range of different examples. The code for this text can be compiled under GHC 8.6, using the libraries `split` and `delay`. It includes an implementation of optics and all the examples we have discussed (Figures 1, 7, 8 and 9).

The complete code can be found at

https://github.com/mroman42/vitrea

5.1. **Concepts of enriched category theory.**

Definition 5.1 (18, §1.2, see also 42)**.** A $\mathcal{V}$-category $\mathbf{C}$ consists of a set $\mathrm{Obj}(\mathbf{C})$ of objects, a hom-object $\mathbf{C}(A, B) \in \mathcal{V}$ for each pair of objects $A, B \in \mathrm{Obj}(\mathbf{C})$, a composition law $\mathbf{C}(A, B) \times \mathbf{C}(B, C) \to \mathbf{C}(A, C)$ for each triple of objects, and an identity element $1 \to \mathbf{C}(A, A)$ for each object; subject to the usual associativity and unit axioms.

```
class Category objc c where
  unit :: (objc x) => c x x
  comp :: (objc x) => c y z -> c x y -> c x z
```

Here, the objects for our category are selected from Haskell types by the constraint `objc`. Hom-objects are selected by the two-argument type constructor `c`.

Definition 5.2 (18, §1.2)**.** A $\mathcal{V}$-functor $F \colon \mathbf{C} \to \mathbf{D}$ consists of a function $\mathrm{Obj}(\mathbf{C}) \to \mathrm{Obj}(\mathbf{D})$ together with a map $\mathbf{C}(A, B) \to \mathbf{D}(FA, FB)$ for each pair of objects; subject to the usual compatibility with composition and units. $\mathcal{V}$-bifunctors and $\mathcal{V}$-profunctors can be defined analogously,

```
class ( Category objc c, Category objd d, Category obje e
      , forall x y . (objc x , objd y) => obje (f x y) )
      => Bifunctor objc c objd d obje e f where
  bimap :: ( objc x1, objc x2, objd y1, objd y2 )
        => c x1 x2 -> d y1 y2 -> e (f x1 y1) (f x2 y2)

class ( Category objc c, Category objd d )
      => Profunctor objc c objd d p where
  dimap :: (objc x1, objc x2, objd y1, objd y2)
        => c x2 x1 -> d y1 y2 -> p x1 y1 -> p x2 y2
```

Definition 5.3 (6)**.** A monoidal $\mathcal{V}$-category is a $\mathcal{V}$-category $\mathbf{M}$ together with a $\mathcal{V}$-functor $(\otimes) \colon \mathbf{M} \otimes \mathbf{M} \to \mathbf{M}$, an object $I \in \mathbf{M}$, and $\mathcal{V}$-natural isomorphisms $\alpha \colon (A \otimes B) \otimes C \cong A \otimes (B \otimes C)$, $\rho \colon A \otimes I \cong A$, and $\lambda \colon I \otimes A \cong A$, satisfying the usual coherence axioms for a monoidal category.

```
class ( Category obja a
      , Bifunctor obja a obja a obja a o
      , obja i )
      => MonoidalCategory obja a o i where
  alpha  :: (obja x, obja y, obja z)
         => a (x `o` (y `o` z)) ((x `o` y) `o` z)
  alphainv  :: (obja x, obja y, obja z)
            => a ((x `o` y) `o` z) (x `o` (y `o` z))
  lambda    :: (obja x) => a (x `o` i) x
  lambdainv :: (obja x) => a x (x `o` i)
  rho       :: (obja x) => a (i `o` x) x
  rhoinv    :: (obja x) => a x (i `o` x)
```

Definition 5.4. A strong monoidal $\mathcal{V}$-action $F\colon \mathbf{M}\otimes\mathbf{C} \to \mathbf{C}$ from a monoidal $\mathcal{V}$-category $\mathbf{M}$ to an arbitrary category $\mathbf{C}$ is a $\mathcal{V}$-functor together with two $\mathcal{V}$-natural isomorphisms $F(I,X) \cong X$ and $F(M,F(N,X)) \cong F((M\otimes N),X)$ satisfying associativity and unitality conditions.

```
class ( MonoidalCategory objm m o i
      , Bifunctor objm m objc c objc c f
      , Category objc c )
      => MonoidalAction objm m o i objc c f where
  unitor :: (objc x) => c (f i x) x
  unitorinv :: (objc x) => c x (f i x)
  multiplicator :: (objc x, objm p, objm q)
                => c (f p (f q x)) (f (p `o` q) x)
  multiplicatorinv :: (objc x, objm p, objm q)
                => c (f (p `o` q) x) (f p (f q x))
```

Definition 5.5. Definition 2.1 has now a direct interpretation in more generality. Note how the coend is modeled as an existential type in x using a GADT.

```
  data Optic objc c objd d objm m o i f g a b s t where
    Optic :: ( MonoidalAction objm m o i objc c f
             , MonoidalAction objm m o i objd d g
             , objc a, objc s , objd b, objd t , objm x )
          => c s (f x a) -> d (g x b) t
          -> Optic objc c objd d objm m o i f g a b s t
```

5.2. **Mixed profunctor optics.** We can implement Tambara modules (Definition 4.1) and profunctor optics using the profunctor representation theorem (Theorem 4.14).

```
class ( MonoidalAction objm m o i objc c f
      , MonoidalAction objm m o i objd d g
      , Profunctor objc c objd d p )
      => Tambara objc c objd d objm m o i f g p where
  tambara :: (objc x, objd y, objm w)
          => p x y -> p (f w x) (g w y)


type ProfOptic objc c objd d objm m o i f g a b s t = forall p .
  ( Tambara objc c objd d objm m o i f g p
  , MonoidalAction objm m o i objc c f
  , MonoidalAction objm m o i objd d g
  , objc a , objd b , objc s , objd t
  ) => p a b -> p s t
```

The isomorphism between existential and profunctor optics can be explicitly constructed from Lemma 4.12.

```
ex2prof :: forall objc c objd d objm m o i f g a b s t .
       Optic     objc c objd d objm m o i f g a b s t
    -> ProfOptic objc c objd d objm m o i f g a b s t
ex2prof (Optic l r) =
  dimap @objc @c @objd @d l r .
  tambara @objc @c @objd @d @objm @m @o @i

prof2ex :: forall objc c objd d objm m o i f g a b s t .
    ( MonoidalAction objm m o i objc c f
    , MonoidalAction objm m o i objd d g
    , objc a , objc s
    , objd b , objd t )
    => ProfOptic objc c objd d objm m o i f g a b s t
    -> Optic     objc c objd d objm m o i f g a b s t
prof2ex p = p (Optic
    (unitorinv @objm @m @o @i @objc @c @f)
    (unitor @objm @m @o @i @objd @d @g))
```

We used the `TypeApplications` language extension to explicitly pass type parameters to polymorphic functions.

5.3. **Combinators.** After constructing optics, an implementation should provide ways of using them. Many optics libraries, such as Kmett's *lens* [20], provide a vast range of combinators. Each of these combinators works on some group of optics that share a common feature. For instance, we could consider all the optics that implement a `view` function, and create a single combinator that lets us view the focus inside a family of optics.

This may seem, at first glance, difficult to model. We do not know, a priori, which of our optics will admit a given combinator. However, the fact that Tambara modules are copresheaves over optics suggests that we can use them to model ways of accessing optics; and in fact, we have found them to be very satisfactory to describe combinators in their full generality.

Remark 5.6. As an example, for any fixed A and B, consider the profunctor $P_{A,B}(S,T) := (S \to A)$. It can be seen as modelling the `view` combinator that some optics provide.

```
newtype Viewing a b s t = Viewing { getView :: s -> a }
instance Profunctor Any (->) Any (->) (Viewing a b) where
  dimap l _ (Viewing f) = Viewing (f . l)
```

If we want to apply this combinator to a particular optic, we need it to be a Tambara module for the actions describing the optic. For instance, we can show that it is a Tambara module for the cartesian product, taking $\mathbf{C} = \mathbf{D} = \mathbf{M}$; this means it can be used with *lenses* in the cartesian case. In other words, *lenses* can be used to `view` the focus.

```
instance Tambara Any (->) Any (->) Any (->) (,) ()
    (,) (,) (Viewing a b) where
  tambara (Viewing f) = Viewing (f . snd)
```

Optic combinators are usually provided as infix functions that play nicely with the composition operator. Specifically, they have "fixity and semantics such that subsequent field accesses can be performed with `Prelude..` [function composition]" [20].

```
 infixl 8 ^.
(^.) :: s -> (Viewing a b a b -> Viewing a b s t) -> a
(^.) s l = getView (l (Viewing id)) s
```

5.3.1. *Table of combinators.* The names of our combinators try to match, where possible, the names used by Kmett's lens library [20].

	Combinators.
`(^.) ::`	`s -> (Viewing a b a b -> Viewing a b s t) -> a` View a single target.

`(?.) ::`	`s -> (Previewing a b a b`
	`-> Previewing a b s t) -> Maybe a`
	Try to view a single target; this can possibly result in failure.

`(.~) ::`	`(Setting a b a b -> Setting a b s t) -> b -> s -> t` Replace a target with a given value.

`(%~) ::`	`(Replacing a b a b -> Replacing a b s t)`
	`-> (a -> b) -> (s -> t)`
	Replace a target by applying a function.

`(.?) ::`	`(Monad m) => (Classifying m a b a b`
	` -> Classifying m a b s t'') -> b -> m s -> t`
	Classifies the target into a complete instance.

`(>-) ::`	`(Aggregating a b a b -> Aggregating a b s t)`
	`-> ([a] -> b) -> [s] -> t`
	Aggregates the whole structure by aggregating the targets.

`(.!) ::`	`(Monad m)=> (Updating m a b a b`
	`-> Updating m a b s t)-> b -> s -> m t`
	Replaces the target, producing a monadic effect.

5.4. **Table of optics.** We can consider all of these optics in the case where some cartesian closed $\mathcal{W}$ is both the enriching category and the base for the optic. This case is of particular interest in functional programming.

Name	Description	Ref.
Adapter	`(s -> a) , (b -> t)`	3.38
Lens	`(s -> a) , (s -> b -> t)`	3.1
Algebraic lens	`(s -> a) , (m s -> b -> t)`	3.8
Prism	`(s -> Either a t) , (b -> t)`	3.16
Coalgebraic prism	`(s -> Either a (c t)) , (b -> t)`	3.8
Grate	`((s -> a) -> b) -> t`	3.29
Glass	`((s -> a) -> b) -> s -> t`	3.31
Affine Traversal	`s -> Either t (a , b -> t)`	3.24

Traversal	`s -> (Vec n a, Vec n b -> t)`	3.20
Kaleidoscope	`(Vec n a -> b) -> (Vec n s -> t)`	3.26
Setter	`(a -> b) , (s -> t)`	3.35
Fold	`s -> [a]`	3.34

6. Conclusions

We have extended a result by Pastro and Street to a setting that is useful for optics in functional programming. Using it, we have refined some of the optics already present in the literature to mixed optics, providing derivations for each one of them. We have also described new optics.

Regarding functional programming, the work suggests an architecture for a library of optics that would benefit from these results. Instead of implementing each optic separately, the general definition can be instantiated in all the particular cases. We can then just consider specific functions for constructing the more common families of optics. Tambara modules can be used to implement each one of the combinators of the library, ensuring that they work for as many optics as possible. The interested reader can find the implementation in Appendix 5.

Many of the other applications of optics may benefit from the flexibility of enriched and mixed optics. They may be used to capture some *lens*-like constructions and provide a general theory of how they should be studied; the specifics remain as future work.

6.1. Van Laarhoven encoding.

This paper has focused on the profunctor representation of optics. A similar representation that also provides the benefit of straightforward composition is the **van Laarhoven encoding** [40]. It is arguably less flexible than the profunctor representation, being based on representable profunctors, but it is more common in practice. For instance, traversals admit a different encoding in terms of profunctors represented by an applicative functor.

Proposition 6.1 (Van Laarhoven-style traversals)**.**

$$\mathbf{Traversal}((A,B),(S,T)) \cong \int_{F \in \mathbf{App}} \mathcal{V}(A, FB) \otimes \mathcal{V}(S, FT).$$

Proof.

$$\begin{aligned}
& \int_{F \in \mathbf{App}} \mathcal{V}(A, FB) \otimes \mathcal{V}(S, FT) \\
\cong\ & \{\text{Yoneda}\} \\
& \int_{F \in \mathbf{App}} [\mathcal{V},\mathcal{V}](A \otimes [B,-], F) \otimes \mathcal{V}(S, FT) \\
\cong\ & \{\text{Adjunction, free applicatives}\} \\
& \int_{F \in \mathbf{App}} \mathbf{App}\left(\sum_{n \in \mathbb{N}} A^n \otimes [B^n, -], F\right) \otimes \mathcal{V}(S, FT) \\
\cong\ & \{\text{Coyoneda}\}
\end{aligned}$$

$$\mathcal{V}\left(S, \sum_{n\in\mathbb{N}} A^n \otimes [B^n, T]\right). \qquad \square$$

Exactly the same technique yields lenses and *grates* [25], using arbitrary representable or corepresentable profunctors, respectively.

Proposition 6.2 (Van Laarhoven lenses [40])**.**

$$\mathbf{LinearLens}_{\otimes,\otimes}((A,B),(S,T)) \cong \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(A, FB) \otimes \mathcal{V}(S, FT).$$

Proof.

$$\begin{aligned} & \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(A, FB) \otimes \mathcal{V}(S, FT) \\ \cong\ & \{\text{Yoneda}\} \\ & \int_{F\in[\mathcal{V},\mathcal{V}]} [\mathcal{V},\mathcal{V}](A \otimes [B,-], F) \otimes \mathcal{V}(S, FT) \\ \cong\ & \{\text{Coyoneda}\} \\ & \mathcal{V}\left(S, A \otimes [B,T]\right). \end{aligned} \qquad \square$$

Proposition 6.3 (Van Laarhoven-style grates)**.**

$$\mathbf{Grate}((A,B),(S,T)) \cong \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(FA, B) \otimes \mathcal{V}(FS, T).$$

Proof.

$$\begin{aligned} & \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(FA, B) \otimes \mathcal{V}(FS, T) \\ \cong\ & \{\text{Yoneda}\} \\ & \int_{F\in[\mathcal{V},\mathcal{V}]} [\mathcal{V},\mathcal{V}](F, [[\bullet, A], B]) \otimes \mathcal{V}(FS, T) \\ \cong\ & \{\text{Coyoneda}\} \\ & \mathcal{V}\left([[S,A],B],T\right). \end{aligned} \qquad \square$$

6.2. **Related work.** Pastro and Street [29] first described the construction of *doubles* in their study of Tambara theory. Their results can be reused for *optics* thanks to the observations of [24]. The profunctor representation theorem and its implications for functional programming have been studied by [3]. We combine their approach with Pastro and Street's to get a proof of a more general version of this theorem.

The case of mixed optics was first mentioned by Riley [33, §6.1], but his work targeted a more restricted case. Specifically, the definitions of *optic* given in the literature [3, 24, 33] deal only with the particular case in which $\mathcal{V}$ = **Sets**, the categories **C** and **D** coincide, and the two actions are the same. Riley derives a class of optics and their laws [33, §4.4] that is closely related to ours in Section 3.6; our proposal makes stronger assumptions but may be more straightforward to apply in programming contexts. Riley uses the results of [16] to propose a description of the traversal in terms of traversable functors [33, §4.6]; our derivation simplifies this approach, which was in principle not suitable for the enriched case.

A central aspect of Riley's work is the extension of the concept of *lawful lens* to arbitrary *lawful optics* [33, §3]. This extension works exactly the same for the optics we define here, so we do not address it explicitly in this paper. A first reasonable notion of lawfulness for the case of mixed optics for two actions $(\textcircled{\scriptsize L})\colon \mathbf{M} \otimes \mathbf{C} \to \mathbf{C}$ and $(\textcircled{\scriptsize R})\colon \mathbf{N} \otimes \mathbf{D} \to \mathbf{D}$ is to use a cospan $\mathbf{C} \to \mathbf{E} \leftarrow \mathbf{D}$ *of actions* to push the two parts of the optic into the same category and then consider lawfulness in $\mathbf{E}$.

6.3. **Further work.** A categorical account of how optics of different kinds compose into optics is left for further work. Specifically, it should be able to explain the "lattice of optics" described in [3, 30]. Some preliminary results have been discussed by [35], but the proposal to model the lattice is still too *ad-hoc* to be satisfactory. The topic of lawfulness [33, §3] and how it relates to composition and mixed optics is also left for further work.

The relation between power series functors and traversables is implicit across the literature on polynomial functors and containers. It can be shown that *traversable* structures over an endofunctor T correspond to certain parameterised coalgebras using the free applicative construction [17]. We believe that it is possible to refine this result to make our derivation for traversals more practical for functional programming.

It can be noted that lenses are the optic for products, functors that distribute over strong functors. Traversals are the optic for traversables, functors that distribute over applicative functors. Both have a van Laarhoven representation in terms of strong and applicative functors respectively. A generalization of this phenomenon needs a certain Kan extension to be given a coalgebra structure [35, Lemma 4.1.3], but it does not necessarily work for any optic.

Optics have numerous applications in the literature, including game theory [13], machine learning [9] and model-driven development [38]. Beyond functional programming, enriched optics open new paths for explorating applications of optics. Both mixed optics and enriched optics allow us to more precisely adjust the existing definitions to match the desired applications.

Acknowledgements

This work was started in the last author's MSc thesis [35]; development continued at the Applied Category School 2019 at Oxford [32], and we thank the organizers of the School for that opportunity. We also thank Paweł Sobociński, Guillaume Boisseau, Mitchell Riley, Exequiel Rivas, Sjoerd Visscher, Christina Vasilakopoulou, Jules Hedges, Bruno Gavranović, Matteo Capucci and multiple anonymous reviewers for helpful comments on previous versions of this manuscript. The code for this text has been continued as a Haskell library [31].

Bryce Clarke was supported by the Australian Government Research Training Program Scholarship. Fosco Loregian and Mario Román were supported by the European Union through the ESF funded Estonian IT Academy research measure (project 2014-2020.4.05.19-0001).

References

[1] Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. Reflections on monadic lenses. In *A List of Successes That Can Change the World - Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*, volume 9600 of *Lecture Notes in Computer Science*, pages 1–31, Heidelberg, 2016. Springer. 8, 11, 12

[2] Guillaume Boisseau. Understanding profunctor optics: A representation theorem. Master's thesis, University of Oxford, 2017. 8, 11, 14, 15

[3] Guillaume Boisseau and Jeremy Gibbons. What you needa know about Yoneda: Profunctor optics and the Yoneda Lemma (functional pearl). *PACMPL*, 2(ICFP):84:1–84:27, 2018. 6, 8, 9, 12, 18, 25, 34, 41, 42

[4] Mario Cáccamo and Glynn Winskel. A higher-order calculus for categories. In *International Conference on Theorem Proving in Higher Order Logics*, pages 136–153. Springer, 2001. 7

[5] Geoffrey S. H. Cruttwell and Michael A. Shulman. A unified framework for generalized multicategories. *Theory and Applications of Categories*, 24(21):580–655, 2010. 32

[6] Brian Day. On closed categories of functors. In *Reports of the Midwest Category Seminar IV*, pages 1–38, Heidelberg, 1970. Springer. 10, 36

[7] Brian J. Day and Miguel L. Laplaza. On embedding closed categories. *Bulletin of the Australian Mathematical Society*, 18(3):357–371, 1978. 21, 22

[8] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936. 14

[9] Brendan Fong and Michael Johnson. Lenses and learners. *CoRR*, abs/1903.03671, 2019. 42

[10] J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. In *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 233–246, 2005. 3, 5, 11, 15

[11] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976. 12

[12] Phil Freeman, Brian Marick, Lukas Heidemann, et al. Purescript Profunctor Lenses. Github `https://github.com/purescript-contrib/purescript-profunctor-lenses`, 2015–2019. 4

[13] Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481, 2018. 42

[14] John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. 32

[15] Bart Jacobs, Chris Heunen, and Ichiro Hasuo. Categorical semantics for arrows. *Journal of Functional Programming*, 19(3-4):403–438, 2009. 32

[16] Mauro Jaskelioff and Russell O'Connor. A representation theorem for second-order functionals. *Journal of Functional Programming*, 25, 2015.

18, 41

[17] Mauro Jaskelioff and Ondrej Rypacek. An investigation of the laws of traversals. In *Proceedings Fourth Workshop on Mathematically Structured Functional Programming, MSFP@ETAPS 2012, Tallinn, Estonia, 25 March 2012.*, pages 40–49, 2012. 6, 18, 42

[18] G. Max Kelly. Basic concepts of enriched category theory. *Reprints in Theory and Applications of Categories*, 1(10):137, 2005. Reprint of the 1982 original. Cambridge Univ. Press, Cambridge; MR0651714. 36

[19] G. Max Kelly. On the operads of J. P. May. *Reprints in Theory and Applications of Categories*, 13(1), 2005. 19

[20] Edward Kmett. Lens library, version 4.16. Hackage `https://hackage.haskell.org/package/lens-4.16`, 2012–2018. 4, 22, 23, 24, 38, 39

[21] Fosco Loregian. Coend calculus. *arXiv preprint arXiv:1501.02503*, 2019. 7, 8

[22] Saunders Mac Lane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer New York, 1978. 20

[23] Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1):1–13, 2008. 18

[24] Bartosz Milewski. Profunctor optics: The categorical view. `https://bartoszmilewski.com/2017/07/07/profunctor-optics-the-categorical-view/`, 2017. 6, 9, 11, 18, 25, 35, 41

[25] Russell O'Connor. Grate: A new kind of optic. `https://r6research.livejournal.com/28050.html`, 2015. 22, 41

[26] Russell O'Connor. Mezzolens: Pure Profunctor Functional Lenses. Hackage `https://hackage.haskell.org/package/mezzolens`, 2015. 4

[27] Frank Joseph Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages.* PhD thesis, Syracuse University, Syracuse, NY, USA, 1982. AAI8301650. 11

[28] Luke Palmer. Making Haskell nicer for game programming. `https://lukepalmer.wordpress.com/2007/07/26/making-haskell-nicer-for-game-programming/`, archived at `https://web.archive.org/web/20141219182332/http://lukepalmer.wordpress.com/2007/07/26/making-haskell-nicer-for-game-programming/`, 2007. 11

[29] Craig Pastro and Ross Street. Doubles for monoidal categories. *Theory and Applications of Categories*, 21(4):61–75, 2008. 6, 8, 25, 26, 27, 33, 34, 41

[30] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *Programming Journal*, 1(2):7, 2017. 6, 18, 25, 42

[31] Emily Pillmore and Mario Román. Vitrea library, version 0.1.0.0. Hackage `https://hackage.haskell.org/package/vitrea-0.1.0.0`, Github `https://github.com/mroman42/vitrea`, 2019–2020. 42

[32] Emily Pillmore and Mario Román. Profunctor optics: The categorical view. `https://golem.ph.utexas.edu/category/2020/01/profunctor_optics_the_categori.html`, 2020. 42

[33] Mitchell Riley. Categories of optics. *arXiv preprint arXiv:1809.00738*, 2018. 6, 8, 9, 11, 12, 15, 18, 19, 23, 25, 41, 42

[34] Exequiel Rivas and Mauro Jaskelioff. Notions of computation as monoids. *Journal of Functional Programming*, 27, 2017. 32

[35] Mario Román. Profunctor optics and traversals. Master's thesis, University of Oxford, 2019. 18, 24, 42

[36] Mario Román. Open diagrams via coend calculus. *Electronic Proceedings in Theoretical Computer Science*, 333:65–78, Feb 2021. 9

[37] David I. Spivak. Generalized categories via functors $\mathcal{C}^{\mathrm{op}} \to \mathsf{Cat}$. *arXiv preprint arXiv:1908.02202*, 2019. 8, 11, 12

[38] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and Systems Modeling*, 9(1):7–20, 2010. 42

[39] Daisuke Tambara. Distributors on a tensor category. *Hokkaido mathematical journal*, 35(2):379–425, 2006. 6, 25, 26

[40] Twan van Laarhoven. CPS-based functional references. `https://www.twanvl.nl/blog/haskell/cps-functional-references`, 2009. 3, 5, 40, 41

[41] Twan van Laarhoven. A non-regular data type challenge. `https://www.twanvl.nl/blog/haskell/non-regular1`, 2009. 21

[42] Sjoerd Visscher. Data.Category library, version 0.10. Hackage `https://hackage.haskell.org/package/data-category`, 2010–2020. 36

APPENDIX C

Curriculum Vitae

CURRICULUM VITAE

MARIO ROMÁN

1. Background

Doctoral student at *Tallinn University of Technology* under the supervision of Paweł Sobociński. MSc. in Mathematics and Computer Science at the *University of Oxford* (2019). Two simultaneous and separate Bachelor Degrees, one in Mathematics and one in Computer Engineering, at the *University of Granada* (2018).

2. Publications

1. **Mario Román**. "Promonads and String Diagrams for Effectful Categories". In: *Applied Category Theory Conference*. ACT '22. Glasgow, United Kingdom, 2022. DOI: 10.48550/arXiv.2205.07664. arXiv: 2205.07664. URL: https://doi.org/10.48550/arXiv.2205.07664
2. Elena Di Lavore and **Mario Román**. "Evidential Decision Theory via Partial Markov Categories". In: *To be presented at the 38th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '23. 2023. DOI: 10.48550/arXiv.2301.12989. arXiv: 2301.12989. URL: https://doi.org/10.48550/arXiv.2301.12989
3. 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. LICS '22. Haifa, Israel: Association for Computing Machinery, 2022. ISBN: 9781450393515. DOI: 10.1145/3531130.3533365

4. 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]

5. Elena Di Lavore, Alessandro Gianola, **Mario Román**, Nicoletta Sabadini, and Pawel Sobocinski. "A Canonical Algebra of Open Transition Systems". In: *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings*. Ed. by Gwen Salaün and Anton Wijs. Vol. 13077. Lecture Notes in Computer Science. Springer, 2021, pp. 63–81. DOI: 10.1007/978-3-030-90636-8_4. URL: https://doi.org/10.1007/978-3-030-90636-8%5C_4

6. Guillaume Boisseau, Chad Nester, and **Mario Román**. "Cornering Optics". In: *Applied Category Theory Conference*. ACT '22. Glasgow, United Kingdom, 2022. DOI: 10.48550/arXiv.2205.00842. arXiv: 2205.00842. URL: https://doi.org/10.48550/arXiv.2205.00842

7. **Mario Román**. "Open Diagrams via Coend Calculus". In: *Applied Category Theory Conference*. Vol. 333. ACT '20. Boston, USA: Open Publishing Association, Feb. 2021, pp. 65–78. DOI: 10.4204/eptcs.333.5. URL: http://dx.doi.org/10.4204/EPTCS.333.5

8. Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and **Mario Román**. "Profunctor optics, a categorical update". In: *Compositionality, to appear* (2020). arXiv: 2001.07488. URL: https://arxiv.org/abs/2001.07488

3. Awards

Kleene Award to the Best Student Paper and *Selected as Distinghised Paper* at LiCS'23 for "Monoidal Streams for Dataflow Programming", joint with Elena Di Lavore and Giovanni de Felice.

Spanish Royal Mathematical Society (RSME-UGR) prize to the best Mathematics Bachelor Thesis at the University of Granada, 2018. "Category Theory and Lambda Calculus", supervised by Pedro García-Sánchez.

Undergraduate Research Fellowship at the Department of Algebra of the University of Granada, for the Haskell development of the "Mikrokosmos" software.

International Mathematical Olympiad (IMO) Honorary Mention, economic support for preparation from the Spanish Royal Mathematical Society (RSME, 2012).

4. Community

Reviewing. Executive editor at Compositionality. Conference reviewer at LiCS, CONCUR, FoSSaCS, CALCO, and ACT. Journal reviewer at LMCS and Compositionality. Program committee member at ACT 2022 and ACT 2023.

Teaching. Teaching assistant for Functional Programming (IT0212) and Category Theory (ITI9200). Teaching Assistant for the Applied Category Theory Adjoint School at Glasgow, 2022.

Version for the 1st November, 2023

APPENDIX D

Curriculum Vitae (Eesti keeles)

CURRICULUM VITAE

MARIO ROMÁN

1. Taustaosa

Doktorant *Tallinna Tehnikaülikoolis* Paweł Sobociński juhendamisel. MSc. matemaatika ja arvutiteaduse erialal *Oxfordi ülikoolis* (2019). Bakalaureusekraad matemaatikas ja arvutitehnikas, *Granada ülikoolis* (2018).

2. Publikatsioonid

1. **Mario Román**. "Promonads and String Diagrams for Effectful Categories". In: *Applied Category Theory Conference*. ACT '22. Glasgow, United Kingdom, 2022. DOI: `10.48550/arXiv.2205.07664`. arXiv: `2205.07664`. URL: `https://doi.org/10.48550/arXiv.2205.07664`
2. Elena Di Lavore and **Mario Román**. "Evidential Decision Theory via Partial Markov Categories". In: *To be presented at the 38th Annual ACM/IEEE Symposium on Logic in Computer Science*. LICS '23. 2023. DOI: `10.48550/arXiv.2301.12989`. arXiv: `2301.12989`. URL: `https://doi.org/10.48550/arXiv.2301.12989`
3. 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*. LICS '22. Haifa, Israel: Association for

Computing Machinery, 2022. ISBN: 9781450393515. DOI: 10.1145/3531130.3533365

4. 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]

5. Elena Di Lavore, Alessandro Gianola, **Mario Román**, Nicoletta Sabadini, and Pawel Sobocinski. "A Canonical Algebra of Open Transition Systems". In: *Formal Aspects of Component Software - 17th International Conference, FACS 2021, Virtual Event, October 28-29, 2021, Proceedings.* Ed. by Gwen Salaün and Anton Wijs. Vol. 13077. Lecture Notes in Computer Science. Springer, 2021, pp. 63–81. DOI: 10.1007/978-3-030-90636-8_4. URL: https://doi.org/10.1007/978-3-030-90636-8%5C_4

6. Guillaume Boisseau, Chad Nester, and **Mario Román**. "Cornering Optics". In: *Applied Category Theory Conference.* ACT '22. Glasgow, United Kingdom, 2022. DOI: 10.48550/arXiv.2205.00842. arXiv: 2205.00842. URL: https://doi.org/10.48550/arXiv.2205.00842

7. **Mario Román**. "Open Diagrams via Coend Calculus". In: *Applied Category Theory Conference.* Vol. 333. ACT '20. Boston, USA: Open Publishing Association, Feb. 2021, pp. 65–78. DOI: 10.4204/eptcs.333.5. URL: http://dx.doi.org/10.4204/EPTCS.333.5

8. Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregiàn, Bartosz Milewski, Emily Pillmore, and **Mario Román**. "Profunctor optics, a categorical update". In: *Compositionality, to appear* (2020). arXiv: 2001.07488. URL: https://arxiv.org/abs/2001.07488

3. Auhinnad

Kleene'i auhind parimale õpilastööle LiCS'23-s 'Monoidal Streams for Dataflow Programming' eest, koos Elena Di Lavore'i ja Giovanni de Felice'iga.

Hispaania Kuningliku Matemaatika Seltsi (RSME-UGR) auhind parimale matemaatika bakalaureusetööle Granada ülikoolis, 2018. "Kategooriateooria ja lambdaarvutus", juhendaja Pedro García-Sánchez.

Bakalaureuseõppe uurimisstipendium Granada ülikooli algebra osakonnas Haskelli tarkvara.

Rahvusvaheline matemaatikaolümpiaad (IMO, RSME, 2012). Aunimetus.

4. Teadustegevus

Retsenseerimine. Compositionality tegevtoimetaja. LiCS, CONCUR, FoSSaCS, CALCO ja ACT retsensent. Ajakirjade LMCS-is ja Compositionality retsensent. ACT 2022 ja ACT 2023 programmikomitee liige.

Õppetöö. Funktsionaalse programmeerimise (IT0212) ja kategooriateooria (ITI9200) õppeassistent. Glasgow' rakenduskategooriateooria ühiskooli õppeassistent, 2022.

Version for the 9th November, 2023

APPENDIX E

Non-Exclusive License for Reproduction and Publication

ISSN 2585-6901 (PDF)
ISBN 978-9916-80-082-9 (PDF)