The Produoidal Algebra of Process Decomposition

Matt Earnshaw
Tallinn University of Technology, Estonia

James Hefford
University of Oxford, UK

Mario Román
Tallinn University of Technology, Estonia

Abstract

We characterize a universal 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. In particular, symmetric monoidal contexts coincide with monoidal lenses and endow them with a novel universal property. We apply this algebraic structure to the analysis of multi-party protocols in arbitrary theories of processes.

2012 ACM Subject Classification Theory of computation → Categorical semantics

Keywords and phrases monoidal categories, profunctors, lenses, duoidal categories

Digital Object Identifier 10.4230/LIPIcs.CSL.2024.25

Related Version *Full Version*: `https://arxiv.org/abs/2301.11867` [20]

Funding 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].

Acknowledgements We thank Pawel Sobocinski, Fosco Loregian, Chad Nester and David Spivak for discussion. We thank the CSL reviewers for suggestions that lead to considerable improvements.

1 Introduction

Theories of processes, such as *stochastic*, *partial* or *linear* functions, are a foundational tool in computer science. They help us model interacting systems using a solid mathematical structure. Any theory of processes involving operations for *sequential* and *parallel composition*, satisfying reasonable axioms, forms a *monoidal category*. Monoidal categories are versatile: they can be used in the description of quantum circuits [2], stochastic processes [11, 24], relational queries [9] and non-terminating processes [13], among other applications [16].

At the same time, monoidal categories have two intuitive, sound and complete calculi: the first in terms of *string diagrams* [40], and the second in terms of their *linear type theory* [63]. 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 [37, 51], which becomes *do-notation* for Kleisli categories of commutative monads [46, 28].

This manuscript studies an algebra of context for monoidal categories. 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 [45], the same concept was missing for 2-dimensional syntaxes, which are still less frequent in computer science [21]. Let us showcase monoidal categories, their string diagrams and the use of do-notation in the description of a protocol.

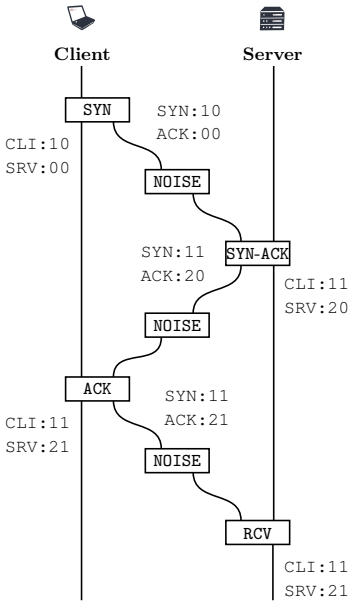


Figure 1 TCP Three way handshake.

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` [55]. 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, $\texttt{SRV} + 1$. When the protocol works correctly, both client and server end up with the pair $(\texttt{CLI} + 1, \texttt{SRV} + 1)$.

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. 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 2a, see the full version [20, Appendix C.1] [46]). 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 the full version [20, Figure 14] for the diagrammatic representation). We say that these are only *monoidal contexts*.

Contrary to monoidal morphisms, which only need to declare their input and output types, monoidal contexts need *behavioural types* [54, 38] that specify the order and type of the exchange of information along their boundary. A monoidal context may declare intermediate *send* ($\mathbf{!}A$) and *receive* ($\mathbf{?}A$) 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**, at the same time it *sends*, *receives* and then *sends* a message; and the server transforms its memory type, **Server** while, dually to the client, it *receives*, *sends* and then *receives* a message (Figure 2b).

(a)

```
syn :: (Cli, Srv) ~>
  ((Cli, Srv), (Syn, Ack))
syn(client, server) = do
  client <- random
  return ((cli, 0), (cli, 0))
```

(b)

$$\in \mathcal{LC}\left(\begin{smallmatrix}\textbf{Client}\\ \textbf{Client}\end{smallmatrix}; \mathbf{!Msg} \triangleleft \mathbf{?Msg} \triangleleft \mathbf{!Msg}\right);$$

$$\in \mathcal{LC}\left(\begin{smallmatrix}\textbf{Server}\\ \textbf{Server}\end{smallmatrix}; \mathbf{?Msg} \triangleleft \mathbf{!Msg} \triangleleft \mathbf{?Msg}\right);$$

$$\texttt{NOISE} \in \mathbb{C}\left(\mathbf{Msg}; \mathbf{Msg}\right);$$

Figure 2 (a) Implementation of `SYN`. (b) Types for the three parties.

Session types [36], including the send ($\mathbf{!}A$) and receive ($\mathbf{?}A$) polarized types, have been commonplace in logics of message passing. Cockett and Pastro [14] 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 ($\mathbf{!}/\mathbf{?}$) 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* [4, 59].

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 3 achieve.

Here, we define $\texttt{SYN}^* = \texttt{SYN} \fatsemi \texttt{PRJ}$ to be the same as the `SYN` process but projecting out only the client `CLI` number. We also define a new $\texttt{ACK}^*$ that ignores the server `SRV` number, so that $\texttt{ACK} = \texttt{PRJ} \fatsemi \texttt{ACK}^*$. These two equations are enough to complete our reasoning.

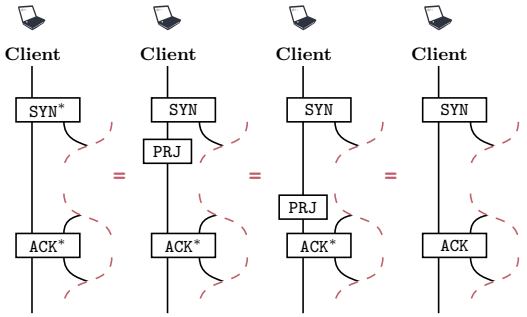


Figure 3 Reasoning only with the client.

Monoidal contexts and their incomplete diagrams are defined to be convenient tuples of morphisms, e.g. (SYN|ACK) in our example; what makes them interesting is the equivalence relation we impose on them: this equivalence relation makes the pair (SYN ⨟ PRJ|ACK*) equal to (SYN|PRJ ⨟ 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 ($\lhd$) and parallel ($\otimes$) composition. This is what duoidal categories provide. Duoidal categories are categories with two monoidal structures, e.g. $(\lhd, N)$ 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 [62]: 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 $(\lhd, \otimes)$ 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 [10, 66].

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 5.4) 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 characterization 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 universally characterize an algebra of monoidal contexts, in the form of a *normal produoidal* category. In fact, recently, there have been multiple implicit applications of monoidal contexts. Kissinger and Uijlen [41] describe higher order quantum processes using contexts with holes in compact closed monoidal categories. Ghani, Hedges, Winschel and Zahn [27] describe economic game theory in terms of *lenses* and incomplete processes in cartesian monoidal categories. Bonchi, Piedeleu, Sobociński and Zanasi [8] study contextual equivalence in their monoidal category of affine signal flow graphs. Di Lavore, de Felice and Román [19] define *monoidal streams* by iterating monoidal context coalgebraically.

Category theory. Street already noted that the endoprofunctors of a monoidal category had a duoidal structure [66]; Pastro and Street described a promonoidal structure on lenses [50] and Garner and López-Franco contributed a partial normalization procedure for duoidal categories [25]. 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 3.8, 4.3, and 5.4).

Language theory. Motivated by language theory and the Chomsky-Schützenberger theorem, Melliès and Zeilberger [45] 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 [21] 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 [35, 36] are the mainstay type formalism for communication protocols, and they have been extensively applied to the π-calculus [60]. Our approach is not set up to capture all of the features of a fully fledged session type theory [43]. Arguably, this makes it more general in what it does: it always provides a universal way of implementing send ($\mathbf{!}A$) and receive ($\mathbf{?}A$) operations in an arbitrary theory of processes represented by a monoidal category. For instance, recursion and the internal/external choice duality [26, 54] are not discussed, although they could be considered as extensions in the same way they are to monoidal categories: via trace [29] and linear distributivity [15].

Lenses and incomplete diagrams. Lenses are a notion of bidirectional transformation [23] that can be cast in arbitrary monoidal categories. The first mention of monoidal lenses separate from their classical database counterparts [39] is due to Pastro and Street [50], who identify them as an example of a promonoidal category. However, it was with a different monoidal structure [56] that they became popular in recent years, spawning applications not only in bidirectional transformations [23] but also in functional programming [53, 12], open games [27], polynomial functors [49] and quantum combs [31]. 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* [57, 59].

Finally, Nester et al. have recently proposed a syntax for lenses and message-passing [48, 7] and lenses themselves have been applied to protocol specification [67]. Spivak [65] also discusses the multicategory of *wiring diagrams*, later used for incomplete diagrams [52] and

related to lenses [61]. 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 universal characterization of a produoidal category of *monoidal contexts* over a monoidal category (Theorem 5.4).

We construct an adjunction between monoidal categories and produoidal categories in Section 3, and we characterize spliced monoidal arrows as the cofree produoidal category over a monoidal category (Theorem 3.8); in order to do this, we also introduce a version of the splice-contour construction that creates an adjunction between categories and promonoidal categories, the interested reader can follow the full version [20, Appendix B].

We introduce the free normalization of an arbitrary produoidal category (Theorem 4.3). Normalization had been only described for well-behaved duoidal categories [25]; we show that any produoidal category can be normalized and we construct an idempotent normalization monad. We universally characterize the algebra of monoidal contexts as a free normalization (Theorem 5.4) in Section 5; we universally characterize the algebra of monoidal lenses as a free symmetric normalization (Theorem 6.2) in Section 6. Finally, we introduce an interpretation of send/receive types (**!**/**?**) (Proposition 6.5) in terms of monoidal lenses.

2 Preliminaries: Profunctors and Dinaturality

Profunctors describe families of processes indexed by the input and output types of a category. Since they will be our main tool in the following, we give a brief introduction. More details can be found in the full version of this paper [20, Appendix A].

▶ **Definition 2.1.** *A* profunctor $P\colon \mathbb{B}_0 \times \ldots \times \mathbb{B}_m \nrightarrow \mathbb{A}_0 \times \ldots \times \mathbb{A}_n$ *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 $(\prec_0, \ldots, \prec_m)$ actions of the morphisms of $\mathbb{A}_0, \ldots, \mathbb{A}_n$ and $\mathbb{B}_0, \ldots, \mathbb{B}_m$, respectively [5, 44].[1]

Composing profunctors 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:

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

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

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

[1] We simply use $(\prec/\succ)$ without any subscript whenever the input/output is unique.

These are different descriptions, made up of two different components. However, they essentially describe the same process [19]: they are *dinaturally equal*. Indeed, profunctors are canonically endowed with this notion of equivalence [5, 44], precisely equating these two descriptions. Profunctors, and their elements, are thus composed *up to dinatural equivalence*.

▶ **Definition 2.2** (Dinatural equivalence). *Consider two profunctors* $P\colon \mathbb{B}_0 \times \ldots \times \mathbb{B}_m \nrightarrow \mathbb{A}_0 \times \ldots \times \mathbb{A}_n$ *and* $Q\colon \mathbb{D}_0 \times \ldots \times \mathbb{D}_h \nrightarrow \mathbb{C}_0 \times \ldots \times \mathbb{C}_k$ *such that* $\mathbb{B}_i = \mathbb{C}_j$*; and let* $\mathbf{S}^{i,j}_{P,Q}(A;D)$ *be the set*

$$\sum_{X \in \mathbb{B}_i} P(A_0 \ldots A_n; B_0 \ldots X \ldots B_m) \times Q(C_0 \ldots X \ldots C_k; D_0 \ldots D_h).$$

Dinatural equivalence, $(\sim)$, *on the set* $\mathbf{S}^{i,j}_{P,Q}(A;D)$ *is the smallest equivalence relation satisfying* $(p \prec_i f \,|_j\, q) \sim (p \,|_i\, f \succ_j q)$*. The* coend *is defined as this coproduct quotiented by dinaturality,* $\mathbf{S}^{i,j}_{P,Q}(A;D)/(\sim)$*, and written as an integral.*

$$\int^{X \in \mathbb{C}_j} P(A_0 \ldots A_n; B_0 \ldots X \ldots B_m) \times Q(C_0 \ldots X \ldots C_k; D_0 \ldots D_h).$$

▶ **Definition 2.3** (Profunctor composition). *Consider two profunctors* $P\colon \mathbb{B}_0 \times \ldots \times \mathbb{B}_m \nrightarrow \mathbb{A}_0 \times \ldots \times \mathbb{A}_n$ *and* $Q\colon \mathbb{D}_0 \times \ldots \times \mathbb{D}_h \nrightarrow \mathbb{C}_0 \times \ldots \times \mathbb{C}_k$ *such that* $\mathbb{B}_i = \mathbb{C}_j$*; their* composition *along ports* i *and* j *is a profunctor* $P \diamond Q : \mathbb{B}_0 \times \ldots \times \mathbb{D}_0 \times \ldots \times \mathbb{D}_h \times \ldots \times \mathbb{B}_m \nrightarrow \mathbb{C}_0 \times \ldots \times \mathbb{A}_0 \times \ldots \times \mathbb{A}_n \times \ldots \times \mathbb{C}_k$*; we write it marking this connection*

$$P(A_0 \ldots A_n; B_0 \ldots \bullet \ldots B_m) \diamond Q(C_0 \ldots \bullet \ldots C_k; D_0 \ldots D_h),$$

and it is defined as the coproduct of the product of both profunctors, indexed by the common variable, and quotiented by dinatural equivalence,

$$\int^{X \in \mathbb{C}} P(A_0 \ldots A_n; B_0 \ldots X \ldots B_m) \times Q(C_0 \ldots X \ldots C_k; D_0 \ldots D_h).$$

3 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).

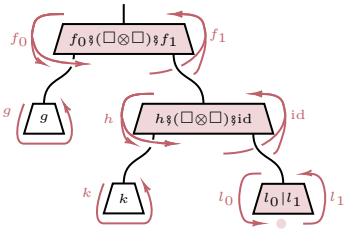


Figure 4 Example decomposition.

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 filled with l_0, id_I, and l_1.

This section studies decomposition of morphisms in a *monoidal* category, in the same way we study decomposition of morphisms in a category (see the full version [20, Appendix B]). We present an algebraic structure for decomposing both sequential and parallel compositions: *produoidal categories*.

3.1 Produoidal Categories

Produoidal categories, first defined by Booker and Street [10], 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 splits*, $\mathbb{V}(X;Y_0 \lhd Y_1)$, and *sequential units*, $\mathbb{V}(X;N)$, provided by a promonoidal structure; and *parallel splits*, $\mathbb{V}(X;Y_0 \otimes Y_1)$ and *parallel units*, $\mathbb{V}(X;I)$, provided by another promonoidal structure.

These splits must be coherent. For instance, imagine we want to decompose X (sequentially) into Y_0, Y_1 and Y_2. Decomposing X into Y_0 and something ($\bullet$), and then decomposing that something into Y_1 and Y_2 *should be doable in essentially the same ways* as decomposing X into something ($\bullet$) and Y_2, and then decomposing that something into Y_0 and Y_1. Formally, we are saying that,

$$\mathbb{V}(X;Y_0 \lhd \bullet) \diamond \mathbb{V}(\bullet;Y_1 \lhd Y_2) \cong \mathbb{V}(X;\bullet \lhd Y_2) \diamond \mathbb{V}(\bullet;Y_0 \lhd Y_1),$$

and, in fact, we just write $\mathbb{V}(X;Y_0 \lhd Y_1 \lhd Y_2)$ for the set of such decompositions. 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,*

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

such that one laxly distributes over the other. This is to say that it is endowed with the following natural laxators*: $\psi_2\colon \mathbb{V}(\bullet;(X \lhd Y) \otimes (Z \lhd W)) \to \mathbb{V}(\bullet;(X \otimes Z) \lhd (Y \otimes W))$, $\psi_0\colon \mathbb{V}(\bullet;I) \to \mathbb{V}(\bullet;I \lhd I)$, $\varphi_2\colon \mathbb{V}(\bullet;N \otimes N) \to \mathbb{V}(\bullet;N)$, and $\varphi_0\colon \mathbb{V}(\bullet;I) \to \mathbb{V}(\bullet;N)$. Laxators, together with unitors and associators, must satisfy coherence conditions (see [20, Definition J.7]). Denote by* **Produo** *the category of produoidal categories and produoidal functors.*

▶ Remark 3.2 (Nesting profunctorial structures). Notation for nesting functorial structures, say ($\lhd$) and ($\otimes$), is straightforward: we use expressions like $(X_1 \otimes Y_1) \lhd (X_2 \otimes Y_2)$ without a second thought. Nesting the profunctorial (or *virtual*) structures ($\lhd$) and ($\otimes$) is more subtle: defining $\mathbb{V}(\bullet;X \otimes Y)$ and $\mathbb{V}(\bullet;X \lhd Y)$ for each pair of objects X and Y does not itself define what something like $\mathbb{V}(\bullet;(X_1 \otimes Y_1) \lhd (X_2 \otimes Y_2))$ means. Recall that, in the profunctorial case, $X_1 \lhd Y_1$ and $X_1 \otimes Y_1$ are not objects themselves: they are just names for the profunctors $\mathbb{V}(\bullet;X_1 \lhd Y_1)$ and $\mathbb{V}(\bullet;X_1 \otimes Y_1)$, which are not *representable*.

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

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 3.3** (Monoidal contour). *The* contour *of a produoidal category* $\mathbb{B}$ *is the monoidal category* $\mathcal{D}\mathbb{B}$ *presented by two objects,* X^L *(left-handed) and* X^R *(right-handed), for each object* $X \in \mathbb{B}_{obj}$*; and generated by arrows that arise from* contouring *both sequential and parallel decompositions of the promonoidal category.*

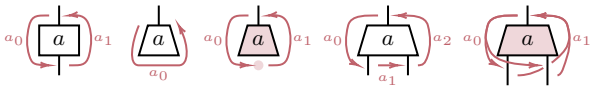
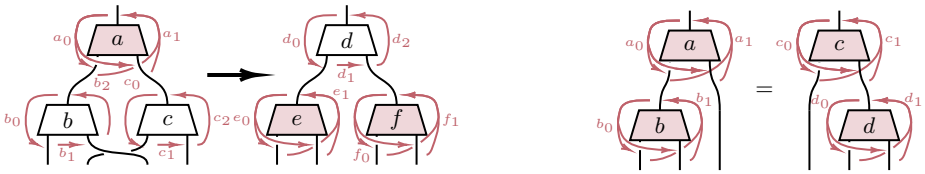


Figure 5 Generators of the monoidal category of contours.

Specifically, it is presented by the following generators *(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 5.

We impose some equations that arise naturally from the associator and unitor of the sequential structure ($\triangleleft$), as done by Melliès and Zeilberger [45]; but moreover, we also impose some new equations, coming from the parallel structure ($\otimes$), as depicted in Figures 6a and 6b. We refer the interested reader to the full version [20] for the full list of equations.



(a) **(b)**

Figure 6 Equations coming from laxators (a) and associators (b).

▶ **Proposition 3.4.** *Monoidal contour extends to a functor* $\mathcal{D} : \mathbf{Produo} \to \mathbf{Mon}$.

Proof. See the full version [20, Proposition E.3]. ◀

3.3 Produoidal Category of Spliced Monoidal Arrows

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 3.5.** *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*

- *morphisms,* $\mathcal{T}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}X\\Y\end{smallmatrix}\right) = \mathbb{C}(A; X) \times \mathbb{C}(Y, B)$;
- *sequential splits,* $\mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}X \triangleleft X'\\Y\end{smallmatrix}) = \mathbb{C}(A; X) \times \mathbb{C}(Y; X') \times \mathbb{C}(Y'; B)$;
- *parallel splits,* $\mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}; \begin{smallmatrix}X \otimes X'\\Y\end{smallmatrix}) = \mathbb{C}(A; X \otimes X') \times \mathbb{C}(Y \otimes Y'; B)$;
- *sequential units,* $\mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}; N) = \mathbb{C}(A; B)$;
- *and parallel units,* $\mathcal{T}\mathbb{C}(\begin{smallmatrix}A\\B\end{smallmatrix}; I) = \mathbb{C}(A; I) \times \mathbb{C}(I; B)$.

In other words, morphisms are pairs of arrows *written as* $f \fatsemi \Box \fatsemi g \in \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \\ Y \end{smallmatrix}\right)$. *Sequential splits are* triples of arrows, *written as* $f \fatsemi \Box \fatsemi g \fatsemi \Box \fatsemi h \in \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \\ Y \end{smallmatrix} \triangleleft \begin{smallmatrix} X' \\ Y' \end{smallmatrix}\right)$. *Parallel splits are* pairs of arrows, *written as* $f \fatsemi (\Box \otimes \Box) \fatsemi h \in \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \\ Y \end{smallmatrix} \otimes \begin{smallmatrix} X' \\ Y' \end{smallmatrix}\right)$. *Sequential units are* arrows, *written simply as* $f \in \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; N\right)$. *parallel units are pairs of arrows, written as* $f \parallel g \in \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; I\right)$.

Finally, the laxators are defined by plugging the different pieces and reinterpreting the relative position of the holes. Let us give an example of what this means; we refer the interested reader to the full version [20, Appendix E.2] for full details.

▶ **Example 3.6.** For instance, the last laxator takes parallel sequences of holes, $f_0 \fatsemi ((h_0 \fatsemi \Box \fatsemi h_1 \fatsemi \Box \fatsemi h_2) \otimes (k_0 \fatsemi \Box \fatsemi k_1 \fatsemi \Box \fatsemi k_2)) \fatsemi f_1$ into sequences of parallel holes, $f_0 \fatsemi (h_0 \otimes k_0) \fatsemi (\Box \otimes \Box) \fatsemi (h_1 \otimes k_1) \fatsemi (\Box \otimes \Box) \fatsemi (h_2 \otimes k_2) \fatsemi f_1$.

▶ Remark 3.7. The produoidal algebra of spliced arrows is a natural construction: abstractly, we know that there is a duoidal structure on the endomodules of any monoidal category [18, 66] – this is its explicit produoidal counterpart. What may be more surprising is that spliced arrows have themselves a universal property as part of an adjunction.

▶ **Theorem 3.8.** *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* $\mathcal{T} : \mathbf{Mon} \to \mathbf{Produo}$. *The monoidal contour and the produoidal splice are left and right adjoints to each other, respectively.*

Proof. See the full version [20, Propositions E.4 and E.9 and Theorem E.10]. ◀

▶ Remark 3.9. When $\mathbb{C}$ is a *symmetric* monoidal category, then $\mathcal{T}\mathbb{C}$ is moreover a symmetric produoidal category with symmetry defined using the symmety of $\mathbb{C}$.

3.4 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, $(\otimes)$. Spliced monoidal arrows form a monoidal category.

▶ **Proposition 3.10.** *Parallel splits and parallel units of spliced monoidal arrows are representable profunctors. That is,* $\mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \\ Y \end{smallmatrix} \otimes \begin{smallmatrix} X' \\ Y' \end{smallmatrix}\right) \cong \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \otimes X' \\ Y \otimes Y' \end{smallmatrix}\right)$, *and* $\mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; I\right) \cong \mathcal{T}\mathbb{C}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} I \\ I \end{smallmatrix}\right)$.

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

Normalization is a procedure to mix both tensors of a duoidal category, $(\otimes)$ and $(\triangleleft)$, but not every duoidal category has a normalization [25]. 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* [18]; thus, one would expect only some produoidal categories to be normalizable. 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 (Theorem 4.3). We use this procedure to universally characterize *monoidal contexts* in Section 5, which form a produoidal category without representable structure.

▶ Remark 3.11. This means $\mathcal{T}\mathbb{C}$ has the structure of a *virtual duoidal category* [64] or *monoidal multicategory*, defined by Aguiar, Haim and López Franco [3] as a pseudomonoid in the cartesian monoidal 2-category of multicategories.

4 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 4.1** (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 4.2** (Normalization construction). *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 [33, 58]. Moreover, the Kleisli category of this promonad is a normal produoidal category with the following splits and units:* $\mathcal{N}\mathbb{V}(A; B \otimes_N C) = \mathbb{V}(A; N \otimes B \otimes N \otimes C \otimes N)$; $\mathcal{N}\mathbb{V}(A; B \triangleleft_N C) = \mathbb{V}(A; (N \otimes B \otimes N) \triangleleft (N \otimes C \otimes N))$; $\mathcal{N}\mathbb{V}(A; I_N) = \mathbb{V}(A; N)$; *and* $\mathcal{N}\mathbb{V}(A; N_N) = \mathbb{V}(A; N)$.

Proof. See the full version [20, Theorem F.1]. ◀

Garner and López Franco [25] 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 4.3). 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**. See the full version [20, Appendix F.4] for an outline of the relation between the two procedures.

▶ **Theorem 4.3** (Free normal produoidal). *Normalization extends to an idempotent monad. Moreover, 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. See the full version [20, Theorems F.3 and F.5]. ◀

In the previous Section 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.

▶ Remark 4.4. 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: symmetric normalization. See the full version [20, Appendix F.2].

5 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 this normal produoidal category of monoidal contexts.

5.1 The Category of Monoidal Contexts

A monoidal context, $\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)$, 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 5.1** (Monoidal context)**.** *Let $(\mathbb{C},\otimes,I)$ be a monoidal category.* Monoidal contexts *are the elements of the profunctor $\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2)\diamond\mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2;B)$ over $\mathbb{C}^{op}\times\mathbb{C}$.*

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 8). We write monoidal contexts as

$$(f\fatsemi(\mathrm{id}_M\otimes\blacksquare\otimes\mathrm{id}_N)\fatsemi g)\in\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right).$$

In this notation, dinaturality explicitly means $(f\fatsemi(m\otimes\mathrm{id}_X\otimes n)\fatsemi(\mathrm{id}_W\otimes\blacksquare\otimes\mathrm{id}_H)\fatsemi g)=(f\fatsemi(\mathrm{id}_M\otimes\blacksquare\otimes\mathrm{id}_N)\fatsemi(m\otimes\mathrm{id}_Y\otimes n)\fatsemi g)$.

▶ Remark 5.2. 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: $(g\otimes\mathrm{id}\otimes h)\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})=(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi(g\otimes\mathrm{id}\otimes h)$.

5.2 The Normal Produoidal Algebra of Monoidal Contexts

▶ **Definition 5.3.** *Let us endow monoidal contexts with a normal produoidal structure. The category of monoidal contexts, $\mathcal{MC}$, has as objects pairs of objects of $\mathbb{C}$. Units are defined by $\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};N\right)=\mathbb{C}(A;B)$. We use the following profunctors to define sequential splits and parallel splits,*

$$\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\triangleleft\begin{smallmatrix}X'\\Y'\end{smallmatrix}\right)=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2)\diamond\mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2;\bullet_3\otimes X'\otimes\bullet_4)\diamond\mathbb{C}(\bullet_3\otimes Y'\otimes\bullet_4;B);$$
$$\mathcal{MC}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\otimes\begin{smallmatrix}X'\\Y'\end{smallmatrix}\right)=\mathbb{C}(A;\bullet_1\otimes X\otimes\bullet_2\otimes X'\otimes\bullet_3)\diamond\mathbb{C}(\bullet_1\otimes Y\otimes\bullet_2\otimes Y'\otimes\bullet_3;B).$$

In other words, sequential splits are triples of arrows quotiented by dinaturality and written as $f\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi g\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi h$. Parallel splits are pairs of arrows quotiented by dinaturality and written as $f\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi g$. Units are simply arrows $f\colon A\to B$. Morphisms are pairs of arrows, written as $f\fatsemi(\mathrm{id}\otimes\blacksquare\otimes\mathrm{id})\fatsemi g$, and also quotiented by dinaturality. Figure 7 gives the diagrammatic representations of these components. Dinaturality for sequential splits and parallel splits is depicted in Figure 8.

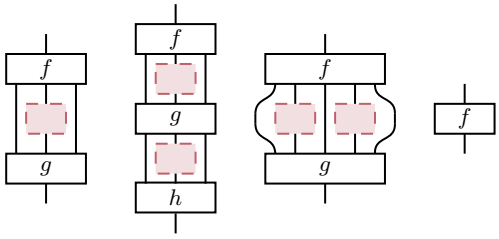


Figure 7 Morphisms, sequential and parallel splits, and units of the splice monoidal arrow produoidal category.

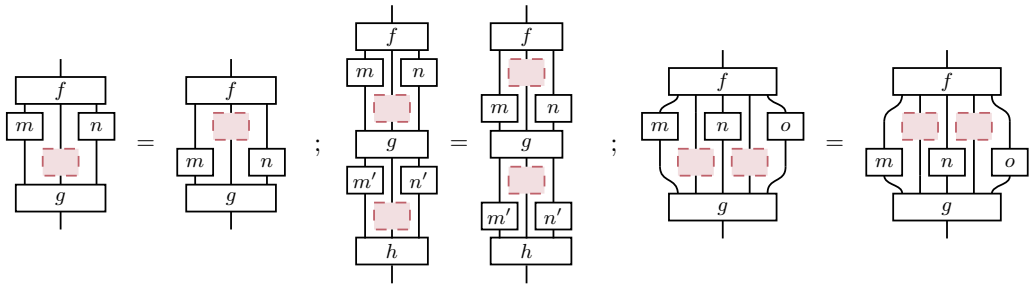


Figure 8 Dinaturality of morphisms, and sequential and parallel splits of monoidal contexts.

▶ **Theorem 5.4.** *The category of monoidal contexts forms a normal produoidal category with its units, sequential and parallel splits. 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 the full version [20, Propositions G.4 and G.5 and Theorem G.13]. ◀

6 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 [56], with applications in decision theory [27], supervised learning [17, 22] and most notably in functional data accessing [42, 53, 6, 12]. The promonoidal structure of lenses was ignored, even when, after now identifying for the first time its relation to the monoidal structure of lenses, 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.

6.1 The Normal Symmetric Produoidal Algebra of Monoidal Lenses

▶ **Definition 6.1** (Monoidal Lens). *Let* $(\mathbb{C}, \otimes, I)$ *be a symmetric monoidal category. A monoidal lens of type* $\mathcal{L}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right)$ *represents a process in a symmetric monoidal category with a hole admitting a process from* X *to* Y *[56]. Explicitly,* monoidal lenses *are the elements of the profunctor* $\mathcal{L}\mathbb{C}\left(\begin{smallmatrix}A\\B\end{smallmatrix};\begin{smallmatrix}X\\Y\end{smallmatrix}\right) = \mathbb{C}(A; \bullet \otimes X) \diamond \mathbb{C}(\bullet \otimes Y; B)$ *over* $\mathbb{C}^{op} \times \mathbb{C}$.

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 $f \fatsemi (\mathrm{id}_M \otimes \blacksquare) \fatsemi g \in \mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \\ Y \end{smallmatrix}\right)$.

▶ **Theorem 6.2.** *Monoidal lenses form a normal symmetric produoidal category with the units given by* $\mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; N\right) = \mathbb{C}(A; B)$*, and the following sequential and parallel splits.*

$$\mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \triangleleft X' \\ Y \end{smallmatrix}\right) = \mathbb{C}(A; \bullet_1 \otimes X) \diamond \mathbb{C}(\bullet_1 \otimes Y; \bullet_2 \otimes X') \diamond \mathbb{C}(\bullet_2 \otimes Y'; B);$$
$$\mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \otimes X' \\ Y \end{smallmatrix}\right) = \mathbb{C}(A; \bullet_1 \otimes X \otimes X') \diamond \mathbb{C}(\bullet_1 \otimes Y \otimes Y'; B).$$

Monoidal lenses are the free symmetric normalization of the cofree symmetric produoidal category over a symmetric monoidal category.

Proof. See the full version [20, Proposition H.1 and Theorem H.9]. ◀

▶ Remark 6.3 (Representable parallel structure). The parallel splitting structure of monoidal lenses is representable, $\mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \otimes X' \\ Y \end{smallmatrix}\right) = \mathcal{LC}\left(\begin{smallmatrix} A \\ B \end{smallmatrix}; \begin{smallmatrix} X \otimes X' \\ Y \otimes Y' \end{smallmatrix}\right)$. Lenses over a symmetric monoidal category are known to be monoidal [56, 30], 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 lenses provide a representable tensor.

▶ Remark 6.4 (Session notation for lenses). We will write $!A = \left(\begin{smallmatrix} A \\ I \end{smallmatrix}\right)$ and $?B = \left(\begin{smallmatrix} I \\ B \end{smallmatrix}\right)$ 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 $!A \otimes ?B = \left(\begin{smallmatrix} A \\ B \end{smallmatrix}\right)$.

▶ **Proposition 6.5.** *Let* $(\mathbb{C}, \otimes, I)$ *be a symmetric monoidal category. There exist monoidal functors* $(!)\colon \mathbb{C} \to \mathcal{LC}$ *and* $(?)\colon \mathbb{C}^{op} \to \mathcal{LC}$*. Moreover, they satisfy the following properties definitionally:* $\mathbb{C}(\bullet; ?A \triangleleft ?B) \cong \mathbb{C}(\bullet; ?A \otimes ?B)$*;* $!(A \otimes B) = !A \otimes !B$*;* $\mathbb{C}(\bullet; !A \triangleleft !B) \cong \mathbb{C}(\bullet; !A \otimes !B)$*;* $?(A \otimes B) = ?A \otimes ?B$*; and* $\mathbb{C}(\bullet; !A \triangleleft ?B) \cong \mathbb{C}(\bullet; !A \otimes ?B)$.

Proof. See the full version [20, Proposition H.7]. ◀

6.2 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,

$$\text{💻} \in \mathcal{LC}\left(\begin{smallmatrix} \text{Client} \\ \text{Client} \end{smallmatrix}; !\text{Msg} \triangleleft ?\text{Msg} \triangleleft !\text{Msg}\right); \qquad \text{🖥} \in \mathcal{LC}\left(\begin{smallmatrix} \text{Server} \\ \text{Server} \end{smallmatrix}; ?\text{Msg} \triangleleft !\text{Msg} \triangleleft ?\text{Msg}\right).$$

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 $(!\text{Msg} \otimes ?\text{Msg}) \triangleleft (?\text{Msg} \otimes !\text{Msg}) \triangleleft (!\text{Msg} \otimes ?\text{Msg})$, and we finally apply the unitors to fill the communication holes with noisy channels of type $\text{Msg} \to \text{Msg}$.

$$\psi_2\left(\text{💻} \otimes \text{🖥}\right) \prec^3_\lambda \texttt{NOISE}^3 \in \mathcal{LC}\left(\begin{smallmatrix} \text{Client} \otimes \text{Server}, \\ \text{Client} \otimes \text{Server} \end{smallmatrix}; N\right).$$

We end up obtaining the protocol as a single morphism $\text{Client} \otimes \text{Server} \to \text{Client} \otimes \text{Server}$ 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.

▶ Remark 6.6. Let () $= (\mathtt{SYN} \fatsemi (\mathrm{id} \otimes \blacksquare) \fatsemi \mathtt{ACK} \fatsemi (\mathrm{id} \otimes \blacksquare))$. The equalities in Figure 1 are a consequence of dinaturality over $\mathtt{PRJ}$, which acts as the interchange law for incomplete morphisms.

$$\mathtt{SYN} \fatsemi (\mathrm{id} \otimes \blacksquare) \fatsemi \mathtt{ACK} \fatsemi (\mathrm{id} \otimes \blacksquare) = \mathtt{SYN}^* \fatsemi (\mathtt{PRJ} \otimes \mathrm{id}) \fatsemi \blacksquare \fatsemi \mathtt{ACK} \fatsemi (\mathrm{id} \otimes \blacksquare) =$$
$$\mathtt{SYN}^* \fatsemi (\mathrm{id} \otimes \blacksquare) \fatsemi (\mathtt{PRJ} \otimes \mathrm{id}) \fatsemi \mathtt{ACK} \fatsemi (\mathrm{id} \otimes \blacksquare) = \mathtt{SYN}^* \fatsemi (\mathrm{id} \otimes \blacksquare) \fatsemi \mathtt{ACK}^* \fatsemi (\mathrm{id} \otimes \blacksquare).$$

7 Conclusions

Monoidal contexts are an algebra of incomplete processes, commonly generalizing lenses [56] and spliced arrows [45]. 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 [1] 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 [62, 34], 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 model a notion as concrete as process composition.

7.1 Further Work

Dependencies. Shapiro and Spivak [62] 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 (see the full version for details [20]). Weakening dependence categories in this way combines the ideas of Shapiro and Spivak [62] with those of Hefford and Kissinger [32], who employ virtual objects to deal with the non-existence of tensor products in models of spacetime.

Language theory. Melliès and Zeilberger [45] used a multicategorical form of splice-contour adjunction to give a novel proof of the Chomsky-Schützenberger representation theorem, generalized to context-free languages in categories. Our produoidal splice-contour adjunction (Section 3), combined with recent work on languages of morphisms in monoidal categories [21] opens the way for a monoidal version of the Chomsky-Schützenberger theorem.

String diagrams for concurrency. Nester et al. [48, 7] have recently introduced an alternative description of lenses in terms of *proarrow equipments*, which have a good 2-dimensional syntax [47] we can use for send/receive types (**!**/**?**). 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*.

References

1 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. doi:10.1016/j.entcs.2005.12.075.

2 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. doi:10.1016/B978-0-444-52869-8.50010-4.

3 Marcelo Aguiar, Mariana Haim, and Ignacio López Franco. Monads on higher monoidal categories. *Applied Categorical Structures*, 26(3):413–458, June 2018. doi:10.1007/s10485-017-9497-8.

4 Bruce Bartlett, Christopher L. Douglas, Christopher J. Schommer-Pries, and Jamie Vicary. Modular categories as representations of the 3-dimensional bordism 2-category, 2015. arXiv:1509.06811.

5 Jean Bénabou. Distributors at work. *Lecture notes written by Thomas Streicher*, 11, 2000.

6 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.

7 Guillaume Boisseau, Chad Nester, and Mario Román. Cornering optics. In *Proceedings Fifth International Conference on Applied Category Theory, ACT 2022, Glasgow, United Kingdom, 18-22 July 2022*, volume abs/2205.00842, 2022. doi:10.48550/arXiv.2205.00842.

8 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. doi:10.1109/LICS.2019.8785877.

9 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. doi:10.4230/LIPIcs.CSL.2018.13.

10 Thomas Booker and Ross Street. Tannaka duality and convolution for duoidal categories. *Theory and Applications of Categories*, 28(6):166–205, 2013.

11 Kenta Cho and Bart Jacobs. Disintegration and Bayesian Inversion via String Diagrams. *Mathematical Structures in Computer Science*, pages 1–34, March 2019. doi:10.1017/S0960129518000488.

12 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. arXiv:2001.07488.

13 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.

14 J. Robin B. Cockett and Craig A. Pastro. The logic of message-passing. *Sci. Comput. Program.*, 74(8):498–533, 2009. doi:10.1016/j.scico.2007.11.005.

15 J. Robin B. Cockett and Robert A. G. Seely. Weakly distributive categories. *Journal of Pure and Applied Algebra*, 114(2):133–173, 1997.

16 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.

17 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.

18 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. doi:10.1007/BFb0060438.

19 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.

20 Matt Earnshaw, James Hefford, and Mario Román. The produoidal algebra of process decomposition, 2023. arXiv:2301.11867.

21 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. doi:10.4230/LIPIcs.MFCS.2022.44.

22 Brendan Fong and Michael Johnson. Lenses and learners. *arXiv preprint*, 2019. arXiv:1903.03671.

23 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.

24 Tobias Fritz. A synthetic approach to Markov kernels, conditional independence and theorems on sufficient statistics. *Advances in Mathematics*, 370:107239, 2020. arXiv:1908.07021.

25 Richard Garner and Ignacio López Franco. Commutativity. *Journal of Pure and Applied Algebra*, 220(5):1707–1751, 2016.

26 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. doi:10.1007/3-540-49099-X_6.

27 Neil Ghani, Jules Hedges, Viktor Winschel, and Philipp Zahn. Compositional game theory. In Anuj Dawar and Erich Grädel, editors, *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 472–481. ACM, 2018. doi:10.1145/3209108.3209165.

28 René Guitart. Tenseurs et machines. *Cahiers de topologie et géométrie différentielle catégoriques*, 21(1):5–62, 1980. URL: http://www.numdam.org/item/CTGDC_1980__21_1_5_0/.

29 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.

30 Jules Hedges. Coherence for lenses and open games. *arXiv preprint*, 2017. arXiv:1704.02230.

31 James Hefford and Cole Comfort. Coend optics for quantum combs. *arXiv preprint*, 2022. arXiv:2205.09027, doi:10.48550/ARXIV.2205.09027.

32 James Hefford and Aleks Kissinger. On the pre- and promonoidal structure of spacetime. *arXiv preprint*, 2022. doi:10.48550/arXiv.2206.09678.

33 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. doi:10.1016/j.entcs.2006.04.012.

34 Chris Heunen and Jesse Sigal. Duoidally enriched Freyd categories. *arXiv preprint*, 2023. arXiv:2301.05162.

35 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. doi:10.1007/3-540-57208-2_35.

36 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. doi:10.1145/1328438.1328472.

37 John Hughes. Generalising monads to arrows. *Science of Computer Programming*, 37(1-3):67–111, 2000. doi:10.1016/S0167-6423(99)00023-4.

38 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. doi:10.1145/2873052.

39 Michael Johnson, Robert Rosebrugh, and Richard J. Wood. Lenses, fibrations and universal translations. *Mathematical structures in computer science*, 22(1):25–42, 2012.

40 André Joyal and Ross Street. The geometry of tensor calculus, I. *Advances in Mathematics*, 88(1):55–112, 1991. doi:10.1016/0001-8708(91)90003-P.

41 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. doi:10.1109/LICS.2017.8005095.

42 Edward Kmett. lens library, version 4.16. *Hackage https://hackage. haskell. org/package/lens-4.16*, 2018, 2012.

43 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. doi:10.1145/237721.237804.

44 Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. doi:10.1017/9781108778657.

45 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.

46 Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991. doi:10.1016/0890-5401(91)90052-4.

47 David Jaz Myers. String diagrams for double categories and equipments, 2016. doi:10.48550/arXiv.1612.02762.

48 Chad Nester. Concurrent Process Histories and Resource Transducers. *Logical Methods in Computer Science*, Volume 19, Issue 1, January 2023. doi:10.46298/lmcs-19(1:7)2023.

49 Nelson Niu and David I. Spivak. Polynomial functors: A general theory of interaction. *In preparation*, 2022.

50 Craig Pastro and Ross Street. Doubles for Monoidal Categories. *arXiv preprint*, 2007. arXiv:0711.1859.

51 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.

52 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*, pages 49–64, February 2021.

53 Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. *Art Sci. Eng. Program.*, 1(2):7, 2017. doi:10.22152/programming-journal.org/2017/1/7.

54 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. doi:10.1109/LICS.1993.287570.

55 J. Postel. Transmission control protocol. RFC 793, RFC Editor, September 1981. doi:10.17487/RFC0793.

56 Mitchell Riley. Categories of Optics. *arXiv preprint*, 2018. arXiv:1809.00738.

57 Mario Román. Comb Diagrams for Discrete-Time Feedback. *CoRR*, abs/2003.06214, 2020. arXiv:2003.06214.

58 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. doi:10.48550/arXiv.2205.07664.

59 Mario Román. Open diagrams via coend calculus. *Electronic Proceedings in Theoretical Computer Science*, 333:65–78, February 2021. doi:10.4204/eptcs.333.5.

60 Davide Sangiorgi and David Walker. *The Pi-Calculus – A theory of mobile processes*. Cambridge University Press, 2001.

61 Patrick Schultz, David I. Spivak, and Christina Vasilakopoulou. Dynamical systems and sheaves. *Applied Categorical Structures*, 28(1):1–57, 2020.

62 Brandon T. Shapiro and David I. Spivak. Duoidal structures for compositional dependence. *arXiv preprint*, 2022. arXiv:2210.01962.

63 Michael Shulman. Categorical logic from a categorical point of view. *Available on the web*, 2016. URL: https://mikeshulman.github.io/catlog/catlog.pdf.

64 Michael Shulman. Duoidal category (nlab entry), section 2, 2017. , Last accessed on 2022-12-14. URL: https://ncatlab.org/nlab/show/duoidal+category.

65 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. arXiv:1305.0297.

66 Ross Street. Monoidal categories in, and linking, geometry and algebra. *Bulletin of the Belgian Mathematical Society-Simon Stevin*, 19(5):769–820, 2012.

67 André Videla and Matteo Capucci. Lenses for composable servers. *CoRR*, abs/2203.15633, 2022. doi:10.48550/arXiv.2203.15633.