Compositionality
http://compositionality-journal.org
ISSN 2631-4444
Volume 6, Issue 1 (2024)

Profunctor Optics: a Categorical Update

Bryce Clarke, Derek Elkins, Jeremy Gibbons, Fosco Loregian, Bartosz Milewski, Emily Pillmore, and Mario Román

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 **Set**. 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.

Bryce Clarke: 0000-0003-0579-2804
Jeremy Gibbons: 0000-0002-8426-9917
Fosco Loregian: 0000-0003-3052-465X
Bartosz Milewski: 0000-0001-9249-0104
Mario Román: 0000-0003-3158-1226

Received 2022-03-16; accepted 2023-03-06; available from https://doi.org/10.32408/compositionality-6-1

Contents

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*, 'focusing' 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] (Figure 1).

```
data Address = Address
  { street'  :: String
  , town'    :: String
  , country' :: String }

viewStreet :: Address -> String
viewStreet = street'

updateStreet :: Address -> String -> Address
updateStreet a s = a {street' = s}

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 s t c) = s ++ ", " ++ t ++ ", " ++ c

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 of failure on pattern matching. In the figure, we verify whether a string can be parsed as an address. The combinator `(?.)` returns the results using the `Maybe` monad (see §6.3.1).

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 [27] 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, and various other representations of lenses, were originally proposed in functional programming as a compositional solution to the problem of accessing fields of nested data structures [10, 41].

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 polymorphic function composition. Together, all *families of 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 [31, §3]. Milewski [25] identified these algebraic structures (cartesian profunctors, cocartesian profunctors, ...) as *Tambara modules* [40] and used a result by Pastro and Street [30] to propose a unified definition of optic. This definition has been later extended by Boisseau and Gibbons [3] and Riley [34], 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* [34, §6.1] and *enriched optics*.

The 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 [25] 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.

Definition 1.1 (Ends and coends)**.** 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.

$$\mathrm{end}(P) \coloneqq \mathrm{eq}\left(\prod_{X \in \mathbf{C}} P(X, X) \rightrightarrows \prod_{f \colon A \to B} P(A, B) \right),$$

$$\mathrm{coend}(P) \coloneqq \mathrm{coeq}\left(\coprod_{f \colon B \to A} P(A, B) \rightrightarrows \coprod_{X \in \mathbf{C}} P(X, X) \right).$$

Ends are usually denoted with a subscripted integral; *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 $(\mathbb{R}^n)^*$. An even 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, 22]. 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. *Evaluation on the identity defines the following isomorphisms, called* Yoneda *and* coYoneda *reductions, respectively.*

$$\int_{X \in \mathbf{C}} \mathcal{V}(\mathbf{C}(A, X), FX) \cong FA. \qquad \int^{X \in \mathbf{C}} \mathbf{C}(X, A) \otimes FX \cong FA,$$

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. *The* Fubini rule *is satisfied up to isomorphism.*

$$\int_{X_1 \in \mathbf{C}} \int_{X_2 \in \mathbf{C}} P(X_1, X_2, X_1, X_2) \cong \int_{X_2 \in \mathbf{C}} \int_{X_1 \in \mathbf{C}} P(X_1, X_2, X_1, X_2).$$

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

Proposition 1.4. Continuity *and* cocontinuity *induce the following isomorphisms.*

$$\mathcal{V}\left(A, \int_{X \in \mathbf{C}} P(X,X)\right) \cong \int_{X \in \mathbf{C}} \mathcal{V}(A, P(X,X)).$$

$$\mathcal{V}\left(\int^{X \in \mathbf{C}} P(X,X), A\right) \cong \int_{X \in \mathbf{C}} \mathcal{V}(P(X,X), A).$$

Proposition 1.5. *The set of* natural transformations *can be rewritten as a coend.*

$$\int_{X \in \mathbf{C}} \mathbf{D}(FX, GX) \cong [\mathbf{C}, \mathbf{D}](F, G)$$

In particular, coend calculus expressions are commonly simplified using *adjunctions* ($F \dashv G$): an adjunction is equivalently an isomorphism $\mathbf{D}(FX, Y) \cong \mathbf{C}(X, GY)$.

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, 34].

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 [38, §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 [30] that had been used to define categories of profunctor optics [3, 34] to *generalized Tambara module*. 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 5 contains concluding remarks. The Appendix (Section 6) 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 $\mathcal{V}(A, B)$. 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 [22]. 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 [34, §2] notes, this will not be a problem in general: even if some coends are indexed by large categories and we cannot argue their existence using the cocompleteness of **Set**, we will still find them represented by objects in the category.

Acknowledgements

This work was started in the last author's MSc thesis [36]; development continued at the Applied Category School 2019 at Oxford [33], and we thank the organizers of the School for that opportunity. We also thank Paweł Sobociński for discussion, and the anonymous reviewers for suggestions about previous versions of this manuscript. The code for this text has been continued as a Haskell library [32].

Bryce Clarke is 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).

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 that splits some data structure of type S into something of the form $M \times A$ and then recombines it back to S. Here, A is the type of the field we want to focus on, and M is the type combining the remaining fields that constitute 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 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. 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.

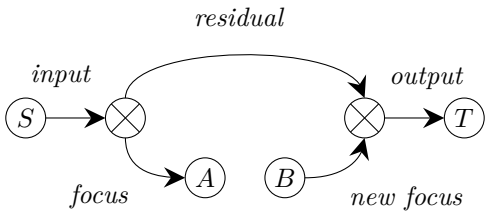


Figure 5: The common structure of an optic. We could provide semantics for diagram of this kind in terms of profunctors, see [37].

Multiple definitions of optics of increasing generality have been given in [3, 25, 34]. 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 $(Ⓛ)\colon \mathbf{M} \to [\mathbf{C}, \mathbf{C}]$ and $(Ⓡ)\colon \mathbf{M} \to [\mathbf{D}, \mathbf{D}]$; and let us write

$$\phi^L_A \colon A \cong I \,Ⓛ\, A, \qquad \phi^L_{M,N,A} \colon M \,Ⓛ\, N \,Ⓛ\, A \cong (M \otimes N) \,Ⓛ\, A,$$
$$\phi^R_B \colon B \cong I \,Ⓡ\, B, \qquad \phi^R_{M,N,B} \colon M \,Ⓡ\, N \,Ⓡ\, B \cong (M \otimes N) \,Ⓡ\, B,$$

for the structure isomorphisms of the 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 $(⓵, ®)$**-optic** from (S, T) with the focus on (A, B) is a (generalized) element of the following object described as a coend:

$$\mathbf{Optic}_{⓵,®}((A,B),(S,T)) \coloneqq \int^{M \in \mathbf{M}} \mathbf{C}(S, M ⓵ A) \otimes \mathbf{D}(M ® B, T).$$

The two monoidal actions ⓵ 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 bullet represents some monoidal action; the brackets represent some monoidal action from the opposite category.

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 can be seen 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, 28, 29] has been generalized in many different directions. *Monadic lenses* [1], *lenses in a symmetric monoidal category* [38, §2.2], *linear lenses* [34, §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 that captures all of these variants.

We present 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 [25] 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* [34, §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 $(\bullet)\colon \mathbf{C} \times \mathbf{D} \to \mathbf{D}$. That is,* $\mathbf{Lens} \cong \mathbf{Optic}_{(\times,\bullet)}$.

Proof. The universal property of the product 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. A **monoidal lens** [38, §2.2] 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 (Fox shows a stronger result [11]). By Proposition 3.2, we can show $\mathbf{MonLens}_{\otimes} \cong \mathbf{Optic}_{(\otimes,\bullet)}$ where $(\bullet)$ is given by $S \bullet A \coloneqq \mathcal{U}S \otimes A$. □

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*: this is because a functor $F\colon \mathcal{W} \to \mathcal{W}$ with a strength is the same thing as a $\mathcal{W}$-enrichment of the functor [21]. 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 [34, §4.9] to describe a non-mixed variant called *effectful lenses*.

```
stamp :: MonadicLens IO a b (Timestamped a) (Timestamped b)
stamp = mkMonadicLens @IO viewContents updateContentsAndStamp
   where
     viewContents :: Timestamped a -> a
     viewContents = contents'

     updateStamp :: Timestamped a -> b -> IO (Timestamped b)
     updateStamp x b = do
       currentTime <- getCurrentTime
       return (x {contents' = b , modified' = currentTime})

greeting :: IO (Timestamped String)
greeting = do
  t <- getCurrentTime
  x <- pure (Timestamped t t "What is the answer?")
  threadDelay (2500000) -- microseconds
  x & stamp .! "42"

 >> greeting
 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-variant monadic lenses for the `IO` monad is used to track how a data holder (`Timestamped`) is accessed 2.5 seconds after creation.

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.

Definition 3.8. Let $\Psi\colon \mathbf{C} \to \mathbf{C}$ be a $\mathcal{W}$-monad on a cartesian $\mathcal{W}$-category $\mathbf{C}$. Let $(\bullet)\colon \mathbf{C} \times \mathbf{D} \to \mathbf{D}$ be a monoidal $\mathcal{W}$-action on 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* ${}_{\mathcal{U}}(\bullet)\colon \mathrm{EM}_\Psi \times \mathbf{D} \to \mathbf{D}$. *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; $\mathcal{U}(\bullet) \times \bullet$ defines a monoidal action. The forgetful $\mathcal{U}$ has a left adjoint $\mathcal{F}^\Psi$ such that $\mathcal{U} \circ \mathcal{F}^\Psi = \Psi$.

$$
\begin{aligned}
& \int^{C \in \mathrm{EM}_\Psi} \mathbf{C}(S, \mathcal{U}C \times A) \times \mathbf{D}(\mathcal{U}C \bullet B, T) \\
\cong\ & \{\text{Adjunction } (\times) \dashv \Delta\} \\
& \int^{C \in \mathrm{EM}_\Psi} \mathbf{C}(S, \mathcal{U}C) \times \mathbf{C}(S, A) \times \mathbf{D}(\mathcal{U}C \bullet B, T) \\
\cong\ & \{\text{Adjunction } \mathcal{F}^\Psi \dashv \mathcal{U}\} \\
& \int^{C \in \mathrm{EM}_\Psi} \mathrm{EM}_\Psi(\mathcal{F}^\Psi S, C) \times \mathbf{C}(S, A) \times \mathbf{D}(\mathcal{U}C \bullet B, T) \\
\cong\ & \{\text{Coyoneda}\} \\
& \mathbf{C}(S, A) \times \mathbf{D}(\Psi S \bullet B, T).
\end{aligned}
$$

□

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 [34, §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 ([34, §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.

$$\begin{aligned} & \int^{D \in \mathbf{D}} \mathbf{C}(S, D \bullet A) \times \mathbf{D}(D + B, T) \\ \cong \quad & \{\text{Adjunction } (+) \dashv \Delta\} \\ & \int^{D \in \mathbf{D}} \mathbf{C}(S, D \bullet A) \times \mathbf{D}(D, T) \times \mathbf{D}(B, T) \\ \cong \quad & \{\text{Coyoneda}\} \\ & \mathbf{C}(S, T \bullet A) \times \mathbf{D}(B, T). \end{aligned}$$

□

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 finitary container [27] 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 %
 [ "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 coproducts. A **traversal** is an element of

$$\mathbf{Traversal}((A, B), (S, T)) \coloneqq \mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes [B^n, T]\right).$$

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).

$$\begin{aligned}
&\int^{X \in [\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes X_n\right) \otimes \mathbf{C}\left(\int^{n \in \mathbb{N}} B^n \otimes X_n, T\right) \\
\cong\ & \{\text{Continuity}\} \\
&\int^{X \in [\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes X_n\right) \otimes \int_{n \in \mathbb{N}} \mathbf{C}\left(X_n \otimes B^n, T\right) \\
\cong\ & \{\text{Adjunction } (- \otimes B^n) \dashv [B^n, -]\} \\
&\int^{X \in [\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes X_n\right) \otimes \int_{n \in \mathbb{N}} \mathbf{C}\left(X_n, [B^n, T]\right) \\
\cong\ & \{\text{Natural transformation}\} \\
&\int^{X \in [\mathbb{N},\mathbf{C}]} \mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes X_n\right) \otimes [\mathbb{N}, \mathbf{C}]\left(X_{(-)}, [B^{(-)}, T]\right) \\
\cong\ & \{\text{Coyoneda}\} \\
&\mathbf{C}\left(S, \int^{n \in \mathbb{N}} A^n \otimes [B^n, T]\right).
\end{aligned}$$

□

The derivation from the general definition of optic to the concrete description of lenses and prisms in functional programming was first described by [31] and [25], 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, 34]. For a derivation using Yoneda, [34] 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 [36, §4.2].

In the **Sets**-based case, the relation between traversable functors, applicative functors [24] 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 monoidal action thanks to the fact that two power series functors compose into a power series functor.

$$\begin{aligned}
&\int^{m \in \mathbb{N}} \left(\int^{n \in \mathbb{N}} A^n \otimes C_n\right)^m \otimes D_m \\
\cong\ & \{\text{Product distributes over colimits}\} \\
&\int^{m} \int^{n_1, \ldots, n_m} A^{n_1} \otimes \cdots \otimes A^{n_m} \otimes C_{n_1} \otimes \cdots \otimes C_{n_m} \otimes D_m \\
\cong\ & \{\text{Rearranging terms}\}
\end{aligned}$$

$$\int^{k\in\mathbb{N}} A^k \otimes \left(\sum_{n_1+\cdots+n_m=k} C_{n_1} \otimes \cdots \otimes C_{n_m} \otimes D_m \right).$$

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.

$$\int^{m}\int^{n_1,\ldots,n_m} \mathbb{N}(n_1+\cdots+n_m,k)\cdot C_{n_1}\otimes\cdots\otimes C_{n_m}\otimes D_m.$$

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

$$\mathbf{Affine}_{\otimes}\left((A,B),(S,T)\right) := \mathbf{C}(S, T + A\otimes[B,T]).$$

Proposition 3.25. *Affine traversals are optics (as in Definition 3.1) for the action* $(+\otimes)\colon \mathbf{C}^2\to[\mathbf{C},\mathbf{C}]$ *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 [23, §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.

$$\begin{aligned}
&\int^{F\in\mathbf{App}} \mathcal{V}(S,\mathcal{U}FA)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Yoneda}\}\\
&\int^{F\in\mathbf{App}} \mathcal{V}\left(S,\int_{C\in\mathcal{V}}\left[[A,C],\mathcal{U}FC\right]\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Continuity}\}\\
&\int^{F\in\mathbf{App}}\int_{C}\mathcal{V}\left(S,\left[[A,C],\mathcal{U}FC\right]\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Adjunction } ([A,C]\otimes -)\dashv\left[[A,C],-\right]\}\\
&\int^{F\in\mathbf{App}}\left(\int_{C}\mathcal{V}\left([A,C]\otimes S,\mathcal{U}FC\right)\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Natural transformation}\}\\
&\int^{F\in\mathbf{App}}[\mathcal{V},\mathcal{V}]\left([A,-]\otimes S,\mathcal{U}F\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Adjunction of free applicatives}\}\\
&\int^{F\in\mathbf{App}}\mathbf{App}\left(\sum_{n\in\mathbb{N}}[A^n,-]\otimes S^n,F\right)\otimes\mathcal{V}(\mathcal{U}FB,T)\\
\cong\ &\{\text{Coyoneda}\}\\
&\mathcal{V}\left(\sum_{n\in\mathbb{N}}S^n\otimes[A^n,B],T\right)\\
\cong\ &\{\text{Continuity}\}\quad\{\text{Adjunction } (S^n\otimes -)\dashv[S^n,-]\}\\
&\prod_{n\in\mathbb{N}}\mathcal{V}\left([A^n,B],[S^n,T]\right).
\end{aligned}$$

□

Remark 3.28. The free applicative we construct here is known in Haskell programming as the `FunList` applicative [42]. 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 $(\bullet)\colon \mathbf{C}\times\mathbf{D}\to\mathbf{D}$ be an arbitrary action. A **grate** is an element of

$$\mathbf{Grate}\left((A,B),(S,T)\right) := \mathbf{D}([S,A]\bullet B, T).$$

Proposition 3.30. *Grates are mixed optics for the action of the exponential and* $(\bullet)\colon \mathbf{C}\times\mathbf{D}\to\mathbf{D}$. *That is,* $\mathbf{Grate} \cong \mathbf{Optic}_{[\ ,\ ],\bullet}$.

Proof. We know from [7, Proposition 1.2] that the adjunction $[-,A]^{op} \dashv [-,A]$ holds in any symmetric monoidal category.

$$\begin{aligned}
&\int^{C\in\mathbf{C}} \mathbf{C}(S,[C,A])\otimes\mathbf{D}(C\bullet B,T)\\
\cong\ &\{\text{Adjunction } [-,A]^{op}\dashv[-,A]\}\\
&\int^{C\in\mathbf{C}} \mathbf{C}(C,[S,A])\otimes\mathbf{D}(C\bullet B,T)\\
\cong\ &\{\text{Coyoneda}\}\\
&\mathbf{D}([S,A]\bullet B,T)
\end{aligned}$$

□

The description of a grate and its profunctor representation in terms of "closed" profunctors was first reported by Deikun and O'Connor [26]; 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

$$\mathbf{Glass}\left((A,B),(S,T)\right) := \mathbf{C}(S\times[[S,A],B],T).$$

Proposition 3.32. *Glasses are optics for the action* $(\bullet\times[\bullet,-])\colon \mathbf{C}^{op}\times\mathbf{C}\to[\mathbf{C},\mathbf{C}]$ *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, **Foldable**, 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 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 and here, $\mathcal{L}$ is 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

We finish our examples of optics with two extremes: the actions from the initial and terminal actions. These are the identity action $\mathrm{id} \colon [\mathcal{V}, \mathcal{V}] \to [\mathcal{V}, \mathcal{V}]$ and picking the monoidal unit, $\mathrm{I} \colon \mathbf{1} \to [\mathcal{V}, \mathcal{V}]$.

Definition 3.35. A setter [20] is an element of

$$
\mathbf{Setter}((A, B), (S, T)) \coloneqq \mathcal{V}([A, B], [S, T]).
$$

Proposition 3.36. *Setters are optics for identity action* $\mathrm{id}\colon [\mathcal{V},\mathcal{V}] \to [\mathcal{V},\mathcal{V}]$. *That is,* **Setter** $\cong$ $\mathbf{Optic}_{\mathrm{id},\mathrm{id}}$.

Proof. The concrete derivation is described by Riley [34, §4.5.2]. There, it is required that functors are strong; this requirement holds automatically for every enriched functor in the base of enrichment [21].

$$
\begin{aligned}
& \int^{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(S,FA)\otimes\mathcal{V}(FB,T) \\
\cong\ & \{\text{Yoneda}\} \\
& \int^{F\in[\mathcal{V},\mathcal{V}]} \left(\int_{C\in\mathcal{V}} [\mathcal{V}(A,C),\mathcal{V}(S,FC)]\right)\otimes\mathcal{V}(FB,T) \\
\cong\ & \{\text{Copower}\} \\
& \int^{F\in[\mathcal{V},\mathcal{V}]} \left(\int_{C\in\mathcal{V}} \mathcal{V}(S\otimes[A,C],FC)\right)\otimes\mathcal{V}(FB,T) \\
\cong\ & \{\text{Natural transformation}\} \\
& \int^{F\in[\mathcal{V},\mathcal{V}]} [\mathcal{V},\mathcal{V}](S\otimes[A,-],F)\otimes\mathcal{V}(FB,T) \\
\cong\ & \{\text{Coyoneda}\} \\
& \mathcal{V}(S\otimes[A,B],T) \\
\cong\ & \{\text{Adjunction } (S\otimes -)\dashv[S,-]\} \\
& \mathcal{V}([A,B],[S,T]).
\end{aligned}
$$

□

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.

$$
\begin{aligned}
\mathcal{V}(S,HA) &\cong [\mathcal{V},\mathcal{V}](\mathsf{Lan}_A S,H),\\
\mathcal{V}(HB,T) &\cong [\mathcal{V},\mathcal{V}](H,\mathsf{Ran}_B T).
\end{aligned}
$$

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 ([36, §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.

$$
\begin{aligned}
& \int^{M \in \mathbf{M}} \mathcal{V}(S, \mathcal{U}MA) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong \quad & \{\text{Kan extension}\} \\
& \int^{M \in \mathbf{M}} [\mathcal{V}, \mathcal{V}]\left(\mathsf{Lan}_A S, \mathcal{U}M\right) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong \quad & \{\text{Adjunction } \mathcal{L} \dashv \mathcal{U}\} \\
& \int^{M \in \mathbf{M}} \mathbf{M}\left(\mathcal{L}\mathsf{Lan}_A S, M\right) \otimes \mathcal{V}(\mathcal{U}MB, T) \\
\cong \quad & \{\text{Coyoneda}\} \\
& \mathcal{V}\left(\mathcal{U}\mathcal{L}\mathsf{Lan}_A S(B), T\right).
\end{aligned}
$$

□

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 [31, §3]. Milewski [25] 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 [40]. 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 [30, §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 [30, 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 [25] already proposed a unified description of optics, later extended by [3] and [34], 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 $\otimes_{\mathcal{A}} \colon \mathcal{A} \otimes \mathcal{A} \to \mathcal{A}$ needs to be substituted by a pair of arbitrary monoidal actions Ⓛ: $\mathbf{M} \otimes \mathbf{C} \to \mathbf{C}$ and Ⓡ: $\mathbf{M} \otimes \mathbf{D} \to \mathbf{D}$, from a common monoidal category $\mathbf{M}$.

This section can be seen both as a partial exposition of one of the main results of the work of Pastro and Street [30, Proposition 6.1] and a generalization of their definitions and propositions to the setting that is most useful for applications in functional programming.

[1] Double, as used by Pastro and Street [30], 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, a tensor product is a profunctor $P \colon \mathcal{A}^{op} \otimes \mathcal{A} \otimes \mathcal{A} \to \mathcal{V}$ and the unit is $J \colon \mathcal{A}^{op} \to \mathcal{V}$

4.1 Generalized Tambara modules

Originally, *Tambara modules* [40] 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

$$P(\phi^L_A,\phi^{-1R}_B)\circ\alpha_{I,A,B}=\mathrm{id},$$
$$P(\phi_{M,N,A},\phi^{-1}_{M,N,B})\circ\alpha_{M\otimes N,A,B}=\alpha_{N,A,B}\circ\alpha_{M,N Ⓒ A,N Ⓡ B},$$

for every $M,N\in\mathbf{M}$, every $A\in\mathbf{C}$ and every $B\in\mathbf{D}$. Let us repeat the equations in terms of commutative diagrams for clarity. This is a family of morphisms making the following two diagrams commute.

$$\begin{array}{ccc} P(A,B) & \xrightarrow{\alpha_{M\otimes N,A,B}} & P((M\otimes N) Ⓛ A,(M\otimes N) Ⓡ B)\\ {\scriptstyle\alpha_{N,A,B}}\downarrow & & \downarrow{\scriptstyle P(\phi_{M,N,A},\varphi^{-1}_{M,N,B})}\\ P(N Ⓛ A, N Ⓡ B) & \xrightarrow[\alpha_{M,N Ⓒ A,N Ⓒ B}]{} & P(M Ⓛ N Ⓛ A, M Ⓡ N Ⓡ B)\end{array}$$

$$\begin{array}{ccc} P(A,B) & \xrightarrow{\alpha_{I,A,B}} & P(I Ⓛ A, I Ⓡ B)\\ & \searrow^{\mathrm{id}} & \downarrow{\scriptstyle P(\phi_A,\varphi^{-1}_B)}\\ & & P(A,B)\end{array}$$

When $\mathcal{V}=\mathbf{Set}$, we can define a morphism of Tambara modules as a natural transformation $\eta_{A,B}\colon P(A,B)\to Q(A,B)$ satisfying

$$\eta_{M Ⓒ A,M Ⓡ B}\circ\alpha_{M,A,B}=\alpha'_{M,A,B}\circ\eta_{A,B}.$$

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 equalizer of the arrows

$$\int_{X,Y}\mathcal{V}(P(X,Y),Q(X,Y))$$
$$\xrightarrow{\pi_{A,B}}\mathcal{V}(P(A,B),Q(A,B))$$
$$\xrightarrow{\mathcal{V}(\mathrm{id},\alpha'_{A,B})}\mathcal{V}(P(A,B),Q(M Ⓛ A,M Ⓡ B))$$

and

$$\begin{aligned}&\int_{X,Y} \mathcal{V}(P(X,Y),Q(X,Y))\\ &\xrightarrow{\pi_{M\circledcirc A,M\circledcirc B}} \mathcal{V}(P(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B), Q(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B))\\ &\xrightarrow{\mathcal{V}(\alpha_{A,B},\mathrm{id})} \mathcal{V}(P(A,B), Q(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B))\end{aligned}$$

for each $A \in \mathbf{C}$, $B \in \mathbf{D}$ and $M \in \mathbf{M}$ [30, §3.2].

Remark 4.2. Pastro and Street [30] 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 [30, §5] in profunctors. 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).

Remark 4.3. Before the definition, let us recall the axioms for a 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 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.

$$\begin{array}{ccccc} I \oslash M \oslash A & \xleftarrow{\phi_{M\oslash A}} & M \oslash A & \xrightarrow{M\oslash\phi_A} & M \oslash I \oslash A \\ \downarrow{\scriptstyle \phi_{I,M,A}} & & \downarrow{\scriptstyle \mathrm{id}} & & \downarrow{\scriptstyle \phi_{M,I,A}} \\ (I \otimes M) \oslash A & \xrightarrow[\lambda_M\oslash A]{} & M \oslash A & \xleftarrow[\rho_M\oslash A]{} & (M \otimes I) \oslash A \end{array}$$

$$\begin{array}{ccc} M \oslash N \oslash K \oslash A & \xrightarrow{\mathrm{id}} & M \oslash N \oslash K \oslash A \\ \downarrow{\scriptstyle \phi_{M,N,K\oslash A}} & & \downarrow{\scriptstyle M\oslash\phi_{N,K,A}} \\ (M \otimes N) \oslash K \oslash A & & M \oslash (N \otimes K) \oslash A \\ \downarrow{\scriptstyle \phi_{M\otimes N,K,A}} & & \downarrow{\scriptstyle \phi_{M,N\otimes K,A}} \\ ((M \otimes N) \otimes K) \oslash A & \xrightarrow[a_{M,N,K}\oslash A]{} & (M \otimes (N \otimes K)) \oslash A \end{array}$$

Definition 4.4. 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 ($\textcircled{L}$) and ($\textcircled{R}$) 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

$$T(M,N,A,B,P) \coloneqq P(M \mathbin{\textcircled{L}} A, N \mathbin{\textcircled{R}} B).$$

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

$$T'(A,B,P) = \int_{M\in\mathbf{M}} P(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B),$$

which can be curried into the $\mathcal{V}$-functor

$$\Theta P(A,B) \coloneqq \int_{M \in \mathbf{M}} P(M \triangleleft A, M \triangleright B).$$

Proposition 4.5. *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,*

$$\Theta P(A,B) \xrightarrow{\pi_I} P(I \triangleleft A, I \triangleright B) \xrightarrow{P(\phi_A, \phi_B^{-1R})} P(A,B).$$

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

$$\Theta P(A,B) \xrightarrow{P\left(\phi_{M,N,A}, \phi^{-1R}_{M,N,B}\right) \circ \pi_{M \otimes N}} P(M \triangleleft N \triangleleft A, M \triangleright N \triangleright B)$$

through the projection

$$\Theta\Theta P(A,B) \xrightarrow{\pi_M \circ \pi_N} P(M \triangleleft N \triangleleft A, M \triangleright N \triangleright B)$$

for every $M, N \in \mathbf{M}$.

Proof. In order to keep the diagrams in this proof manageable, we introduce the notation $P[M](A,B) \coloneqq P(M \triangleleft A, M \triangleright B)$. 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.

$$\begin{array}{ccccc}
\Theta P(A,B) & \xrightarrow{\delta_P} & \Theta\Theta P(A,B) & \xrightarrow{(\Theta\varepsilon)_P} & \Theta P(A,B) \\
 & & \downarrow{\scriptstyle \pi_M} & & \downarrow{\scriptstyle \pi_M} \\
\downarrow{\scriptstyle \pi_{I\otimes M}} & & \Theta P[M](A,B) & \xrightarrow{\varepsilon_{P[M]}} & P[M](A,B) \\
 & & \downarrow{\scriptstyle \pi_I} & & \downarrow{\scriptstyle \mathrm{id}} \\
P[I \otimes M](A,B) & \xrightarrow[P(\phi_{I,M,A}, \phi^{-1R}_{I,M,B})]{} & P[M][I](A,B) & \xrightarrow[P(\phi_{M\otimes A}, \phi^{-1R}_{M\otimes B})]{} & P[M](A,B)
\end{array}$$

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 4.3), the bottom edge of the whole diagram can be rewritten as

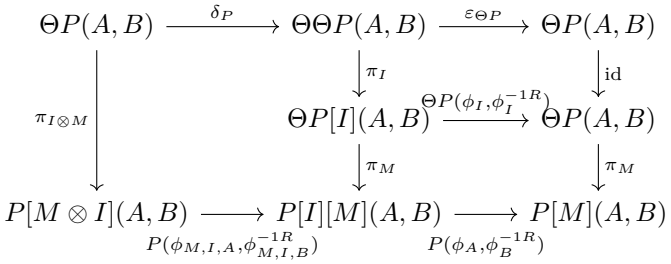
$$P(\phi_{M\otimes A}, \phi^{-1R}_{M\otimes B}) \circ P(\phi_{I,M,A}, \phi^{-1R}_{I,M,B}) = P(\lambda_M^{-1} \oslash A, \lambda_M \oslash B).$$

Now, by the wedge condition of the end, the left-bottom side of the previous diagram is just the projection π_M.

$$\begin{array}{ccccc}
 & & \Theta P(A,B) & & \\
 & \swarrow^{\pi_{I\otimes M}} & & \searrow^{\pi_M} & \\
P[I \otimes M](A,B) & & & & P[M](A,B) \\
 & \searrow_{P(\mathrm{id}, \lambda_M \oslash B)} & & \swarrow_{P(\lambda_M \oslash A, \mathrm{id})} & \\
 & & P(I \otimes M \triangleleft A, M \triangleright B) & &
\end{array}$$

Finally, by the universal property of the end, that implies that $\Theta\varepsilon_P \circ \delta_P$ must coincide with the identity.

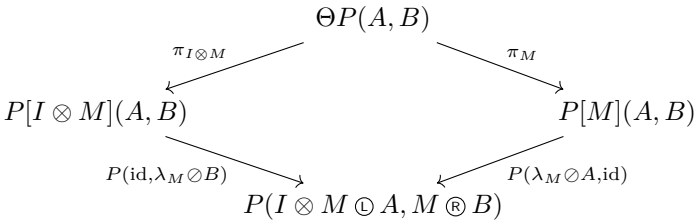
Let us now prove right counitality, $\varepsilon_{\Theta P} \circ \delta_P = \mathrm{id}_{\Theta P}$, which follows from the commutativity of the following diagram.



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 4.3), the bottom edge of the diagram can be rewritten as

$$P(\phi_A^L, \phi_B^{-1R}) \circ P(\phi_{M,I,A}^L, \phi_{M,I,B}^{-1R}) = P(\rho_M^{-1} \oslash A, \rho_M \oslash B).$$

Now, by the wedge condition of the end, the left-bottom side of the previous diagram is just the projection π_M.



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.

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 4.3). 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* **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

$$\begin{aligned}
&\mathbf{Prof}(\mathbf{C},\mathbf{D})(P,\Theta P)\\
\cong\ &\{\text{Natural transformation}\}\\
&\int_{A,B} \mathcal{V}\left(P(A,B), \int_{M\in\mathbf{M}} P(M \mathbin{⦶} A, M \mathbin{⦸} B)\right)\\
\cong\ &\{\text{Continuity}\}\\
&\int_{A,B,M} \mathcal{V}\left(P(A,B), P(M \mathbin{⦶} A, M \mathbin{⦸} B)\right)
\end{aligned}$$

whose elements can be seen as a family of morphisms that is dinatural in $M \in \mathbf{M}$ and natural in $(A,B) \in \mathbf{C}^{op} \otimes \mathbf{D}$. The two conditions in the definition of Tambara module can be rewritten as the axioms of the coalgebra. □

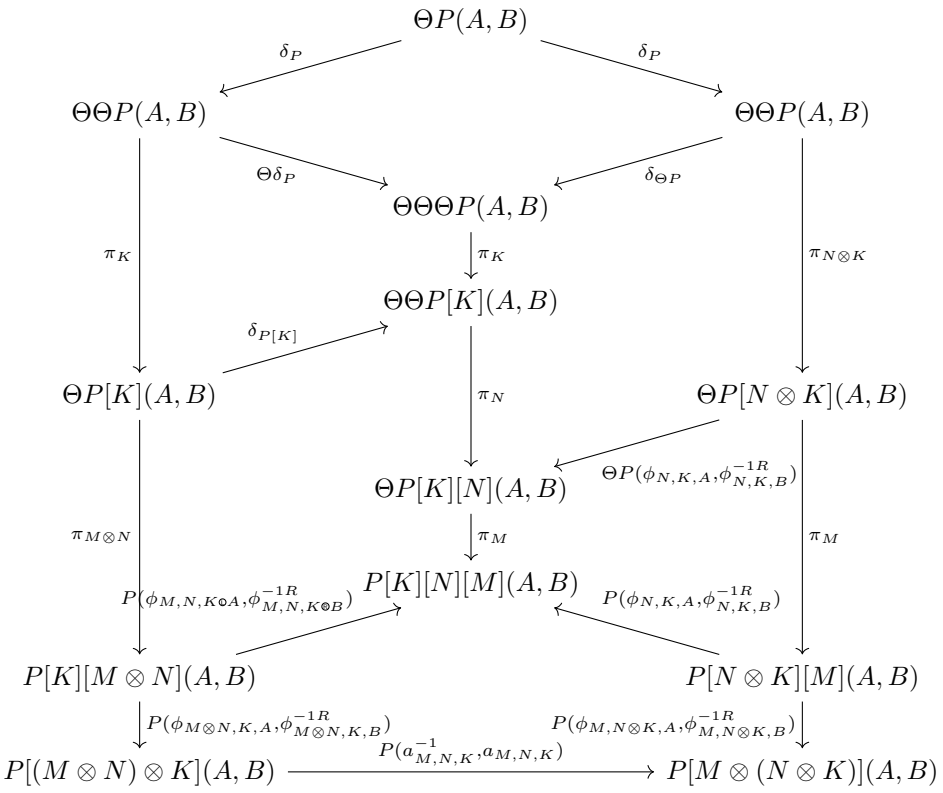


Figure 11: Diagram for the coassociativity axiom.

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.*

$$\Phi Q(X,Y) = \int^{M,U,V} Q(U,V) \otimes \mathbf{C}(X, M \mathbin{\textcircled{L}} U) \otimes \mathbf{D}(M \mathbin{\textcircled{R}} V, Y).$$

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.

$$\begin{aligned}
&\int_{A,B} \mathcal{V}\left(Q(A,B), \int_M P(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B)\right) \\
\cong\ & \{\text{Continuity}\} \\
&\int_{M,A,B} \mathcal{V}(Q(A,B), P(M \mathbin{\textcircled{L}} A, M \mathbin{\textcircled{R}} B)) \\
\cong\ & \{\text{Yoneda}\} \\
&\int_{M,A,B} \mathcal{V}\left(Q(A,B), \int_{X,Y} \mathcal{V}\Big(\mathbf{C}(X, M \mathbin{\textcircled{L}} A) \otimes \mathbf{D}(M \mathbin{\textcircled{R}} B, Y), P(X,Y)\Big)\right) \\
\cong\ & \{\text{Continuity}\} \\
&\int_{M,A,B,X,Y} \mathcal{V}\left(Q(A,B), \mathcal{V}\Big(\mathbf{C}(X, M \mathbin{\textcircled{L}} A) \otimes \mathbf{D}(M \mathbin{\textcircled{R}} B, Y), P(X,Y)\Big)\right) \\
\cong\ & \{\text{Copower}\} \\
&\int_{M,A,B,X,Y} \mathcal{V}(Q(A,B) \otimes \mathbf{C}(X, M \mathbin{\textcircled{L}} A) \otimes \mathbf{D}(M \mathbin{\textcircled{R}} B, Y), P(X,Y)) \\
\cong\ & \{\text{Continuity}\} \\
&\int_{X,Y} \mathcal{V}\left(\int^{M,A,B} Q(A,B) \otimes \mathbf{C}(X, M \mathbin{\textcircled{L}} A) \otimes \mathbf{D}(M \mathbin{\textcircled{R}} B, Y), P(X,Y)\right).
\end{aligned}$$

Alternatively, the adjunction can be deduced from the definition of the comonad Θ and the characterization of global Kan extensions as adjoints to precomposition. □

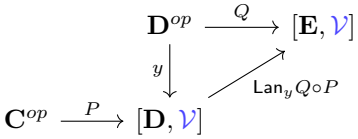
4.3 Pastro-Street's "double" promonad

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]

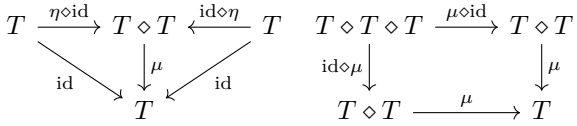
$$(Q \diamond P)(C, E) = \int^{D \in \mathbf{D}} P(C, D) \otimes Q(D, E).$$

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.

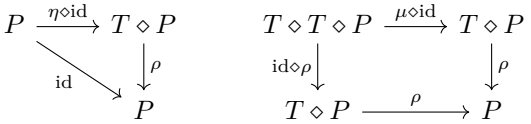


In the same way that we can construct a Kleisli category over a monad, we will perform a Kleisli construction over the monoids of the bicategory **Prof**, which are called **promonads**. Promonads over the base category $\mathcal{V}$ that are also Tambara modules for the product appear frequently in the literature on functional programming languages under the name of *arrows* [14, 15, 35].

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 $\eta_{X,Y}\colon \mathbf{C}(X, Y) \to T(X, Y)$ and $\mu_{X,Y}\colon (T \diamond T)(X, Y) \to T(X, Y)$ satisfying the following unitality and associativity axioms.



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.



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.

[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].

Lemma 4.9. *The bicategory* **Prof** *admits the Kleisli construction [30, §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 $\mathrm{Kl}(T)$. The following isomorphism is a consequence of continuity.

$$\mathcal{V}\left(\int^{Z\in\mathbf{C}} T(X,Z)\otimes T(Z,Y), T(X,Y)\right) \quad\cong\quad \int_{Z\in\mathbf{C}} \mathcal{V}\left(T(X,Z)\otimes T(Z,Y), T(X,Y)\right)$$

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

$$[\mathrm{Kl}(\Phi), \mathcal{V}] \cong [\mathbf{I}^{op}\otimes \mathrm{Kl}(\Phi), \mathcal{V}] \cong \mathbf{Prof}(\mathbf{I}, \mathrm{Kl}(\Phi)) \cong \mathrm{EM}(\Phi).$$ □

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 $\check{T} := T\circ y$ 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. □

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

$$\check{\Phi}(A, B, S, T) = \int^{M} \mathbf{C}(S, M \mathbin{Ⓛ} A) \otimes \mathbf{D}(M \mathbin{Ⓡ} B, T).$$

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 [30] 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 $\mathcal{V} = \mathbf{Set}$ 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 [25].

Lemma 4.12 ("Double Yoneda" from [25])**.** *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` [25]. 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)**.**

$$\int_{P \in \mathbf{Tamb}} \mathcal{V}(P(A, B), P(S, T)) \cong \mathbf{Optic}((A, B), (S, T)).$$

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).

The immediate practical application of this theorem is to justify the following profunctor representation

```
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, which are Tambara modules for 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)
```

A similar encoding works for the rest of the optics.

□

5 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 6.

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.

5.1 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** [41]. It is arguably less flexible than the profunctor representation, being based on representable profunctors, but it is more common in practice. For instance,

Proposition 5.1 (Van Laarhoven-style traversals)**.** *Traversals admit a different encoding in terms of profunctors represented by an applicative functor.*

$$\mathbf{Traversal}((A,B),(S,T)) \cong \int_{F\in\mathbf{App}} \mathcal{V}(A,FB)\otimes\mathcal{V}(S,FT).$$

Proof.

$$\begin{aligned}
&\int_{F\in\mathbf{App}} \mathcal{V}(A,FB)\otimes\mathcal{V}(S,FT)\\
\cong\ &\{\text{Yoneda}\}\\
&\int_{F\in\mathbf{App}} [\mathcal{V},\mathcal{V}](A\otimes[B,-],F)\otimes\mathcal{V}(S,FT)\\
\cong\ &\{\text{Adjunction, free applicatives}\}\\
&\int_{F\in\mathbf{App}} \mathbf{App}\left(\sum_{n\in\mathbb{N}} A^n\otimes[B^n,-],F\right)\otimes\mathcal{V}(S,FT)\\
\cong\ &\{\text{Coyoneda}\}\\
&\mathcal{V}\left(S,\sum_{n\in\mathbb{N}} A^n\otimes[B^n,T]\right).
\end{aligned}$$

□

Exactly the same technique yields lenses and *grates* [26], using arbitrary representable or corepresentable profunctors, respectively.

Proposition 5.2 (Van Laarhoven lenses [41])**.** *Lenses admit an encoding in terms of functors, or representable profunctors.*

$$\mathbf{LinearLens}_{\otimes,\otimes}((A,B),(S,T)) \cong \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(A,FB)\otimes\mathcal{V}(S,FT).$$

Proof. By coend calculus, using the Yoneda lemma.

$$\begin{aligned}
&\int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(A,FB)\otimes\mathcal{V}(S,FT)\\
\cong\ &\{\text{Yoneda}\}\\
&\int_{F\in[\mathcal{V},\mathcal{V}]} [\mathcal{V},\mathcal{V}](A\otimes[B,-],F)\otimes\mathcal{V}(S,FT)\\
\cong\ &\{\text{Coyoneda}\}\\
&\mathcal{V}\left(S, A\otimes[B,T]\right).
\end{aligned}$$

□

Proposition 5.3 (Van Laarhoven-style grates)**.** *Grates admit an encoding in terms of corepresentable functors.*

$$\mathbf{Grate}((A,B),(S,T)) \cong \int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(FA,B)\otimes\mathcal{V}(FS,T).$$

Proof. Again by coend calculus, and using the Yoneda lemma.

$$\begin{aligned}
&\int_{F\in[\mathcal{V},\mathcal{V}]} \mathcal{V}(FA,B)\otimes\mathcal{V}(FS,T)\\
\cong\ &\{\text{Yoneda}\}\\
&\int_{F\in[\mathcal{V},\mathcal{V}]} [\mathcal{V},\mathcal{V}](F,[[\bullet,A],B])\otimes\mathcal{V}(FS,T)\\
\cong\ &\{\text{Coyoneda}\}\\
&\mathcal{V}\left([[S,A],B],T\right).
\end{aligned}$$

□

5.2 Related work

Pastro and Street [30] first described the construction of *doubles* in their study of Tambara theory. Their results can be reused for *optics* thanks to the observations of [25]. 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 [34, §6.1], but his work targeted a more restricted case. Specifically, the definitions of *optic* given in the literature [3, 25, 34] deal only with the particular case in which $\mathcal{V} = \mathbf{Set}$, the categories $\mathbf{C}$ and $\mathbf{D}$ coincide, and the two actions are the same. Riley derives a class of optics and their laws [34, §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 [34, §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* [34, §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 $(\copyright)\colon \mathbf{M}\otimes\mathbf{C}\to\mathbf{C}$ and $(\circledR)\colon \mathbf{N}\otimes\mathbf{D}\to\mathbf{D}$ 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 $\mathbf{E}$.

5.3 Further work

In terms of functional programming, optics of different kinds compose using polymorphic function composition. 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, 31]. Some preliminary results have been discussed by [36], but the proposal to model the lattice is still too *ad-hoc* to be satisfactory. The topic of lawfulness [34, §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 [36, 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 [39]. 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.

6 Appendix: 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

6.1 Concepts of enriched category theory

Definition 6.1 ([18, §1.2], see also [43])**.** 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
```

In Haskell, we define a class of higher-order type constructors that represent a category. 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`. We then ask that they have a identity, `unit`, and a composition, `comp`.

Definition 6.2 ([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
```

Bifunctors and profunctors are both instances of functors. Here, we ask for the maps on morphisms, `bimap` for bifunctors and `dimap` for profunctors.

Definition 6.3 ([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)
```

We define the monoidal category relative to a bifunctor representing the tensor and an object representing the unit. We ask for all the coherence maps, i.e. `alpha` for the associator.

Definition 6.4. A 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))
```

Monoidal actions are bifunctors with extra maps representing the coherence conditions, e.g. `unitor` for the oplaxator.

Definition 6.5. 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
```

6.2 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.

6.3 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 6.6. As an example, for any fixed A and B, consider the profunctor $P_{A,B}(S,T) \coloneqq (S \to A)$. 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
```

6.3.1 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.

6.4 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

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. https://doi.org/10.1007/978-3-319-30936-1_1. 7, 9, 10

[2] Guillaume Boisseau. Understanding Profunctor optics: A Representation Theorem. Master's thesis, University of Oxford, 2017. 7, 9, 13

[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. https://doi.org/10.1145/3236779. 5, 7, 8, 11, 15, 21, 29, 31, 32

[4] Mario Cáccamo and Glynn Winskel. A Higher-Order Calculus for Categories. In Richard J. Boulton and Paul B. Jackson, editors, *Theorem Proving in Higher Order Logics, 14th International Conference, TPHOLs 2001, Edinburgh, Scotland, UK, September 3-6, 2001, Proceedings*, volume 2152 of *Lecture Notes in Computer Science*, pages 136–153. Springer, 2001. https://doi.org/10.1007/3-540-44755-5_11. 6

[5] Geoffrey Cruttwell and Michael Shulman. A unified framework for generalized multicategories. *Theory and Applications of Categories*, 24(21):580–655, 2010. https://doi.org/10.48550/arXiv.0907.2460. 27

[6] Brian Day. On Closed Categories of Functors. In *Reports of the Midwest Category Seminar IV*, pages 1–38, Heidelberg, 1970. Springer. 8, 33

[7] Brian Day and Miguel Laplaza. On Embedding Closed Categories. *Bulletin of the Australian Mathematical Society*, 18(3):357–371, 1978. 18

[8] Ronald Fisher. The Use of Multiple Measurements in Taxonomic Problems. *Annals of Eugenics*, 7(2):179–188, 1936. 12

[9] Brendan Fong and Michael Johnson. Lenses and Learners. In James Cheney and Hsiang-Shang Ko, editors, *Proceedings of the 8th International Workshop on Bidirectional Transformations co-located with the Philadelphia Logic Week, Bx@PLW 2019, Philadelphia, PA, USA, June 4, 2019*, volume 2355 of *CEUR Workshop Proceedings*, pages 16–29. CEUR-WS.org, 2019. https://doi.org/10.48550/arXiv.1903.03671. 32

[10] Nathan Foster, Michael Greenwald, Jonathan Moore, Benjamin 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. https://doi.org/10.1145/1040305.1040325. 3, 4, 9, 13

[11] Thomas Fox. Coalgebras and cartesian categories. *Communications in Algebra*, 4(7):665–667, 1976. 10

[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. https://doi.org/10.1145/3209108.3209165. 32

[14] John Hughes. Generalising Monads to Arrows. *Science of Computer Programming*, 37(1-3): 67–111, 2000. https://doi.org/10.1016/S0167-6423(99)00023-4. 27

[15] Bart Jacobs, Chris Heunen, and Ichiro Hasuo. Categorical Semantics for Arrows. *Journal of Functional Programming*, 19(3-4):403–438, 2009. https://doi.org/10.1017/S0956796809007308. 27

[16] Mauro Jaskelioff and Russell O'Connor. A Representation Theorem for Second-Order Functionals. *Journal of Functional Programming*, 25, 2015. https://doi.org/10.1017/S0956796815000088. 15, 31

[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. https://doi.org/10.4204/EPTCS.76.5. 6, 15, 32

[18] 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. 33

[19] Max Kelly. On the Operads of J. P. May. *Reprints in Theory and Applications of Categories*, 13(1), 2005. 16

[20] Edward Kmett. Lens library, version 4.16. Hackage https://hackage.haskell.org/package/lens-4.16, 2012–2018. 4, 19, 20, 35

[21] Paul Levy. Strong functors on many-sorted sets. *Commentationes Mathematicae Universitatis Carolinae*, 60(4):533–540, 2019. 11, 20

[22] Fosco Loregian. *(Co)end Calculus*. London Mathematical Society Lecture Note Series. Cambridge University Press, 2021. https://doi.org/10.1017/9781108778657. 6, 7

[23] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics. Springer New York, 1978. https://doi.org/10.1007/978-1-4757-4721-8. 17

[24] Conor McBride and Ross Paterson. Applicative programming with effects. *Journal of Functional Programming*, 18(1):1–13, 2008. https://doi.org/10.1017/S0956796807006326. 15

[25] Bartosz Milewski. Profunctor Optics: The Categorical View. https://bartoszmilewski.com/2017/07/07/profunctor-optics-the-categorical-view/, 2017. 5, 6, 8, 9, 15, 21, 29, 31

[26] Russell O'Connor. Grate: a New Kind of Optic. https://r6research.livejournal.com/28050.html, 2015. 18, 31

[27] Russell O'Connor. Mezzolens: Pure Profunctor Functional Lenses. Hackage https://hackage.haskell.org/package/mezzolens, 2015. 4, 14

[28] Frank Joseph Oles. *A Category-Theoretic Approach to the Semantics of Programming Languages*. PhD thesis, Syracuse University, 1982. 9

[29] 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. 9

[30] Craig Pastro and Ross Street. Doubles for Monoidal Categories. *Theory and Applications of Categories*, 21(4):61–75, 2008. 5, 7, 21, 23, 28, 29, 31

[31] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor Optics: Modular Data Accessors. *Programming Journal*, 1(2):7, 2017. https://doi.org/10.22152/programming-journal.org/2017/1/7. 5, 15, 21, 32

[32] 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. 8

[33] Emily Pillmore and Mario Román. Profunctor optics: The categorical view. https://golem.ph.utexas.edu/category/2020/01/profunctor_optics_the_categori.html, 2020. 8

[34] Mitchell Riley. Categories of optics. *arXiv preprint arXiv:1809.00738*, 2018. 5, 7, 8, 9, 11, 13, 15, 20, 21, 31, 32

[35] Exequiel Rivas and Mauro Jaskelioff. Notions of Computation as Monoids. *Journal of Functional Programming*, 27, 2017. https://doi.org/10.1017/S0956796817000132. 27

[36] Mario Román. Profunctor Optics and Traversals. Master's thesis, University of Oxford, 2019. 8, 15, 20, 32

[37] Mario Román. Open Diagrams via Coend Calculus. *Electronic Proceedings in Theoretical Computer Science*, 333, Applied Category Theory 2020:65–78, Feb 2021. ISSN 2075-2180. https://doi.org/10.4204/eptcs.333.5. 8

[38] David Spivak. Generalized lens categories via functors $\mathcal{C}^{\mathrm{op}} \to \mathsf{Cat}$. *arXiv preprint arXiv:1908.02202*, 2019. https://doi.org/10.48550/arXiv.1908.02202. 7, 9, 10

[39] Perdita Stevens. Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. *Software and Systems Modeling*, 9(1):7–20, 2010. https://doi.org/10.1007/s10270-008-0109-9. 32

[40] Daisuke Tambara. Distributors on a tensor category. *Hokkaido mathematical journal*, 35(2):379–425, 2006. 5, 21, 22

[41] Twan van Laarhoven. CPS-based functional references. https://www.twanvl.nl/blog/haskell/cps-functional-references, 2009. 4, 30, 31

[42] Twan van Laarhoven. A non-regular data type challenge. https://www.twanvl.nl/blog/haskell/non-regular1, 2009. 17

[43] Sjoerd Visscher. Data.Category library, version 0.10. Hackage https://hackage.haskell.org/package/data-category, 2010–2020. 33