Lenses and Learners

Brendan Fong
Department of Mathematics
Massachusetts Institute of Technology
bfo@mit.edu

Michael Johnson
Faculty of Science and Engineering
Macquarie University
michael.johnson@mq.edu.au

Abstract

Lenses are a well-established structure for modelling bidirectional transformations, such as the interactions between a database and a view of it. Lenses may be symmetric or asymmetric, and may be composed, forming the morphisms of a monoidal category. More recently, the notion of a learner has been proposed: these provide a compositional way of modelling supervised learning algorithms, and again form the morphisms of a monoidal category. In this paper, we show that the two concepts are tightly linked. We show both that there is a faithful, identity-on-objects symmetric monoidal functor embedding a category of *asymmetric* lenses into the category of learners, and furthermore there is such a functor embedding the category of learners into a category of *symmetric* lenses.

1 Introduction

This paper presents surprising links between two apparently disparate areas: a compositional treatment of supervised learning algorithms, called *learners*, and and a mathematical treatment of bidirectional transformations, called *lenses*. Until this work there had been no known non-trivial relationships between the two areas, and, naively at least, there seemed to be little reason to expect them to be closely related mathematically.

But lenses and learners are indeed mathematically closely related. The main result that we present here is the existence of a faithful, identity-on-objects, symmetric monoidal functor, from the category whose arrows are learners to a category whose arrows are *symmetric lenses*. In addition, the symmetric lenses in the image of that functor have particularly simple structure: as spans of asymmetric lenses, their left legs are the long studied and easily understood lenses known as *constant complement lenses*. Roughly speaking, this means that a supervised learning algorithm may be understood as a special type of symmetric lens.

The right legs of the symmetric lenses in the image of the functor are also simple, but in a different way. The right legs are bare asymmetric lenses — lenses which have, as all lenses do, a Put and Get, but which have no axioms restricting the way the Put and Get interact. Such simple lenses were defined in the very first papers on lenses, and were the ones blessed with the name "lens", but the study of such bare lenses has so far been somewhat neglected by the lens community, and certainly by us, because of a lack of examples or applications requiring them. This paper provides some compelling examples and has led us to study them seriously.

The symmetric lenses that correspond to learners also have links, as will be explained in the discussion section, with recent work on some quite sophisticated notions of lenses called *amendment lenses*, or in a slightly simplified

In: J. Cheney, H-S. Ko (eds.): Proceedings of the Eighth International Workshop on Bidirectional Transformations (Bx 2019), Philadelphia, PA, USA, June 4, 2019, published at http://ceur-ws.org

form, *a-lenses*. These a-lenses do have extra axioms — the ones we have looked at most are called Stable PutGet (or SPG) a-lenses — and it turns out that the extra structure provided by amendment lenses can be added in a canonical way to the lenses that we study here, and in a way that allows then to recover at least the amendment-lens versions of GetPut and PutGet axioms.

Terminology and notation

Before we begin, a few words about notation. In particular, we may need to thank readers for their indulgence in translating notations — since this paper brings together two areas that already have established notation, in order to make the comparisons with those areas easier we have generally retained the appropriate notations for lenses and for learners, but that means that readers sometimes have to carry both notations in their minds and translate between them. We have endeavoured to help.

In a small number of cases where we judged it would do little harm we have taken the opposite approach and made minor changes to established notation. In those cases, if the notation comes from the readers' own area of expertise we have to ask for even more indulgence. We hope we do not engender confusion.

On more mathematical matters, note that we are both category theorists, but that the lenses and learners that we are dealing with here are all set-based, and there is very little explicit use of category theory beyond some pullbacks, the construction of functions from other functions, and the judicious, and mostly unremarked, use of isomorphisms to for example re-order variables inside tuples, and to rebracket tuples of tuples. But sadly that makes for some long strings of variables as parameters for functions.

To manage the visual complexity of some of these complicated function compositions we have also presented them using string diagrams in Set. While we hope that, written alongside the usual elementwise function notation, these will be easy enough to read, the reader unfamiliar with this notation might consult introductory references [Sel11, FS19]. We look forward to a treatment of some of the things presented here as notions internal to a category, and anticipate that they will probably bring with them significant simplifications, especially in notation.

Finally, a remark about how "well-behaved" is a technical term. Many of the lenses we consider here are not well-behaved, but that simply means that they don't satisfy certain conditions, conditions which for the applications considered here would be undesirable. So it's important to remember that being not-well-behaved may be desirable and is certainly not derogatory (contrary to normal English usage).

Outline

The paper is structured as follows.

In the next two sections we introduce lenses, firstly in their asymmetric form (Section 2), and then in their symmetric form (Section 3). Along the way we note how to compose both asymmetric lenses, and symmetric lenses, and we introduce the particularly simple constant complement lenses, and show briefly how constant complement lenses compose to give constant complement lenses. Because of the bare lenses that we are using in this paper, we need to extend somewhat the usual definition of composition of symmetric lenses which is generally defined only when the lenses are well-behaved, or at least each satisfy the PutGet axiom. We show that, with a slightly complicated construction, the usual definition of composition can be extended to symmetric lenses in which one leg satisfies PutGet even if the other leg satisfies no axioms at all.

In Section 4 we introduce learners, their composition and monoidal structure, and how they form a category. As might be expected, to make a category (rather than say a bicategory) we need to take equivalence classes of learners, and the equivalence relation is introduced. It corresponds so closely to the usual equivalence for symmetric lenses presented as spans of asymmetric lenses that we can suppress any detailed treatment of the equivalences in this paper and present the results in terms of representatives of equivalence classes. Some readers at first find the definition of composition for learners a little daunting, so we include string diagrams to illustrate how the composition works.

In Section 5 we present in detail the theorem already alluded to, and illustrate the precise and remarkably parallel relationship between learners and lenses. And then in Section 6 we discuss some of the observations that follow from this work, and conclude with some speculations on possible directions for further studies flowing from these results.

2 Asymmetric Lenses

Asymmetric lenses have been studied in a variety of different categories, and with a range of different forms including d-lenses [DXC11], c-lenses [JR13], and most recently amendment lenses, also known as a-lenses [DKL18]. Furthermore those lenses have been studied as algebraic structures with a number of axioms such as the so-called PutGet law [FG+07], the PutPut law [FG+07, JR12], and many others. In this paper we predominantly restrict our attention to the very simplest cases of set based lenses, as originally presented in [PS03], with no further axioms.

Definition 2.1. *An **asymmetric lens** $(p, g)\colon A \longrightarrow B$ is a pair of functions: a **Put** $p\colon B \times A \longrightarrow A$ and a **Get** $g\colon A \longrightarrow B$.*

These basic asymmetric lenses are the main thing that we need in order to see the relationships with learners, but we will sometimes need to refer to lenses that are "well-behaved", or at least that satisfy one of the following two conditions required for so-called well-behaved lenses.

Definition 2.2. *An asymmetric lens $(p, g)\colon A \longrightarrow B$ is called **well-behaved** if it satisfies the following two conditions*

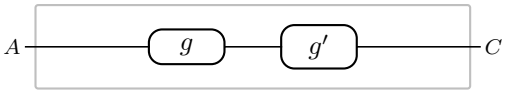
(PutGet) *The Get of a Put is the projection. That is, $g(p(a, b)) = b$, or in string diagrams*

(GetPut) *The Put of an unchanged Get result is unchanged. That is, $p(g(a), a) = a$, or in string diagrams*

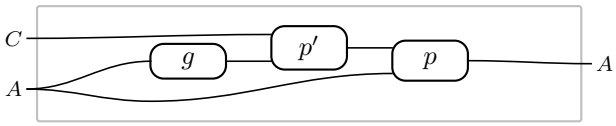
(Here the splitting wire represents the diagonal map $A \longrightarrow A \times A$, i.e. $a \mapsto (a, a)$.)

Even without meeting the definition of well-behaved, lenses compose in a straightforward way, and well-behaved lenses do compose to give well-behaved lenses too. In fact, each of the two well-behaved conditions (PutGet and GetPut) is respected by composition separately, so we can, when we need to, talk about the composition of lenses that satisfy merely PutGet say, and know that the result will also satisfy PutGet.

Definition 2.3. *The **composite lens** $(p, g) \fatsemi (p', g')\colon A \longrightarrow C$ constructed from lenses $(p, g)\colon A \longrightarrow B$ and $(p', g')\colon B \longrightarrow C$ has as Get simply the composite $g'g$ of the Gets*



and as Put, $q\colon C \times A \longrightarrow A$ given by $q(c, a) = p(p'(g(a), c), a)$.



Thus in the notation just introduced, $(p, g) \fatsemi (p', g') = (q, g'g)$.

Definition 2.4. *With the composition just defined, asymmetric lenses are the arrows of a symmetric monoidal category whose objects are sets, and whose monoidal product is given by cartesian product. That category is denoted* Lens*. Furthermore, there is a subcategory of* Lens *called* WBLens *whose arrows are well-behaved asymmetric lenses, as well as subcategories whose arrows satisfy merely PutGet or GetPut.*

Given a cartesian product $B \times A$, we write the projection notation $\pi_2 \colon B \times A \longrightarrow\!\!\!\!\!\rightarrow A$ for the function mapping (b, a) to a; that is, for projection onto the second factor. More generally, we overload the notation π_i using it for any projection onto an ith factor, with it being disambiguated once the domain, expressed as a cartesian product, is known.

The identity lens $(\pi_1, \mathrm{id}_A) \colon A \longrightarrow\!\!\!\!\!\rightarrow A$ on a set A has identity function as Get and projection onto the first factor as Put.

Example 2.5. One of the most basic forms of asymmetric lenses, introduced many years ago in the database community [BS81], is the *constant complement view updating lens*. These are asymmetric lenses of the form $(k, \pi_1) \colon A_1 \times A_2 \longrightarrow\!\!\!\!\!\rightarrow A_1$, where $k(a_1', (a_1, a_2)) = (a_1', a_2)$ — the reader can see the source of the name "constant complement" in the presence of a_2 in both the input and the output of k. It is easy to see that (k, π_1) is well-behaved.

The composite of constant complement lenses is again constant complement. Indeed, suppose further that $A_1 = B_1 \times B_2$, and that $(k', \pi_1) \colon B_1 \times B_2 \longrightarrow\!\!\!\!\!\rightarrow B_1$ is a constant complement lens (so $k'(b_1', (b_1, b_2)) = (b_1', b_2)$). Then the composite lens $(k, \pi_1) \fatsemi (k', \pi_1) \colon B_1 \times B_2 \times A_2 \longrightarrow\!\!\!\!\!\rightarrow B_1$ has Put $k'' \colon B_1 \times (B_1 \times B_2 \times A_2) \longrightarrow\!\!\!\!\!\rightarrow B_1 \times B_2 \times A_2$ given by

$$\begin{aligned} k''\big(b_1', (b_1, b_2, a_2)\big) &= k\big(k'(b_1', (b_1, b_2)), (b_1, b_2, a_2)\big) \\ &= k\big((b_1', b_2), (b_1, b_2, a_2)\big) \\ &= (b_1', b_2, a_2), \end{aligned}$$

and Get given by $\pi_1 \colon B_1 \times (B_2 \times A_2) \longrightarrow\!\!\!\!\!\rightarrow B_1$. This is, up to isomorphism, the constant complement lens $(k'', \pi_1) \colon B_1 \times B_2 \times A_2 \longrightarrow\!\!\!\!\!\rightarrow B_1$.

Such simple composites of constant complement lenses, or indeed of other more complicated lenses, arise very frequently in practice (for example in the database world whenever one deals with views of views). In common with the overloaded notation for projections π_i which are the Gets of constant complement lenses, it is convenient to introduce overloaded notation for the Puts: When there is little risk of confusion we will simply write k for the constant complement Put corresponding to a given Get π_i.

We will return to constant complement lenses when we use them as the "left leg" of certain symmetric lenses to obtain our main result in Section 5.

3 Symmetric Lenses

Asymmetric lenses model well situations where one system, denoted A in the previous section, includes all the relevant information, and another system, B, has information simply derived from A. Of course in practice it's important to deal with the more symmetric situation where two systems share some common structures, but each has information that the other doesn't have. To address this, not long after the seminal work on asymmetric lenses, Hofmann *et al.* developed a symmetrized version of lenses [HPW11] which can be described conveniently as (approximately) spans of asymmetric lenses.

Recall that a **span** in a category is a pair of arrows with common domain, $A_1 \longleftarrow S \longrightarrow A_2$. Despite their evident symmetry, spans are usually considered to be oriented from one of the **feet** to the other, so the span just drawn would be called a span from A_1 to A_2, and the same two arrows also form a span from A_2 to A_1, usually drawn as $A_2 \longleftarrow S \longrightarrow A_1$. The object S is called the **head**, (or sometimes **peak** or **apex**) of the span. When we need to name the arrows, for example if they are asymmetric lenses $(p_1, g_1) \colon S \longrightarrow A_1$ and $(p_2, g_2) \colon S \longrightarrow A_2$, it is convenient to notate the span as $A_1 \xleftarrow{(p_1, g_1)} S \xrightarrow{(p_2, g_2)} A_2$. The lens (p_1, g_1) will be called the **left leg** of the span, and the lens (p_2, g_2) will similarly be called the **right leg** of the span.

The reader may choose to think of a symmetric lens, shortly to be introduced, as a span of asymmetric lenses. But in fact, in common with other descriptions of similar structures, symmetric lenses are really equivalence classes of spans of asymmetric lenses. In this paper it will be convenient to talk about, for example, "the symmetric lens $A_1 \xleftarrow{(p_1, g_1)} S \xrightarrow{(p_2, g_2)} A_2$", when strictly speaking that span is just a representative of an equivalence class of similar spans, and that equivalence class is the symmetric lens.

For completeness we include here the description of the equivalence relation.

Definition 3.1. *Suppose given two spans of asymmetric lenses* $A_1 \xleftarrow{(p_1,g_1)} S \xrightarrow{(p_2,g_2)} A_2$ *and* $A_1 \xleftarrow{(p'_1,g'_1)} S' \xrightarrow{(p'_2,g'_2)} A_2$ *with common feet* A_1 *and* A_2*. A function* $f\colon S \longrightarrow S'$ *is said to satisfy* ***conditions (E)*** *[JR17] when*

(i) f *is surjective.*

(ii) f *preserves Gets:* $g'_1 f = g_1$ *and* $g'_2 f = g_2$*.*

(iii) f *preserves Puts: for all* a_1*,* s *we have* $p'_1(a_1, f(s)) = f(p_1(a_1, s))$ *and* $p'_2(a_2, f(s)) = f(p_2(a_2, s))$*.*

Let $\equiv_{sp}$ be the equivalence relation on spans of asymmetric lenses generated by those functions f between their heads that satisfy conditions (E).

Definition 3.2. *A* ***symmetric lens*** *from* A_1 *to* A_2 *is a* $\equiv_{sp}$*-class of spans of asymmetric lenses from* A_1 *to* A_2*.*

The composition of symmetric lenses is usually defined for (equivalence classes of) spans of well-behaved asymmetric lenses [JR17], but we aim to be more general here. In particular, we will show that the composition rule for well-behaved symmetric lenses generalises to a composition rule for *symmetric lenses with left legs satisfying PutGet*.

Note first that if a symmetric lens has as a representative a span of asymmetric lenses in which one leg satisfies PutGet then all the representatives of that equivalence class have corresponding leg satisfying PutGet.

Definition 3.3. *Suppose that*

$$A_1 \xleftarrow{(q_1,h_1)} S_1 \xrightarrow{(p_2,g_2)} A_2 \qquad \text{and} \qquad A_2 \xleftarrow{(q_2,h_2)} S_2 \xrightarrow{(p_3,g_3)} A_3$$

are spans of asymmetric lenses whose left legs satisfy PutGet. We define their ***composite symmetric lens****, from* A_1 *to* A_3*, as follows.*

Let $S_1 \xleftarrow{\overline{h_2}} T \xrightarrow{\overline{g_2}} S_2$ *be the pullback in* Set *of the cospan* $S_1 \xrightarrow{g_2} A_2 \xleftarrow{h_2} S_2$*. More concretely, without loss of generality, we may suppose that*

$$T = \{(s_1, s_2) \in S_1 \times S_2 \mid g_2(s_1) = h_2(s_2)\},$$

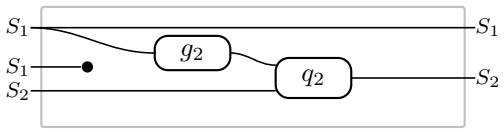
and that $\overline{h_2}$ *and* $\overline{g_2}$ *are the projections onto the first and second components respectively. Next, we equip* $\overline{h_2}$ *with the Put* $\overline{q_2}\colon S_1 \times T \longrightarrow T$ *defined by* $\overline{q_2}\big(s'_1, (s_1, s_2)\big) = \big(s'_1,\, q_2(g_2(s'_1), s_2)\big)$*, and equip* $\overline{g_2}$ *with the Put* $\overline{p_2}\colon S_2 \times T \longrightarrow T$ *defined by*

$$\overline{p_2}\big(s'_2, (s_1, s_2)\big) = \big(p_2(h_2(s'_2), s_1),\, q_2(g_2(p_2(h_2(s'_2), s_1)), s'_2)\big) \in T \subseteq S_1 \times S_2.$$

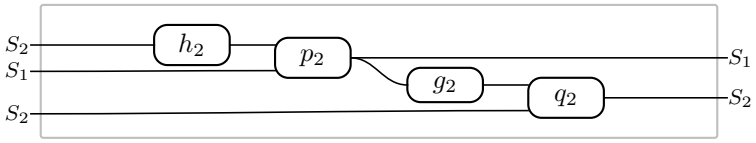
A representative for the composite symmetric lens $A_1 \longrightarrow\!\!\!\!\!\rightarrow A_3$ *is then given by*

$$A_1 \xleftarrow{(\overline{q_2},\overline{h_2})\fatsemi(q_1,h_1)} T \xrightarrow{(\overline{p_2},\overline{g_2})\fatsemi(p_3,g_3)} A_2.$$

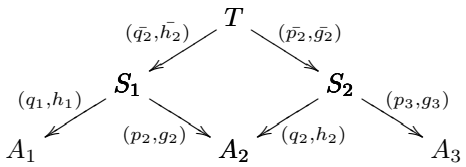
To help parse these expressions, we draw string diagrams for $\overline{q_2}$



and $\overline{p_2}$



It is straightforward to check that the images of the functions $\overline{q_2}$ and $\overline{p_2}$ do indeed lie in their codomain T, and hence that they are well-defined. Note that the construction above gives a diagram of asymmetric lenses



The composite symmetric lens is just given by composing the pairs of arrows on the left and right of the diagram. It is straightforward to verify that $(\overline{q_2}, \overline{h_2})$ satisfies PutGet, and hence that the left leg of the composite also satisfies PutGet.

Remark 3.4. The somewhat complicated second component $q_2(g_2(p_2(h_2(s_2'), s_1), s_2'))$ of $\overline{p_2}$ arises because we do not assume that (p_2, g_2) satisfies PutGet. If we do assume (p_2, g_2) obeys PutGet, then the expression simplifies to the more familiar $q_2(h_2(s_2'), s_2')$. This second component correspondingly also shows that $(\overline{p_2}, \overline{g_2})$ may also fail to satisfy PutGet.

Remark 3.5. It is worth remarking, as noted elsewhere [JR17], that the composition just defined is reminiscent of, but different from, the normal composition of spans in a category with pullbacks. The peak of the span, T, is indeed a pullback, but not in the category Lens. Since the pullback is calculated in Set, the pullback projections are not a priori lenses, but the construction shows how to extend them to be lenses in a canonical way.

The construction presented here is also more general than the usual construction because it has to deal with (right leg) lenses that might not satisfy PutGet. We will comment further on this in the discussion section below.

Using the techniques of [JR17], it can be shown that $\equiv_{sp}$ is a congruence for this more general composition of spans of asymmetric lenses. Thus the composition is well-defined on equivalence classes and provides a symmetric lens composition for those symmetric lenses whose left leg satisfies PutGet. We thus make the following definition.

Definition 3.6. *We define the symmetric monoidal category* SLens *to have sets as objects, symmetric lenses with left leg satisfying PutGet as arrows, and monoidal product given by cartesian product of sets.*

Again, the reader should note that this is slightly more general than other definitions of categories of symmetric lenses, which normally require both legs to satisfy PutGet (and possibly other conditions too). Nonetheless, using the standard techniques it is straightforward to check that this composition rule is associative and unital, and moreover that SLens is a well-defined symmetric monoidal category. Identity symmetric lenses are simply given by the span in which both legs are identity asymmetric lenses.

4 Learners

Learners provide a categorical framework for modelling supervised learning algorithms. They can be seen as parametrised version of asymmetric lenses. In this section we introduce the basic ideas; more detail can be found in [FST19].

Definition 4.1. *A* ***learner*** $(P, I, U, r) \colon A \longrightarrow B$ *is a set* P*, together with three functions: an* ***implementation*** $I \colon P \times A \longrightarrow B$*, an* ***update*** $U \colon B \times P \times A \longrightarrow P$*, and a* ***request*** $r \colon B \times P \times A \longrightarrow A$*.*

The goal of supervised learning is to approximate a function $f \colon A \longrightarrow B$ using pairs $(a, f(a)) \in A \times B$ of sample values, or *training data*. We view P as a set of parameters, and the implementation function as detailing how this set P parametrises functions, seen as hypotheses, $A \longrightarrow B$. Next, given a current hypothesis $p \in P$ and training datum $(a, b) \in A \times B$, the update and request functions describe two ways to react to differences between $I(p, a)$ and b: first by *updating* the hypothesis p to $U(b, p, a)$, and second by *requesting* an alternative input $r(b, p, a)$.

Remark 4.2. While the implementation and update functions are evidently necessary structure for supervised learning, the role of the request function is more subtle. Indeed, the request function only becomes necessary through compositional considerations: it is what permits the construction of new learners by interconnecting given ones. Crucially, it captures the backpropagation part of the widely-used backpropagation algorithm for efficient training neural networks. Further discussion regarding interpretation of the request function can be found in [FST19, Remark II.2].

Example 4.3. Learners are not required to obey any axioms, and so are straightforward to construct. There are, however, learners which have been shown to be more useful than others in practice. One useful way of constructing learners is by using gradient descent on any differentiably parametrised class of functions, such as one defined using a neural net.

Indeed, given a set $\mathbb{R}^k$ and differentiable function $I\colon \mathbb{R}^k \times \mathbb{R}^m \longrightarrow \mathbb{R}^n$, as well as a real number $\epsilon > 0$ that we call a *step size*, we may define a learner $(P, I, U, r)\colon \mathbb{R}^m \longrightarrow \mathbb{R}^n$ by setting

$$U(b, p, a) = p - \epsilon \nabla_p \tfrac{1}{2} \|I(p, a) - b\|^2,$$

$$r(b, p, a) = a - \nabla_a \tfrac{1}{2} \|I(p, a) - b\|^2,$$

where $\|x\|$ is the Euclidean norm on $\mathbb{R}^n$.

A key property of the category of learners is that this interpretation of a differentiable function I is functorial, and indeed this functor captures the structure of the backpropagation algorithm. For more details see [FST19, Theorem III.2].

To state the aforementioned functoriality result, we must first describe what it means to compose learners. As with symmetric lenses, this first relies on stating what it means for two learners to be equivalent.

Definition 4.4. *Given two learners $(P, I, U, r), (P', I', U', r')\colon A \longrightarrow B$ a function $f\colon P \longrightarrow P'$ is said to satisfy **conditions** (E') when*

(i) f is surjective.

(ii) f preserves implementations: $I'(f(p), a) = I(p, a)$.

(iii) f preserves updates: $U'(b, f(p), a) = f(U(b, p, a))$.

(iv) f preserves requests: $r'(b, f(p), a) = r(b, p, a)$.

Just as for symmetric lenses, this generates an equivalence relation $\equiv_l$ on learners. Equivalence classes of learners form the morphisms of a symmetric monoidal category.

Definition 4.5. *The symmetric monoidal category* Learn *has sets as objects and $\equiv_l$-classes of learners as morphisms.*

The composite of learners

$$A \xrightarrow{(P,I,U,r)} B \xrightarrow{(Q,J,V,s)} C.$$

is defined to be $(Q \times P,\ I * J,\ U * V,\ r * s)$, where

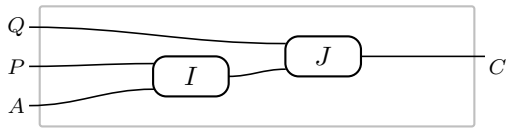
$$\begin{aligned}
(I * J)(q, p, a) &= J(q, I(p, a)), \\
(U * V)(c, q, p, a) &= \Big(U\big(s(c, q, I(p, a)), p, a\big), V\big(c, q, I(p, a)\big)\Big), \\
(r * s)(c, q, p, a) &= r\big(s(c, q, I(p, a))\big).
\end{aligned}$$

The monoidal product of objects A and B is their cartesian product $A \times B$, while the monoidal product of morphisms $(P, I, U, r)\colon A \longrightarrow B$ and $(Q, J, V, s)\colon C \longrightarrow D$ is $(P \times Q,\ I \| J,\ U \| V,\ r \| s)$, where the implementation function is

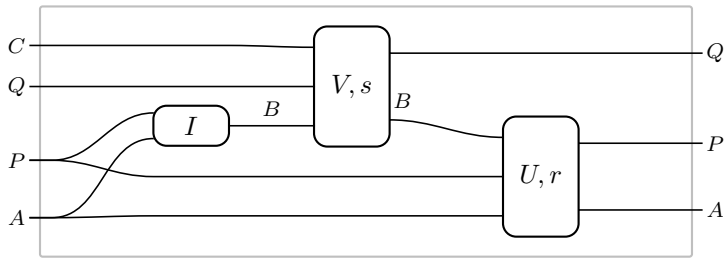
$$\begin{aligned}
(I \| J)(p, q, a, c) &= (I(p, a), J(q, c)), \\
(U \| V)(b, d, p, q, a, c) &= (U(b, p, a), V(d, q, c)), \\
(r \| s)(b, d, p, q, a, c) &= (r(b, p, a), s(d, q, c)).
\end{aligned}$$

Remark 4.6. A proof that this definition indeed specifies a well defined symmetric monoidal category follows from the same arguments as those given in [FST19, Proposition II.4]. Note, however, a key change: in the setting of [FST19], conditions (E′) are strengthened to require f be a bijection. The requirement that f be a bijection was made to avoid a digression about differentiability in [FST19, Definition III.1], and yet still permit a straightforward statement of the main theorem [FST19, Theorem III.2]. Nonetheless, the authors of [FST19] believe conditions (E′) give the more natural notion of equivalence of learner, as it allows identification of parameters that have the same implementation. We believe that the correspondence with conditions (E) from [JR17] via Theorem 5.3 provides further evidence of this claim; indeed, we view this added clarity as a positive outcome of this work and the interaction between our two communities.

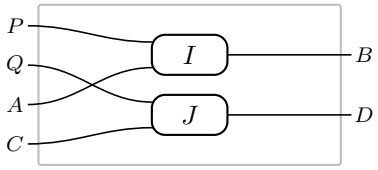
For clarity, let us also present the composition rule using string diagrams in $(\mathsf{Set}, \times)$. Given learners (P, I, U, r) and (Q, J, V, s) as above, the composite implementation function can be written as



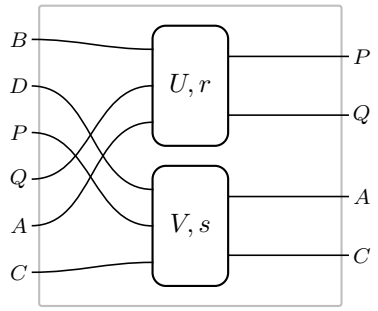
while the composite update–request function $(U * V, r * s)$ can be written as:



The monoidal product of learners is represented in string diagrams as follows. The product implementation function $I \| J$ is



while the composite update and request function $(U \| V, r \| s)$ is



Remark 4.7. Note that lenses are learners with trivial, that is singleton, parameter set (as observed already by Fong et al [FST19]):

Learner $A \longrightarrow B$	**Asymmetric lens** $A \longrightarrow B$
Hypotheses P	1
Implementation $I \colon P \times A \longrightarrow B$	Get $g \colon A \longrightarrow B$
Update $U \colon B \times P \times A \longrightarrow P$	—
Request $r \colon B \times P \times A \longrightarrow A$	Put $p \colon B \times A \longrightarrow A$

This in fact extends to an inclusion of categories.

Proposition 4.8. *There is a faithful, identity-on-objects, symmetric monoidal functor* $\mathsf{Lens} \longrightarrow \mathsf{Learn}$.

Proof. It is straightforward to check that the correspondence laid out above preserves composition and monoidal products. □

5 The Main Result

While it is interesting, it is perhaps not surprising, and may not be especially enlightening, to find that lenses are learners with trivial parameter set (which amounts to barely being a learner at all). There are other ways of seeing relationships between lenses and learners, and in particular of seeing the entire gamut of learners (not just ones with trivial parameters) as lenses.

We first note the following.

Lemma 5.1. *Every learner* $(P, I, U, r)\colon A \longrightarrow B$ *is an asymmetric lens* $(p, g)\colon P \times A \longrightarrow B$.

Proof. Let $g = I\colon P \times A \longrightarrow B$ and let $p = \langle U, r\rangle\colon B \times (P \times A) \longrightarrow P \times A$ be the unique function into the product $P \times A$ determined by U and r. □

The resulting lenses $(\langle U, r\rangle, I)\colon P \times A \longrightarrow B$ will not in general be well-behaved. In particular the training process in supervised learning would not usually be expected to satisfy PutGet. Whether learners, when viewed as in this lemma as lenses should satisfy GetPut is a subject of ongoing research, so for now we make no assumptions. We make a comment again on this in Section 6.

Of course, merely observing that learners are lenses in this way is not especially useful if the composition of learners does not correspond to the composition of lenses. And it cannot. Given two learners $(P, I, U, r)\colon A \longrightarrow B$ and $(P', I', U', r')\colon B \longrightarrow C$ their corresponding lenses under the lemma are not even composable, since they have types $(p, g)\colon P \times A \longrightarrow B$ and $(p', g')\colon P' \times B \longrightarrow C$.

As it happens, however, there is a Kleisli-like composition of these lenses that uses the monoidal product in the category Lens to convert the first of these lenses, by taking its cartesian product with P', to obtain a lens $P' \times P \times A \longrightarrow P' \times B$ which is then composable with the second lens. Remarkably the resulting composition *does* correspond precisely to the composition of the original learners. But all this can be expressed better by relating learners to certain *symmetric* lenses, and we do that now.

Lemma 5.2. *Every learner* $(P, I, U, r)\colon A \longrightarrow B$ *is a symmetric lens*

$$A \xleftarrow{(k,\pi_2)} P \times A \xrightarrow{(\langle U,r\rangle,I)} B$$

with left leg a constant complement (and therefore well-behaved) lens.

Proof. The right leg $(\langle U, r\rangle, I)$ is the asymmetric lens given in Lemma 5.1, while the left leg (k, π_2) is the constant complement (see Example 2.5) lens of the specified type. □

This gives the following correspondence:

Learner $A \longrightarrow B$	**Symmetric lens** $A \longrightarrow B$
Hypotheses P	Left leg complement P
Implementation $I\colon P \times A \longrightarrow B$	Right leg Get $g\colon P \times A \longrightarrow B$
Update $U\colon B \times P \times A \longrightarrow P$	Right leg Put $p\colon B \times P \times A \longrightarrow P \times A$ (1st component)
Request $r\colon B \times P \times A \longrightarrow A$	Right leg Put $p\colon B \times P \times A \longrightarrow P \times A$ (2nd component)

It should be remarked that in both the preceding lemmas we would normally use the word "yields" rather than "is", and we would be explicit about the process that converts a learner into a lens. However, here we have used "is" to emphasise that what we are describing is nothing more than minor repackaging of the data. Furthermore, as the following theorem shows, the important interactions among the data (composition and monoidal product) are exactly the same, whether one treats the data as learners or lenses.

We are now in a position to state the main result.

Theorem 5.3. *There is a faithful, identity-on-objects, symmetric monoidal functor* Learn $\longrightarrow$ SLens *mapping*

$$(P, I, U, r)\colon A \longrightarrow B$$

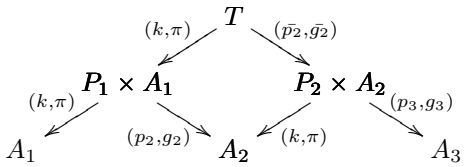
to

$$A \xleftarrow{(k,\pi_2)} P \times A \xrightarrow{(\langle U,r\rangle,I)} B.$$

With the correspondences established, the proof is largely a routine verification. Since Learn and SLens both have sets as objects, we may define the functor to act as the identity on objects. On arrows, the functor acts as in correspondence presented in Lemma 5.2 and outlined in the table above; the fact that this operation is independent of representive chosen follows immediately from the similarity between equivalence relation conditions (E) and (E′) (see Definitions 3.1 and 4.4). It is easy to see that this proposed functor preserves identities and the monoidal product.

The main difficulty is proving that the proposed functor preserves composition. Before walking through this in detail on the next page, we first present a useful lemma regarding composition of symmetric lenses.

Note that in Section 3 the composition of symmetric lenses with left legs satisfying PutGet was presented in its maximum generality. The motivation for that will be discussed in Section 6. To prove the above theorem, it will be helpful to understand the simpler case where the left legs are known to be constant complement. We summarise our notation for the composition in the following diagram (in which notation is abused in the usual way for constant complement lenses):



The following lemma gives an explicit formula for the composite of two symmetric lenses whose left legs are constant complement.

Lemma 5.4. *Suppose that* $A_1 \xleftarrow{(k,\pi_2)} P_1 \times A_1 \xrightarrow{(p_2,g_2)} A_2$ *and* $A_2 \xleftarrow{(k,\pi_2)} P_2 \times A_2 \xrightarrow{(p_3,g_3)} A_3$ *are spans of asymmetric lenses whose left legs, both denoted* (k, π_2)*, are constant complement lenses. Note that constant complement lenses satisfy PutGet and hence we may compose them using Definition 3.3.*

Their composite symmetric lens from A_1 *to* A_3 *is represented by*

$$A_1 \xleftarrow{(k,\pi_3)} P_2 \times P_1 \times A_1 \xrightarrow{(p,g)} A_3,$$

where p *and* g *are given by*

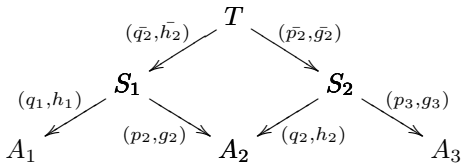
$$p(a_3, (m_1, m_2, a_1)) = \big(m_2', p_2(a_2', (m_1, a_1))\big),$$

in which we let $(m_2', a_2') = p_3(a_3, (m_2, g_2(m_1, a_1))) \in P_2 \times A_2$*, and*

$$g(m_2, m_1, a_1) = g_3(m_2, g_2(m_1, a_1)).$$

(Note that to avoid overloading the notation p_i*, we have written* m_i *for elements of* P_i*.)*

Proof. Recall that in Definition 3.3 we use notation as in the following diagram.



Consider then, as in Definition 3.3, the pullback T in Set of the cospan $P_1 \times A_1 \xrightarrow{g_2} A_2 \xleftarrow{\pi_2} P_2 \times A_2$. Knowing how to calculate pullbacks in Set, we may suppose without loss of generality that the elements of T are tuples (m_2, m_1, a_1), in which there is no a_2 explicitly mentioned since it must be equal to $g_2(m_1, a_1)$, and that $\overline{h_2}$ and $\overline{g_2}$ are π_{23} and $P_2 \times g_2$ respectively. More explicitly still, T is just the product $P_2 \times P_1 \times A_1$, $\overline{h_2}$ is the projection onto $P_1 \times A_1$, and $\overline{g_2}$ is the arrow $P_2 \times P_1 \times A_1 \longrightarrow P_2 \times A_2$ which preserves the P_2 value and uses g_2 to convert the other two values into an A_2 value.

According to Definition 3.3, the left leg of the composite is the composition of two asymmetric lenses denoted there as $(\overline{q_2}, \overline{h_2}) \fatsemi (q_1, h_1)$. In the current context, (q_1, h_1) is the constant complement lens $P_1 \times A_1 \longrightarrow A_1$, and we have seen the $\overline{q_2}$ may be taken to be the projection $P_2 \times P_1 \times A_1 \longrightarrow P_1 \times A_1$. Furthermore the definition of

$\overline{h_2}$ in Definition 3.3 is easily seen, up to reordering of variables, to be in this context the constant complement Put. Finally, we have already seen in Example 2.5 how the composition of two constant complement lenses is a constant complement lens, so the left leg of the composition here is simply the constant complement lens $(k, \pi_3) : P_2 \times P_1 \times A_1 \longrightarrow\!\!\!\!\!\rightarrow A_1$.

We turn now to the right hand leg (p, g). Again Definition 3.3 tells us that it is given by the asymmetric lens composition denoted there as $(\overline{p_2}, \overline{g_2}) \fatsemi (p_3, g_3)$, in which $\overline{p_2}$ was defined by what we referred to there as the "somewhat complicated expression". Referring to Definition 2.3 for how to compose asymmetric lenses, and using the notation of the statement of the lemma, we see that

$$p(a_3, (m_1, m_2, a_1)) = \overline{p_2}((m_2', a_2'), (m_2, m_1, a_1)).$$

We now show that this becomes, simply by substituting and simplifying, $\big(m_2', p_2(a_2', (m_1, a_1))\big)$.

In detail

$$\begin{aligned}
\overline{p_2}((m_2', a_2'), (m_2, m_1, a_1)) &= \overline{p_2}(s_2', (s_1, s_2)) && \text{(in the notation of Definition 3.3)}\\
&= \big(p_2(h_2(s_2'), s_1),\ q_2(g_2(p_2(h_2(s_2'), s_1)), s_2')\big) && \text{(by Definition 3.3)}\\
&= \big(p_2(a_2', s_1), q_2(g_2(p_2(a_2', s_1)), s_2')\big) && \text{(since } h_2(s_2') = \pi_2(m_2', a_2') = a_2')\\
&= \big(p_2(a_2', s_1), q_2(g_2(p_2(a_2', s_1)), (m_2', a_2'))\big) && \text{(since } s_2' = (m_2', a_2'))\\
&= \big(p_2(a_2', s_1), (m_2', g_2(p_2(a_2', s_1)))\big) && \text{(since } q_2 \text{ is constant complement)}\\
&= \big(p_2(a_2', s_1), m_2'\big) && \text{(by comment below)}\\
&= \big(p_2(a_2', (m_1, a_1)), m_2'\big) && \text{(since } s_1 = (m_1, a_1))
\end{aligned}$$

The second to last line holds since, as noted above, a fourth component in T is superfluous since it has to be (and indeed is) g_2 applied to the first component. Reordering the variables in that last line, because we have chosen to keep the A_i in the last position, completes the proof. □

We now return to the proof of Theorem 5.3.

Proof of Theorem 5.3. We will check that the proposed functor preserves composition. This is simply a matter of comparing the Get and Put of the right leg of composite symmetric learners, as detailed in Lemma 5.4, with the formulas for composition of learners as detailed in Definition 4.5.

First compare the definition of $I * J$ in Definition 4.5 with the description of the Get of the composite right legs, $g(m_2, m_1, a)$ of Lemma 5.4, recalling the correspondence between the implementation operations I and J and the right leg Gets g_2 and g_3. In other words, compare

$$g(m_2, m_1, a) = g_3(m_2, g_2(m_1, a_1))$$

with

$$(I * J)(p, q, a) = J(q, I(p, a))$$

noting the naming of variables means p and q correspond respectively to m_1 and m_2 (and that while the order of parameters for g is not important, the choice made for symmetric lens composition was to add new parameters on the left, corresponding to the choice used in function composition).

Next, we compare the right leg Put p of Lemma 5.4 with the composite update–request function $\langle U * V, r * s\rangle$. We do this by considering each of the two components separately.

The A_1 component is $\pi_2(p_2(a_2', (m_1, a_1)))$, which since $a_2' = \pi_2(p_3(a_3, (m_2, g_2(m_1, a_1))))$ is

$$\pi_2 p_2\big(m_1, a_1, \pi_2 p_3(a_3, m_2, g_2(m_1, a_1))\big)$$

which should be compared with

$$(r * s)(c, p, q, a) = r\big(p, a, s(c, q, I(p, a))\big)$$

recalling again that I corresponds to g_2, that p and q correspond to m_1 and m_2, that a and c correspond to a_1 and a_3, and that r and s correspond to $\pi_2 p_2$ and $\pi_2 p_3$ respectively.

Finally the $P_1 \times P_2$ component has, as its two coordinates,

$$\pi_1 p_2\big(\pi_2 p_3(a_3, m_2, g_2(m_1, a_1)), m_1, a_1\big) \qquad \text{and} \qquad \pi_1 p_3\big(a_3, m_2, g_2(m_1, a_1)\big)$$

which should be compared respectively with

$$U\big(s(c, q, I(p, a)), p, a\big) \qquad \text{and} \qquad V\big(c, q, I(p, a)\big),$$

recalling all the correspondences we've already pointed out, along with the correspondences between U and V and $\pi_1 p_2$ and $\pi_1 p_3$ respectively.

These functions are all the same up to the specified renaming correspondences, and hence our functor preserves composition. □

6 Discussion

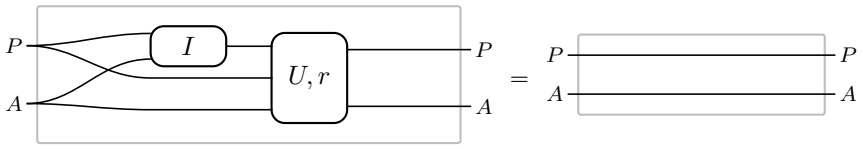
In this section we discuss two directions of research suggested by the main theorem: laws for well-behaved learners, and links between learners and multiary lenses.

6.1 Learner laws

We have shown that the usual notion of composition of symmetric lenses admits a generalization that receives a functor from Learn. Instead, one might consider adding conditions to the notion of learner that permit learners to embed into a more familiar notion of symmetric lens. Put another way: the lens laws suggest analogues for learners.

For example, the GetPut law generalises as follows.

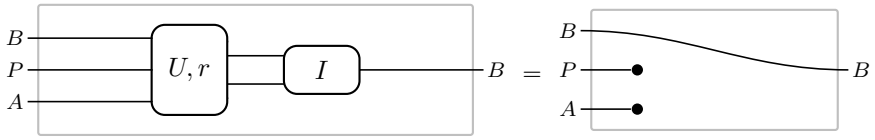
Definition 6.1. *We say that a learner (P, I, U, r) obeys the **I-UR law** if for every parameter $p \in P$ we have both $r(I(p, a), p, a) = a$ and $U(I(p, a), p, a) = p$, or in string diagrams*



This law asks that the lens $(\langle U, r\rangle, I)$ obey the GetPut law. Intuitively, it states that if, at a given parameter p, the training pair (a, b) provided is already classified correctly by the learner—that is, if the training pair is of the form $(a, I(p, a))$—then the update function does not change the parameter and the request function does not request any change to the input. This sort of property can be a desirable property of learning algorithm, and a number of simple, important examples of learners, including those of Example 4.3, satisfy the I-UR law.

We have focussed more on the PutGet law in this paper. This may be generalised as follows.

Definition 6.2. *We say that a learner (P, I, U, r) obeys the **UR-I law** if for every parameter $p \in P$ and input $a \in A$ we have $I(U(b, p, a), r(b, p, a)) = b$, or in string diagrams*



This law asks that the lens $(\langle U, r\rangle, I)$ obey the PutGet law. The intuition here is that when given a training pair (a, b), if the requested input $r(b, p, a)$ is given to the implementation function at the new parameter $U(b, p, a)$, then the training pair $(r(b, p, a), b)$ will be correctly classified by the learner. This is too strong for the incremental learning witnessed in practical supervised learning algorithms such as neural networks. Nonetheless, it is clear that a learner with this property would be in some sense desirable, or well-behaved.

Indeed, learning algorithms in practice must take into account practical considerations such as learning speed and convergence, and the prioritisation of these considerations leads to methods that violate abstract properties such as the I-UR and UR-I laws that might characterise what it means to learn effectively. Nonetheless, we believe

the formulation of these properties, from well-motivated considerations such as our main theorem, suggest ideas that could help frame and guide development of learning algorithms, especially should the intent be to construct algorithms which can be reasoned about to some extent.

This view of learners obeying generalised lens laws suggests a view of a learner as just a parametrised family of lenses, together with a rule for choosing which lens in this family to use given some examples of what you want the lens to do.

6.2 Links with multiary lenses

During the course of preparing this work for this workshop an interesting similarity has come to light. In this paper the main tool we are using is symmetric lenses with left leg constant complement, and right leg bare lenses. In another paper presented at this workshop [JR19] that studies an entirely different area, the main tool the authors use is symmetric lenses (in fact, wide spans of lenses, so they may have in general more than two legs, but they do have at least two legs) with left leg what is known as a closed spg-lens, and right leg(s) arbitrary spg-lenses.

The similarity is more than just the linguistic parallel just described. We'll not define spg-lenses or closed spg-lenses here, but we remark that constant complement lenses are indeed closed spg-lenses. In both cases composition along those left legs is important, and the fact that they are, in both cases, closed and satisfy PutGet is what is important for the composition to work. The nature of the left legs is critical for the main idea in both papers.

What of the right legs? At first they seem very different. In this paper the right legs are bare lenses — they have a Put p and Get g and nothing else, neither more structure, nor axioms. In apparent contrast, the right legs in [JR19] seem to have a substantial amount of structure. They do have a p and a g, but they also have something called an *amendment* and several axioms. The basic idea is that an update, expressed there as an arrow v in a category (here we only have the codomain of such an arrow when we are doing an update because these are set-based lenses), might result in not only a modification of the other component (or in the case of the multiary lenses of [JR19], the other components), but also an *amendment*. This amendment a can be composed with v so that while the Get of the Put might not be v, it will be av. In other words the amendment *repairs* PutGet.

Now are the right hand lenses really that different? There is a standard way of seeing set based lenses as so called delta lenses (lenses that take arrows, not just codomain objects, as the input for Put). It appears as part of a unified treatment of many different kinds of lenses [JR16] and involves co-discrete categories. In a codiscrete category there is a unique way of extending each of the view updates from the bare lenses of this paper to make them line up with their own Put, in other words a unique way of extending bare lenses that satisfy no axioms to spg-lenses that satisfy PutGet. So, the two "main tools" are actually remarkably similar.

They still differ in one respect (only): spg-lenses are required to satisfy an axiom that corresponds to GetPut here. But we have just discussed how GetPut is in fact a desirable property that might be asked of Learners. If it were, then the two very different projects are in fact using exactly alignable, novel, tools: Closed amendment lenses (of the constant complement variety here) as left leg and spg amendment lenses as right leg(s).

The similarities and what they might mean (if anything) will be considered further in future work.

7 Conclusion

To summarise, in this paper we have described a faithful, identity-on-objects, symmetric monoidal functor from a category which captures the notion of composable supervised learning algorithms to a suitable category of lenses. To do this, we presented a slight generalisation of the usual notion of symmetric lens, in which we require a very weak form of well behavedness: a span of asymmetric lenses in which the left leg satisfies the PutGet law. Despite the general definition, these symmetric lenses still compose, and indeed equivalence classes of them form the morphisms of a symmetric monoidal category SLens. Our main theorem describes the aforementioned close, functorial relationship between the category of learners, as defined in [FST19], with this category of lenses. In this theorem, we witness a surprising yet highly robust link between two previously unrelated fields. We believe, as hinted by our brief discussion in Section 6, this to be a rich connection deserving of further exploration.

Acknowledgements

This work has been supported by the Australian Research Council and USA AFOSR grants FA9550-14-1-0031, FA9550-17-1-0058. BF thanks Jules Hedges for first bringing the contents of Remark 4.7 to his attention.

References

[BS81] Bancilhon, F. and Spyratos, N. (1981) Update semantics of relational views. *ACM Trans. Database Syst.* **6**, 557–575.

[DXC11] Diskin, Z., Xiong, Y. and Czarnecki, K. (2011) From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology* **10**, 1–25. doi:10.5381/jot.2011.10.1.a6

[DKL18] Diskin, Z., König, H. and Lawford, M. (2018) Multiple model synchronization with multiary delta lenses. *Lecture Notes in Computer Science* **10802**, 21–37.

[FST19] Fong, B., Spivak, D. I. and Tuyéras, R. (2019) Backprop as functor: a compositional perspective on supervised learning. To appear in *Proceedings of the 34th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2019*. Preprint available as arXiv:1711.10455.

[FS19] Fong, B., and Spivak, D. I. (2019) *An Invitation to Applied Category Theory: Seven Sketches in Compositionality*, Cambridge University Press.

[FG+07] Foster, J., Greenwald, M., Moore, J., Pierce, B. and Schmitt, A. (2007) Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems* **29**.

[HPW11] Hofmann, M., Pierce, B., and Wagner, D. (2011) Symmetric Lenses. *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), ACM SIGPLAN Notices* **46**, 371–384. doi:10.1145/1925844.1926428

[JR12] Johnson M. and Rosebrugh, R. (2012) Lens put-put laws: monotonic and mixed. Proceedings of the 1st International Workshop on Bidirectional Transformations, Tallin *Electronic Communications of the EASST*, **49**, 13pp.

[JR13] Johnson, M. and Rosebrugh, R. (2013) Delta lenses and fibrations. Proceedings of the 2nd International Workshop on Bidirectional Transformations, Rome *Electronic Communications of the EASST* **57**, 18pp.

[JR16] Johnson, M. and Rosebrugh, R. (2016) Unifying set-based, delta-based and edit-based lenses. Proceedings of the 5th International Workshop on Bidirectional Transformations, Eindhoven *CEUR Proceedings* **1571**, 1–13.

[JR17] Johnson, M. and Rosebrugh, R. (2017) Symmetric delta lenses and spans of asymmetric delta lenses. *Journal of Object Technology*, **16**, 2:1–32.

[JR19] Johnson, M. and Rosebrugh, R. (2019) Multicategories of Multiary Lenses. To appear in *Proceedings of the Eighth International Workshop on Bidirectional Transformations (Bx2019).*

[PS03] Pierce, B. and Schmitt, A. (2003) Lenses and view update translation. Preprint.

[Sel11] Selinger, P. (2011) A survey of graphical languages for monoidal categories. In Bob Coecke, editor, *New Structures for Physics*, Lecture Notes in Physics 813:289–355, Springer.