Limits of bimorphic lenses

Jules Hedges

Bimorphic lenses are a simplification of polymorphic lenses that (like polymorphic lenses) have a type defined by 4 parameters, but which are defined in a monomorphic type system (i.e. an ordinary category with finite products). We show that the category of bimorphic lenses is complete when the base category is complete, cocomplete and cartesian closed, and so symmetric bimorphic lenses can be defined as spans of ordinary bimorphic lenses. This is in contrast to monomorphic lenses, which do not have pullbacks, and for which the category of spans can be defined in an ad-hoc way only when the lenses satisfy a certain axiom (the put-get law). This is a step towards a theory of symmetric polymorphic lenses. Bimorphic lenses additionally play an essential role in compositional game theory, and spans of bimorphic lenses are a step towards a compact closed category of open games.

1 Introduction

There are many variants of lenses, including asymmetric vs. symmetric lenses. One distinction that has received little attention is that between monomorphic and polymorphic lenses. A monomorphic lens is one in which the updated view must have the same type as the original, whereas a polymorphic lens allows it to have a different type. The lenses originally introduced in [FGM+07] were monomorphic, and the lenses studied by the bx community are mainly monomorphic. On the other hand, the lenses that have seen widespread use in the Haskell and Purescript programming languages are polymorphic. In this paper we suggest a way to combine symmetric lenses [HPW11] with polymorphic lenses.

A monomorphic lens $\lambda : X \rightarrow Y$ consists of a view function $v_\lambda : X \rightarrow Y$ and an update function $u_\lambda : X \times Y \rightarrow X$. We think of $X$ as being a datatype (for example a database) and $Y$ a ‘zoomed-in’ piece of that datatype (for example, the results of a particular query). $Y$ is called a ‘view’ of $X$, and the view function returns the state of the view given the state of the whole. The update function takes an initial state of the whole and an updated state of the view, and propagates the update to give the new state for the whole. For example there is a lens $\lambda : X \times Y \rightarrow X$ that focusses on the first component of a pair, with view function $v_\lambda(x, y) = x$ and update function $u_\lambda((x, y), x') = (x', y)$. 

1
In [JR14], the direct definition of symmetric lenses in [HPWT11] was shown to be equivalent in a certain sense to spans of asymmetric lenses. However the usual construction of categories of spans requires the underlying category to be complete, and categories of asymmetric lenses are not complete, so Johnson and Rosebrugh define the category of spans in an ad-hoc way. This construction works only for lenses that satisfy the put-get law, namely \( v_\lambda \circ u_\lambda = \pi_2 \).

A polymorphic lens goes between pairs of types: a polymorphic lens \( \lambda : \left( \frac{S}{T} \right) \to \left( \frac{A}{B} \right) \) consists of a view function \( v_\lambda : S \to A \) and an update function \( u_\lambda : S \times B \to T \). Crucially, \( S, T, A \) and \( B \) are types in polymorphic type systems and will generally have type variables in common. Indeed, when formulating the lens laws for polymorphic lenses, a well-behaved lens necessarily has type variables shared between \( S \) and \( T \), and between \( A \) and \( B \). This fact appears in [Kme12], but the author is not aware of any academic work formalising well-behaved polymorphic lenses in a model of polymorphic type theory, either using parametricity or fibred categories. There has however been work on formalising van Laarhoven and profunctor lenses [PGW17, BG18], equivalent formulations of the definition given above that are used respectively in Haskell and Purescript.

Bimorphic lenses, introduced by the author in [Hed17], are an intermediate notion which are ‘4-legged’ but do not use a polymorphic type system. Thus a bimorphic lens has type \( \lambda : \left( \frac{S}{T} \right) \to \left( \frac{A}{B} \right) \) where \( S, T, A, B \) are ordinary sets. This can be seen as a lens in which the view (of type \( A \)) can be updated to have a possibly different but fixed type \( B \), resulting in type of the whole changing from \( S \) to \( T \). It is not possible to formulate the lens laws for a bimorphic lens, and for this reason they are unlikely to be of direct interest to bx theorists.

Bimorphic lenses are studied by the author because of the crucial role they play in compositional game theory [Hed18], and the ultimate intention is to use spans of bimorphic lenses to formulate a compact closed category of open games. It should also be possible to formalise polymorphic lenses by combining bimorphic lenses with a polymorphic type theory, and for this reason this paper can still be seen as a major step towards symmetric polymorphic lenses.

2 The category of bimorphic lenses

Categories of lenses generally have (formalisations of) datatypes as objects and lenses as morphisms. In this section we define the category \( \text{Bilens} \) from [Hed17], whose objects are pairs of sets and whose morphisms are bimorphic lenses.

Throughout this section, \( \mathcal{C} \) refers to a category with finite products.

**Definition 1.** Let \( S, T, A, B \) be objects of \( \mathcal{C} \). A bimorphic lens \( \lambda : \left( \frac{S}{T} \right) \to \left( \frac{A}{B} \right) \) over \( \mathcal{C} \) consists of a view morphism \( v_\lambda : S \to A \) and an update morphism \( u_\lambda : S \times B \to T \) in \( \mathcal{C} \).

Throughout the remainder of this paper we refer to bimorphic lenses simply as lenses.

**Definition 2.** Let \( S, T \) be objects of \( \mathcal{C} \). The identity lens \( \text{id}(\frac{S}{T}) : \left( \frac{S}{T} \right) \to \left( \frac{S}{T} \right) \) is given by \( v_{\text{id}}(\frac{S}{T}) = \text{id}_S \) and \( u_{\text{id}}(\frac{S}{T}) = \pi_2 : S \times T \to T \).
Definition 3. Let $\lambda : (S_T) \to (A_B)$ and $\mu : (A_B) \to (P_Q)$ be lenses over $\mathcal{C}$. The composition $\mu \circ \lambda : (S_T) \to (P_Q)$ is given by $v_{\mu \circ \lambda} = v_\mu \circ v_\lambda$ and

$$u_{\mu \circ \lambda} : S \times Q \xrightarrow{\Delta_S \times Q} S \times S \times Q \xrightarrow{S \times v_\lambda \times Q} S \times A \times Q \xrightarrow{S \times u_\mu} S \times B \xrightarrow{u_\mu} T$$

Proposition 1. With this structure, there is a category $\text{Bilens}(\mathcal{C})$ whose objects are pairs of sets and morphisms are lenses.

Proposition 2. There is an identity-on-objects functor $(-) : \mathcal{C} \times \mathcal{C}^{\text{op}} \to \text{Bilens}(\mathcal{C})$, defined as follows. If $f : S \to A$ and $g : B \to T$ are morphisms of $\mathcal{C}$, then $(f) : (S_T) \to (A_B)$ is the lens with $v_{(f)} = f$ and $u_{(f)} = g \circ \pi_2$.

$\mathcal{C} \times \mathcal{C}^{\text{op}}$ plays the role of the category of isos. However, in the absence of lens laws it is no longer guaranteed that $f$ and $g$ form an isomorphism, so we instead follow [PGW17] and refer to morphisms of $\mathcal{C} \times \mathcal{C}^{\text{op}}$ as adaptors. We write an object of $\mathcal{C} \times \mathcal{C}^{\text{op}}$ as $(S,T)$, in order to notationally distinguish it from the object $(S_T)$ of $\text{Bilens}(\mathcal{C})$.

3 Products of lenses

It is easy to prove that $\text{Bilens}$ has products, given by $(S_T) \times (S_T) = (S_1 \times S_2, T_1 + T_2)$. It is a special case of the fact that intuitionistic dialectica categories have products [dP91, proposition 5]. The fact that $\text{Bilens}$ is a degenerate dialectica category was pointed out to the author by Dusko Pavlovic (private communication).

We give a new proof, using the fact that the embedding of adaptors into lenses has a left adjoint and hence preserves limits, and limits of adaptors are easy to compute.

Proposition 3. Let $\mathcal{C}$ be a category with finite products. There is a functor $V : \text{Bilens}(\mathcal{C}) \to \mathcal{C}$ given by $V(S_T) = S$ and $V(\lambda) = v_\lambda$.

As pointed out in [Hed17], $V$ is a fibration, and is in fact the fibrewise opposite of Jacobs’ simple fibration $s(\mathcal{C}) \to \mathcal{C}$ [Jac99, section 1.3]. $V$ is called the view fibration. For the case $\mathcal{C} = \text{Set}$ it was further pointed out in [Hed18] that $V \cong \text{hom}_{\text{Bilens}(\mathcal{C})}(S_T, -)$, and this also holds for any cartesian closed $\mathcal{C}$ since we can enrich $\text{Bilens}(\mathcal{C})$ in $\mathcal{C}$ by

$$\text{hom}_{\text{Bilens}(\mathcal{C})} \left( \begin{array}{c} S_T \\ \downarrow \lambda \\ \hline A \\ \downarrow B \\ \hline T \end{array} \right) = (S \to A) \times (S \times B \to T)$$

Proposition 4. Let $\mathcal{C}$ be a cartesian closed category. There is a functor $K : \text{Bilens}(\mathcal{C})^{\text{op}} \to \mathcal{C}$ given on objects by $K(S_T) = S \to T$, and on lenses $\lambda : (S_T) \to (A_B)$ by $K(\lambda) : (A \to B) \to (S \to T)$ by the currying of

$$S \times (A \to B) \xrightarrow{\Delta_S \times (A \to B)} S \times S \times (A \to B) \xrightarrow{S \times v_\lambda \times (A \to B)} S \times A \times (A \to B) \xrightarrow{S \times v_\mu \times A_B} S \times B \xrightarrow{u_\mu} T$$

3
$K$ is called the *continuation functor*. In the case $\mathcal{C} = \text{Set}$ it can be written more plainly as $K(\lambda)(k)(s) = u_\lambda(s, k(v_\lambda(s)))$. Again viewing $\text{Bilens}(\mathcal{C})$ as enriched in $\mathcal{C}$, we have $K \cong \text{hom}_{\text{Bilens}(\mathcal{C})}(-, (1))$. The dual functors $V$ and $K$ play a central role in compositional game theory, where they describe the contexts in which an open game can be played.

**Proposition 5.** Let $\mathcal{C}$ be a cartesian closed category with finite (all) products and coproducts. Then $\text{Bilens}(\mathcal{C})$ has finite (all) products, given by

$$\prod_{i: I} (S_i, T_i) = \left( \prod_{i: I} S_i \right) \sqcup \left( \prod_{i: I} T_i \right)$$

**Proof.** We show that $(-, -) : \mathcal{C} \times \mathcal{C}^{\text{op}} \to \text{Bilens}(\mathcal{C})$ is a right adjoint, and hence preserves limits. The result follows since products in $\mathcal{C} \times \mathcal{C}^{\text{op}}$ are given by $\prod_{i: I} (S_i, T_i) = \left( \prod_{i: I} S_i \right) \sqcup \left( \prod_{i: I} T_i \right)$.

Let $\langle V, K \rangle : \text{Bilens}(\mathcal{C}) \to \mathcal{C} \times \mathcal{C}^{\text{op}}$ be the universal functor

Then $\langle V, K \rangle \vdash (-, -)$, since the left adjoint acts on objects by $\langle V, K \rangle (S, T) = (S, S \to T)$, and there are natural isomorphisms

$$\text{hom}_{\mathcal{C} \times \mathcal{C}^{\text{op}}}(\langle V, K \rangle (S, T), (A, B)) = (S \to A) \times (B \to (S \to T))$$

$$\cong (S \to A) \times (S \times B \to T)$$

$$= \text{hom}_{\text{Bilens}(\mathcal{C})}(S, T \to (A, B))$$

Naturality is the fact that for all morphisms $f : A \to A'$, $g : B' \to B$ in $\mathcal{C}$ and lenses $\lambda : (S', T') \to (S, T)$ the following diagram commutes:

4
4 Pullbacks of lenses

Since \( \text{Bilens}(C) \) has products, in order to prove that it is complete it suffices to prove that it has either equalisers or pullbacks. Although equalisers are simpler, we will focus on pullbacks instead because we are interested in the category \( \text{Span}(\text{Bilens}(C)) \), whose composition involves pullbacks in \( \text{Bilens}(C) \).

**Proposition 6.** Let \( C \) be complete, cartesian closed and have pushouts. Let \( \begin{array}{c} (S,T) \\ \downarrow \lambda \\ (A,B) \end{array} \rightarrow \begin{array}{c} (S',T') \\ \downarrow \lambda' \end{array} \) be a cospan in \( \text{Bilens}(C) \). The pullback of the cospan is \( \begin{array}{c} S \times_A S' \\ \downarrow \pi_1 \times_B \pi_2 \\ T + (S \times_A S') \times B \end{array} \rightarrow \begin{array}{c} (S \times_A S') \times B \\ \downarrow \pi_1 \times_B \pi_2 \\ (S \times_A S') \times B \end{array} \rightarrow \begin{array}{c} T' \end{array} \), where \( S \times_A S' \) is the pullback in \( C \) of \( \begin{array}{c} S \\ \downarrow u \end{array} \rightarrow \begin{array}{c} A \\ \downarrow \lambda' \end{array} \leftarrow \begin{array}{c} S' \end{array} \), and \( T + (S \times_A S') \times B \) is the pushout in \( C \) of

\[ T \leftarrow S \times B \xrightarrow{\pi_1 \times_B} (S \times_A S') \times B \xrightarrow{\pi_2 \times_B} S' \times B \xrightarrow{u \times_B} T' \]

The projection lenses

\[ \begin{array}{c} S \\ \downarrow \pi_1 \end{array} \rightarrow \begin{array}{c} S \times_A S' \\ \downarrow \lambda \end{array} \xleftarrow{\pi_2} \begin{array}{c} T' \end{array} \]

have view morphisms \( S \xleftarrow{\pi_1} S \times_A S' \xrightarrow{\pi_2} S' \), and update morphisms

\( (S \times_A S') \times T \xrightarrow{\pi_2} T \xrightarrow{\pi_1} T + (S \times_A S') \times B T' \xrightarrow{\pi_2} T' \xleftarrow{\pi_2} (S \times_A S') \times T' \)

**Proof.** Suppose we have lenses
We must show that there is a unique lens $\alpha$ making the triangles commute. The view morphism $v_\alpha$ is given by the universal morphism of the pullback

Since $\mathcal{C}$ is cartesian closed, the functor $P \times -$ preserves colimits, so the diagram

is a pushout. The update morphism $u_\alpha$ is given by the universal morphism
It remains to show uniqueness. For the triangle

\[
\begin{array}{c}
\Delta
\end{array}
\]

in \textbf{Bilens}(C) to commute is equivalent to having

\[
\begin{array}{c}
P \\
Q
\end{array}
\]

\[
\begin{array}{c}
\alpha \\
\mu
\end{array}
\]

\[
\begin{array}{c}
S \times_A S' \\
T + (S \times_A S') \times B T'
\end{array}
\]

\[
\begin{array}{c}
\pi_1 \\
(S) \\
(T)
\end{array}
\]

and

\[
\begin{array}{c}
P \\
S \times_A S'
\end{array}
\]

\[
\begin{array}{c}
v_\alpha \\
\pi_1 \\
v_\mu
\end{array}
\]
$P \times T \xrightarrow{u_\mu} Q$

$\Delta_P \times T$

$P \times P \times T \xrightarrow{P \times u_{\iota_1}} P \times (S \times_A S') \times T$

$u_\alpha$

$P \times (T + (S \times_A S') \times_B T')$

$P \times v_\alpha \times T$

$P \times (S \times_A S') \times T$

$\pi_2$

$\iota_1$

$v_\alpha$ is the unique morphism making the triangle in $C$ commute. (The other triangle in $\text{Bilens}(C)$ is exactly symmetric.)

Since the lower two shapes always commute, the whole commutes iff the upper triangle commutes. $u_\mu$ is the unique morphism with this property.

We have therefore proven:

**Theorem 1.** Let $C$ be complete, cocomplete and cartesian closed. Then $\text{Bilens}(C)$ is complete.

Note that $\text{Bilens}(C)$ is generally not cocomplete, and does not even have all coproducts. However those coproducts that it does have are applied to game theory in [Hed18], and it has a ‘weak coproduct’ that is applied to the semantics of linear logic in [AP91].
References

[BG18] Guillaume Boisseau and Jeremy Gibbons. What you need to know about Yoneda: profunctor optics and the Yoneda lemma (functional pearl). In Proceedings of International Conference on Functional Programming (ICFP) 2018, Proceedings of the ACM on Programming Languages. ACM, 2018.

[dP91] Valeria de Paiva. The dialectica categories. Technical report, University of Cambridge, 1991.

[FGM+07] Nate Foster, Michael Greenwald, Jonathan Moore, Benjamin Pierce, and Alan Schmitt. Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. ACM Transactions on Programming Languages and Systems, 29(3), 2007.

[Hed17] Jules Hedges. Coherence for lenses and open games. arXiv:1704.02230, 2017.

[Hed18] Jules Hedges. Morphisms of open games. arXiv:1711.07059, to appear in Mathematical Foundations of Programming Semantics (MFPS) 2018, 2018.

[HPW11] Martin Hofmann, Benjamin Pierce, and Daniel Wagner. Symmetric lenses. In Proceedings of the 38th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages (POPL’11), volume 46, pages 371–384, 2011.

[Jac99] Bart Jacobs. Categorical logic and type theory. Studies in logic and the foundations of mathematics. Elsevier, 1999.

[JR14] Michael Johnson and Robert Rosebrugh. Spans of lenses. In Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference, volume 1133 of CEUR Workshop Proceedings, pages 112–118, 2014.

[Kme12] Edward Kmett. Mirrored lenses. http://comonad.com/reader/2012/mirrored-lenses/, 2012.

[PGW17] Matthew Pickering, Jeremy Gibbons, and Nicolas Wu. Profunctor optics: Modular data accessors. The art, science and engineering of programming, 1(2), 2017.