Chapter 2

Symmetric Lenses

In this chapter, we address the problem of symmetry without regard for alignment or performance issues. We will begin from asymmetric, state-based lenses and build a theory of symmetric, state-based lenses from them, and show how to recover the rich asymmetric syntax in symmetric form. In particular, we will show how to implement lens composition—the process of running two bidirectional transformations, one after the other—long thought to be an operation fundamentally in conflict with symmetric bidirectional presentations. In order to support this operation with the usual algebraic properties like associativity, we will need to develop a theory of behavioral equivalence. Unlike asymmetric theories, where ordinary equality suffices, our symmetric lenses have hidden state whose importance should be discounted when checking whether two lenses compute the same transformation. We will also discuss a collection of bidirectional operations which correspond to common transformations of container-based data types as well as inductive data types built up from products, sums, and type-level recursion. Finally, we will give an account of the connection between asymmetric and symmetric lenses: asymmetric lenses can be lifted to symmetric lenses, and symmetric lenses can be represented as a span of asymmetric lenses.

2.1 Fundamental Definitions

Complements The key step toward symmetric lenses is the notion of *complements*. The idea dates back to a famous paper in the database literature on the view update problem [5] and was adapted to lenses in [6] (and, for a slightly different definition, [29]), and it is quite simple. If we think of the *get* component of a lens as a sort of projection function, then we can find another projection from X into some set C that keeps all the information discarded by *get*. Equivalently, we can think of *get* as returning two results—an element of Y and an element of C—that together contain all the information needed to reconstitute the original element of X. Now the *put* function doesn't need a whole $x \in X$ to recombine with some updated $y \in Y$; it can just take the complement $c \in C$ generated from x by the *get*, since this will

contain all the information that is missing from y. Moreover, instead of a separate *create* function, we can simply pick a distinguished element $\mathit{missing} \in C$ and define $\mathit{create}(y)$ as $\mathit{put}(y, \mathit{missing})$.

Formally, an *asymmetric lens with complement* mapping between X and Y consists of a set C, a distinguished element $\mathit{missing} \in C$, and two functions

$$\begin{array}{rcl} \mathit{get} & \in & X \to Y \times C \\ \mathit{put} & \in & Y \times C \to X \end{array}$$

obeying the following laws for every $x \in X$, $y \in Y$, and $c \in C$:[1]

$$\frac{\mathit{get}\ x = (y, c)}{\mathit{put}\ (y, c) = x} \qquad \text{(GetPut)}$$

$$\frac{\mathit{get}\ (\mathit{put}\ (y, c)) = (b', c')}{b' = y} \qquad \text{(PutGet)}$$

Note that the type is just "lens from X to Y": the set C is an internal component, not part of the externally visible type. In symbols, $\mathit{Lens}(X, Y) = \exists C.\ \{\mathit{missing} : C,\ \mathit{get} : X \to Y \times C,\ \mathit{put} : Y \times C \to X\}$.

Symmetric Lenses Now we can symmetrize. First, instead of having only *get* return a complement, we make *put* return a complement too, and we take this complement as a second argument to *get*.

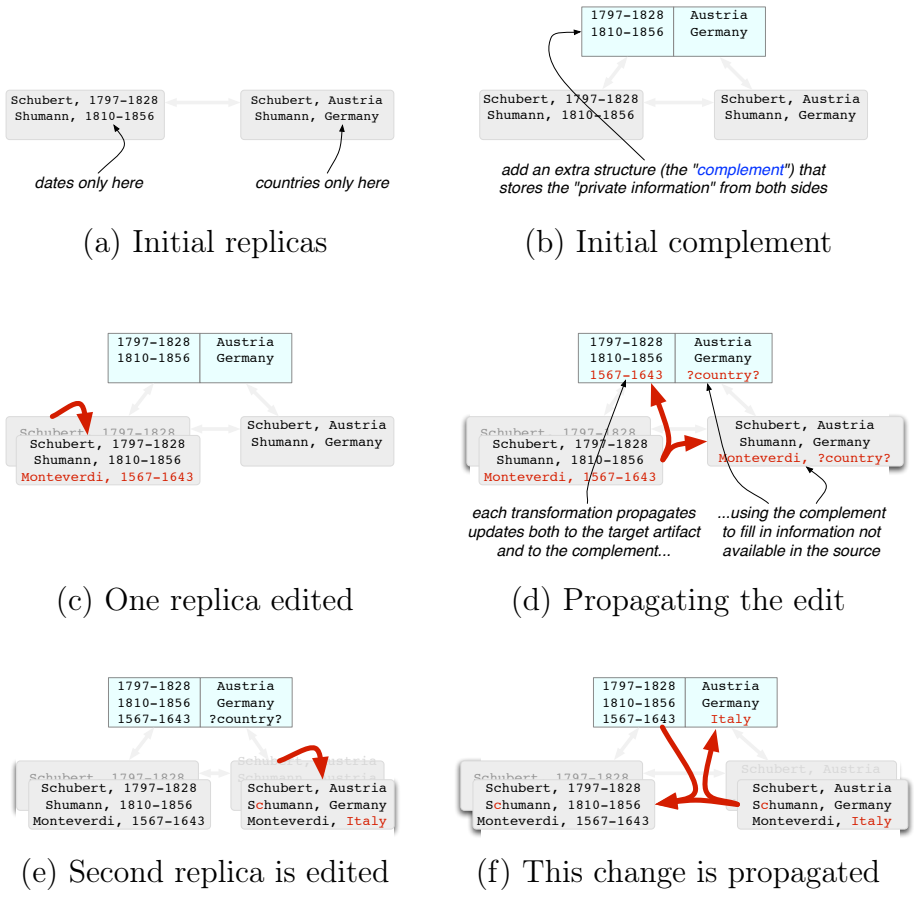
$$\begin{array}{rcl} \mathit{get} & \in & X \times C_Y \to Y \times C_X \\ \mathit{put} & \in & Y \times C_X \to X \times C_Y \end{array}$$

Intuitively, C_X is the "information from X that is discarded by *get*," and C_Y is the "information from Y that is discarded by *put*." Next we observe that we can, without loss of generality, use the same set C as the complement in both directions. (This "tweak" is actually critical: it is what allows us to define composition of symmetric lenses.)

$$\begin{array}{rcl} \mathit{get} & \in & X \times C \to Y \times C \\ \mathit{put} & \in & Y \times C \to X \times C \end{array}$$

We can think of the combined complement C as $C_X \times C_Y$—that is, each complement contains some "private information from X" and some "private information from Y"; by convention, the *get* function reads the C_Y part and writes the C_X part, while

[1] We can convert back and forth between the two presentations; in particular, if $(\mathit{get}, \mathit{put}, \mathit{create})$ are the components of a traditional lens, then we define a canonical complement by $C = \{f \in Y \to X \mid \forall y.\ \mathit{get}(f(y)) = y\}$. We then define the components $\mathit{missing}'$, get', and put' of an asymmetric lens with complement as $\mathit{missing}' = \mathit{create}$ and $\mathit{get}'(x) = (\mathit{get}(x), \lambda y.\mathit{put}(y, x))$ and $\mathit{put}'(y, f) = f(y)$. Going the other way, if $(\mathit{get}, \mathit{put}, \mathit{missing})$ are the components of an asymmetric lens with complement, we can define a traditional lens by $\mathit{get}'(x) = \mathsf{fst}(\mathit{get}(x))$ and $\mathit{put}'(y, x) = \mathit{put}(y, \mathsf{snd}(\mathit{get}(x)))$ and $\mathit{create}(y) = \mathit{put}(y, \mathit{missing})$.



(a) Initial replicas

(b) Initial complement

(c) One replica edited

(d) Propagating the edit

(e) Second replica is edited

(f) This change is propagated

Figure 2.1: Behavior of a symmetric lens

the *put* reads the C_X part and writes the C_Y part. Lastly, now that everything is symmetric, the *get* / *put* distinction is not helpful, so we rename the functions to *putr* and *putl*. This brings us to our core definition.

2.1.1 Definition [Symmetric lens]: A lens ℓ from X to Y (written $\ell \in X \leftrightarrow Y$) has three parts: a set of complements C, a distinguished element $\mathit{missing} \in C$, and two functions

$$\begin{aligned} \mathit{putr} &\in X \times C \to Y \times C \\ \mathit{putl} &\in Y \times C \to X \times C \end{aligned}$$

satisfying the following round-tripping laws:

$$\frac{\mathit{putr}(x, c) = (y, c')}{\mathit{putl}(y, c') = (x, c')} \qquad \text{(PutRL)}$$

$$\frac{\mathit{putl}(y, c) = (x, c')}{\mathit{putr}(x, c') = (y, c')} \qquad \text{(PutLR)}$$

When several lenses are under discussion, we use record notation to identify their parts, writing $\ell.C$ for the complement set of ℓ, etc.

The force of the PUTRL and PUTLR laws is to establish some "consistent" or "steady-state" triples (x, y, c), for which *put*s of x from the left or y from the right will have no effect—that is, will not change the complement. The conclusion of each rule has the same variable c' on both sides of the equation to reflect this. We will use the equation $putr(x, c) = (y, c)$ to characterize the steady states. In general, a *put* of a new x' from the left entails finding a y' and a c' that restore consistency. Additionally, we often wish this process to involve the complement c from the previous steady state; as a result, it can be delicate to choose a good value of *missing*. This value can often be chosen compositionally; each of our primitive lenses and lens combinators specify one good choice for *missing*.

Examples Figure 2.1 illustrates the use of a symmetric lens. The structures in this example are lists of textual records describing composers. The partially synchronized records (a) have a name and two dates on the left and a name and a country on the right. The complement (b) contains all the information that is discarded by both *put*s—all the dates from the left-hand structure and all the countries from the right-hand structure. (It can be viewed as a pair of lists of strings, or equivalently as a list of pairs of strings; the way we build list lenses later actually corresponds to the latter.) If we add a new record to the left hand structure (c) and use the *putr* operation to propagate it through the lens (d), we copy the shared information (the new name) directly from left to right, store the private information (the new dates) in the complement, and use a default string to fill in both the private information on the right and the corresponding right-hand part of the complement. If we now update the right-hand structure to fill in the missing information and correct a typo in one of the other names (e), then a *putl* operation will propagate the edited country to the complement, propagate the edited name to the other structure, and use the complement to restore the dates for all three composers.

Viewed more abstractly, the connection between the information about a single composer in the two tables is a lens from $X \times Y$ to $Y \times Z$, with complement $X \times Z$—let's call this lens e. Its *putr* component is given (x, y) as input and has (x', z) in its complement; it constructs a new complement by replacing x' by x to form (x, z), and it constructs its output by pairing the y from its input and the z from its complement to form (y, z). The *putl* component does the opposite, replacing the z part of the complement and retrieving the x part. Then the top-level lens in Figure 2.1—let's call it $e^\star$—abstractly has type $(X \times Y)^\star \leftrightarrow (Y \times Z)^\star$ and can be thought of as the "lifting" of e from elements to lists.

There are several plausible implementations of $e^\star$, with slightly different behaviors when list elements are added and removed—i.e., when the input and complement arguments to *putr* or *putl* are lists of different lengths. One possibility is to take $e^\star.C = (e.C)^\star$ and maintain the invariant that the complement list in the output

is the same length as the input list. When the lists in the input have different lengths, we can restore the invariant by either truncating the complement list or padding it with $e.missing$. For example, taking $X = \{a, b, c, \ldots\}$, $Y = \{1, 2, 3, \ldots\}$, $Z = \{A, B, C, \ldots\}$, and $e.missing = (m, M)$, and writing $\langle a, b, c\rangle$ for the sequence with the three elements a, b, and c, we could have:

$$\begin{array}{ll} & putr(\langle (a,1)\rangle\,,\ \langle (p,P),(q,Q)\rangle) \\ = & putr(\langle (a,1)\rangle\,,\ \langle (p,P)\rangle) \qquad\qquad \text{(truncating)} \\ = & (\langle (1,P)\rangle\,,\ \langle (a,P)\rangle) \end{array}$$

$$\begin{array}{ll} & putr(\langle (a,1),(b,2)\rangle\,,\ \langle (a,P)\rangle) \\ = & putr(\langle (a,1),(b,2)\rangle\,,\ \langle (a,P),(m,M)\rangle) \qquad \text{(padding)} \\ = & (\langle (1,P),(2,M)\rangle\,,\ \langle (a,P),(b,M)\rangle) \end{array}$$

Notice that, after the first *putr*, the information in the second element of the complement list (q, Q) is lost. The second *putr* creates a brand new second element for the list, so the value Q is gone forever; what's left is the default value M.

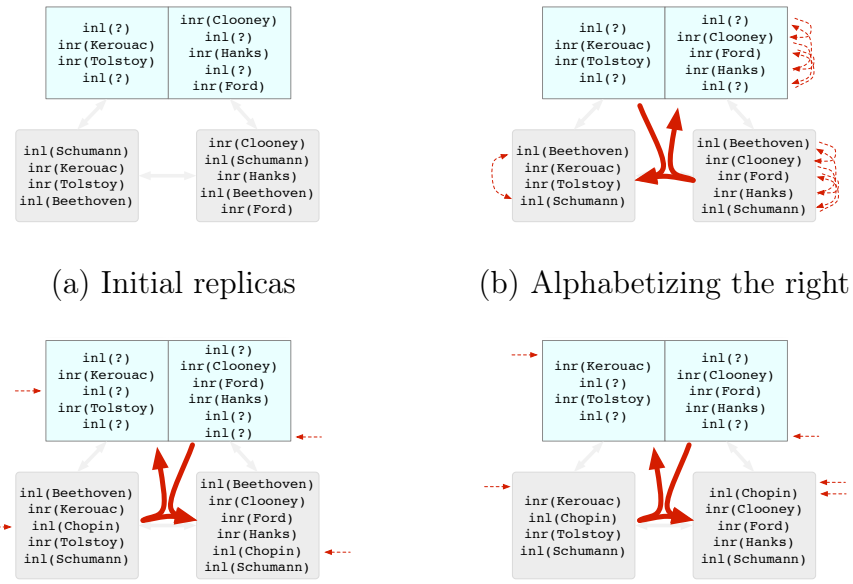
Another possibility—arguably better behaved—is to keep an *infinite* list of complements. Whenever we do a *put*, we use (and update) a prefix of the complement list of the same length as the current value being *put*, but we keep the infinite tail so that, later, we have values to use when the list being *put* is longer.

$$\begin{array}{ll} & putr(\langle (a,1)\rangle\,,\ \langle (p,P),(q,Q),(m,M),(m,M),\ldots\rangle) \\ = & (\langle (1,P)\rangle\,,\ \langle (a,P),(q,Q),(m,M),(m,M),\ldots\rangle) \end{array}$$

$$\begin{array}{ll} & putr(\langle (a,1),(b,2)\rangle\,,\ \langle (a,P),(q,Q),(m,M),(m,M),\ldots\rangle) \\ = & (\langle (1,P),(2,Q)\rangle\,,\ \langle (a,P),(b,Q),(m,M),\ldots\rangle) \end{array}$$

We call the first form the *forgetful* list mapping lens and the second the *retentive* list mapping lens. We will see, later, that the difference between these two precisely boils down to a difference in the behavior of the lens-summing operator $\oplus$ in the specification $e^\star \simeq id_{Unit} \oplus (e \otimes e^\star)$ of the list mapping lens.

Figure 2.2 illustrates another use of symmetric lenses. The structures in this example are lists of categorized data; each name on the left is either a composer (tagged `inl`) or an author (tagged `inr`), and each name on the right is either a composer or an actor. The lens under consideration will synchronize just the composers between the two lists, leaving the authors untouched on the left and the actors untouched on the right. The synchronized state (a) shows a complement with two lists, each with holes for the composers. If we re-order the right-hand structure (b), the change in order will be reflected on the left by swapping the two composers. Adding another composer on the left (c) involves adding a new hole to each complement; on the left, the location of the hole is determined by the new list, and on the right it simply shows up at the end. Similarly, if we remove a composer (d), the final hole on the other side disappears.



(a) Initial replicas (b) Alphabetizing the right

(c) Inserting Chopin on the left (d) Deleting Beethoven from the left

Figure 2.2: Synchronizing lists of sums

Abstractly, to achieve this behavior we need to define a lens *comp* between $(X+Y)^\star$ and $(X+Z)^\star$. To do this, it is convenient to first define a lens that connects $(X+Y)^\star$ and $X^\star \times Y^\star$; call this lens *partition*. The complement of the *partition* is a list of booleans telling whether the corresponding element of the left list is an X or a Y. The *putr* function is fairly simple: we separate the $(X+Y)$ list into X and Y lists by checking the tag of each element, and set the complement to exactly match the tags. For example:

$$putr(\langle \mathsf{inl}\ a, \mathsf{inl}\ b, \mathsf{inr}\ 1\rangle, c) = ((\langle a, b\rangle, \langle 1\rangle), \langle \mathsf{false}, \mathsf{false}, \mathsf{true}\rangle)$$
$$putr(\langle \mathsf{inl}\ a, \mathsf{inr}\ 1, \mathsf{inl}\ b\rangle, c) = ((\langle a, b\rangle, \langle 1\rangle), \langle \mathsf{false}, \mathsf{true}, \mathsf{false}\rangle)$$

These examples demonstrate that *putr* ignores the complement entirely, fabricating a completely new one from its input. The *putl* function, on the other hand, relies entirely on the complement for its ordering information. When there are extra entries (not accounted for by the complement), it adds them at the end. Consider taking the output of the second *putr* above and adding c to the X list and 2 to the Y list:

$$\begin{aligned} &putl((\langle a, b, c\rangle, \langle 1, 2\rangle), \langle \mathsf{false}, \mathsf{true}, \mathsf{false}\rangle) = \\ &\quad (\langle \mathsf{inl}\ a, \mathsf{inr}\ 1, \mathsf{inl}\ b, \mathsf{inl}\ c, \mathsf{inr}\ 2\rangle, \\ &\quad \langle \mathsf{false}, \mathsf{true}, \mathsf{false}, \mathsf{false}, \mathsf{true}\rangle) \end{aligned}$$

The *putl* fills in as much of the beginning of the list as it can, using the complement to indicate whether to draw elements from $X^\star$ or from $Y^\star$. (How the remaining X and Y elements are interleaved is a free choice, not specified by the lens laws, since this case only arises when we are *not* in a round-tripping situation. The strategy shown

here, where all new X entries precede all new Y entries, is just one possibility.)

Given *partition*, we can obtain *comp* by composing three lenses in sequence: from $(X + Y)^\star$ we get to $X^\star \times Y^\star$ using *partition*, then to $X^\star \times Z^\star$ using a variant of the lens e discussed above, and finally to $(X + Z)^\star$ using a "backwards" *partition*.

Put-Put Laws

2.1.2 Lemma: The following "put the same thing twice" laws follow from the ones we have:

$$\frac{putr(x, c) = (y, c')}{putr(x, c') = (y, c')} \qquad \text{(PutR2)}$$

$$\frac{putl(y, c) = (x, c')}{putl(y, c') = (x, c')} \qquad \text{(PutL2)}$$

We could consider generalizing these to say that putting an arbitrary pair of values, one after the other, is the same as doing just the second *put* into the first complement:

$$\frac{putr(x, c) = (\_\,, c')}{putr(x', c') = putr(x', c)} \qquad \text{(Strong-PutPutR*)}$$

$$\frac{putl(y, c) = (\_\,, c')}{putl(y', c') = putl(y', c)} \qquad \text{(Strong-PutPutL*)}$$

But these laws are very strong—probably too strong to be useful (the * annotations in their names are a reminder that we do *not* adopt them). The reason is that they demand that the effect of every update is completely undoable—not only the effect on the other replica, but also the effect of the first update on the complement must be completely forgotten if we make a second update. In particular, neither of the list-mapping lenses in §2.5 satisfy these laws.

A weaker version of these laws, constraining the output but not the effect on the complement, may be more interesting:

$$\frac{\begin{array}{c} putr(x, c) = (\_\,, c') \\ putr(x', c) = (y, \_) \\ putr(x', c') = (y', \_) \end{array}}{y = y'} \qquad \text{(Weak-PutPutR*)}$$

$$\frac{\begin{array}{c} putl(y, c) = (\_\,, c') \\ putl(y', c) = (x, \_) \\ putl(y', c') = (x', \_) \end{array}}{x = x'} \qquad \text{(Weak-PutPutL*)}$$

We do not choose to adopt these laws here because they are not satisfied by the "forgetful" variants of our summing and list mapping lenses. However, the forgetful

variants are mainly interesting because of their close connection to analogous asymmetric lenses; in practice, the "retentive" variants seem more useful, and these do satisfy the weak PUTPUT laws.

Alignment One important *non*-goal of the present chapter is dealing with the issue of alignment. We consider only the simple case of lenses that work "positionally." For example, the lens $e^\star$ in the example will always use e to propagate changes between the first element of x and the first element of y, between the second element of x and the second of y, and so on. This amounts to assuming that the lists are edited either by editing individual elements in place or by adding or deleting elements at the end of the list; if an actual edit inserts an element at the head of one of the lists, positional alignment will produce surprising (and probably distressing) results. We will incorporate a richer notion of alignment in Chapter 3.

2.2 Equivalence

Since each lens carries its own complement—and since this need not be the same as the complement of another lens with the same domain and codomain—we now need to define what it means for two lenses to be indistinguishable, in the sense that no user could ever tell the difference between them by observing just the X and Y parts of their outputs. We will use this relation pervasively in what follows: indeed, most of the laws we would like our constructions to validate—even things as basic as associativity of composition—will not hold "on the nose," but only up to equivalence.

2.2.1 Definition [R-similarity]: Given sets X, Y, C_f, C_g and a relation $R \subset C_f \times C_g$, we say that functions $f \in X \times C_f \to Y \times C_f$ and $g \in X \times C_g \to Y \times C_g$ are *R-similar*, written $f \sim_R g$, if they take inputs with R-related complements to equal outputs with R-related complements:

$$\frac{\begin{array}{c}(c_f, c_g) \in R \\ f(x, c_f) = (y_f, c'_f) \\ g(x, c_g) = (y_g, c'_g)\end{array}}{y_f = y_g \wedge (c'_f, c'_g) \in R}$$

2.2.2 Definition [Lens equivalence]: Two lenses k and ℓ are *equivalent* (written $k \equiv \ell$) if there is a relation $R \subset k.C \times \ell.C$ on their complement sets with

1. $(k.missing, \ell.missing) \in R$
2. $k.putr \sim_R \ell.putr$
3. $k.putl \sim_R \ell.putl$.

We write $X \Longleftrightarrow Y$ for the set of equivalence classes of lenses from X to Y. When ℓ is a lens, we write $[\ell]$ for the equivalence class of ℓ (that is, $\ell \in X \leftrightarrow Y$ iff $[\ell] \in X \Longleftrightarrow Y$). Where no confusion results, we abuse notation and drop these brackets, using ℓ for both a lens and its equivalence class.

2.2.3 Lemma: Lens equivalence is an equivalence relation.

Proof: Reflexivity and symmetry are obvious. We briefly sketch transitivity.

Suppose $k \equiv \ell$ (as witnessed by $R_{k\ell}$) and $\ell \equiv m$ (as witnessed by $R_{\ell m}$). We show that the relation

$$R_{km} = R_{k\ell} \circ R_{\ell m} = \{(c_k, c_m) \mid \exists c_\ell.\ c_k\ R_{k\ell}\ c_\ell \wedge c_\ell\ R_{\ell m}\ c_m\}$$

witnesses the equivalence $k \equiv m$. It is clear that

$$(k.missing, m.missing) \in R_{km},$$

since we can choose $c_\ell = \ell.missing$. Next, we show that $k.putr \sim_{R_{km}} m.putr$. We may assume three things:

$$\begin{aligned}(c_k, c_m) &\in R_{km}\\ k.putr(x, c_k) &= (y_k, c'_k)\\ m.putr(x, c_m) &= (y_m, c'_m)\end{aligned}$$

Since $(c_k, c_m) \in R_{km}$, we can choose c_ℓ such that $(c_k, c_\ell) \in R_{k\ell}$ and $(c_\ell, c_m) \in R_{\ell m}$. Choosing $(y_\ell, c'_\ell) = \ell.putr(x, c_\ell)$, we then conclude that $y_k = y_\ell$ and $(c'_k, c'_\ell) \in R_{k\ell}$, since $k.putr \sim_{R_{k\ell}} \ell.putr$. Similarly, we can conclude that $y_\ell = y_m$ and $(c'_\ell, c'_m) \in R_{\ell m}$ because $\ell.putr \sim_{R_{\ell m}} m.putr$. Thus $y_k = y_m$ and because of the existence of c'_ℓ, we know $(c'_k, c'_m) \in R_{km}$. But these are exactly the two facts we need to conclude that $k.putr \sim_{R_{km}} m.putr$. A similar argument shows that $k.putl \sim_{R_{km}} m.putl$, and hence that $k \equiv m$. □

2.2.4 Definition [Put object]: Given a lens $\ell \in X \leftrightarrow Y$, define a *put object* for ℓ to be a member of $X + Y$. Define a function *apply* taking a lens, an element of that lens' complement set, and a list of put objects as follows (using ML-like syntax):

$$\begin{aligned}apply(\ell, c, (\mathsf{inl}\ x){:}puts) &= \mathsf{let}\ (y, c') = \ell.putr(x, c)\ \mathsf{in}\\ &\quad\ (\mathsf{inr}\ y){:}apply(\ell, c', puts)\\ apply(\ell, c, (\mathsf{inr}\ y){:}puts) &= \mathsf{let}\ (x, c') = \ell.putl(y, c)\ \mathsf{in}\\ &\quad\ (\mathsf{inl}\ x){:}apply(\ell, c', puts)\\ apply(\ell, c, \langle\rangle) &= \langle\rangle\end{aligned}$$

2.2.5 Definition [Observational equivalence]: Lenses $k, \ell \in X \leftrightarrow Y$ are *observationally equivalent* (written $k \approx \ell$) if, for every sequence of put objects $P \in (X+Y)^\star$

we have

$$apply(k, k.missing, P) = apply(\ell, \ell.missing, P).$$

2.2.6 Theorem [Equivalence of equivalence]: $k \approx \ell$ iff $k \equiv \ell$.

Proof: ($\Longleftarrow$) Suppose that $k \equiv \ell$ via relation R. For all sequences of put objects P, and for elements $c \in k.C$ and $d \in k.C$ such that $(c, d) \in R$, we have $apply(k, c, P) = apply(\ell, d, P)$. This follows by induction on the length of P from the definition of *apply*. Thus, $k \approx \ell$ follows by specialization to $c = k.missing$ and $d = \ell.missing$.

($\Longrightarrow$) Now suppose $k \approx \ell$. To show $k \equiv \ell$, define $R \subseteq k.C \times \ell.C$ by

$$R = \{(c, d) \mid apply(k, c, P) = apply(\ell, d, P) \text{ for all } P\}.$$

By assumption, we have $(k.missing, \ell.missing) \in R$.

Now suppose that $(c, d) \in R$ and that $k.putr(x, c) = (y, c')$ and $\ell.putr(x, d) = (y', d')$. Applying the assumption $(c, d) \in R$ to the length-one sequence $P = \langle \mathsf{inl}\ (x) \rangle$ shows $y = y'$. To show $(c', d') \in R$ let P be an arbitrary sequence of put objects and define $P' = \mathsf{inl}\ (x){:}P$. The assumption $(c, d) \in R$ gives $apply(k, c, P') = apply(\ell, d, P')$, hence in particular $apply(k, c', P) = apply(\ell, d', P)$, thus $(c', d') \in R$. We have thus shown that $k.putr \sim_R \ell.putr$. Analogously, we show that $k.putl \sim_R \ell.putl$, and it follows that $k \equiv \ell$ via relation R. □

2.3 Basic Constructions

With the basic definitions in hand, we can start defining lenses. We begin in this section with several relatively simple constructions.

2.3.1 Definition [Identity lens]: Let *Unit* be a distinguished singleton set and () its only element.

$$id_X \in X \leftrightarrow X$$

$$\begin{array}{lcl} C & = & Unit \\ missing & = & () \\ putr(x, ()) & = & (x, ()) \\ putl(x, ()) & = & (x, ()) \end{array}$$

To check that this definition is well formed, we must show that the components defined in the lower box satisfy the round-trip laws implied by the upper box. The proof is a straightforward calculation.

2.3.2 Definition [Lens composition]:

$$\frac{k \in X \leftrightarrow Y \qquad \ell \in Y \leftrightarrow Z}{k;\ell \in X \leftrightarrow Z}$$

$$\begin{array}{lcl}
C & = & k.C \times \ell.C \\
\mathit{missing} & = & (k.\mathit{missing}, \ell.\mathit{missing}) \\
\mathit{putr}(x,(c_k,c_\ell)) & = & \mathsf{let}\ (y,c'_k) = k.\mathit{putr}(x,c_k)\ \mathsf{in} \\
 & & \mathsf{let}\ (z,c'_\ell) = \ell.\mathit{putr}(y,c_\ell)\ \mathsf{in} \\
 & & (z,(c'_k,c'_\ell)) \\
\mathit{putl}(z,(c_k,c_\ell)) & = & \mathsf{let}\ (y,c'_\ell) = \ell.\mathit{putl}(z,c_\ell)\ \mathsf{in} \\
 & & \mathsf{let}\ (x,c'_k) = k.\mathit{putl}(y,c_k)\ \mathsf{in} \\
 & & (x,(c'_k,c'_\ell))
\end{array}$$

Proof of well-formedness: We show that the lens satisfies PutRL; the proof that it satisfies PutLR is entirely symmetric. Assume that k and ℓ each satisfy PutRL, and that $(k;\ell).\mathit{putr}(x,(c_k,c_\ell)) = (z,(c'_k,c'_\ell))$. From the definition of $(k;\ell).\mathit{putr}$, we can conclude that there is a y such that $k.\mathit{putr}(x,c_k) = (y,c'_k)$ and $\ell.\mathit{putr}(y,c_\ell) = (z,c'_\ell)$.

$$\begin{array}{rcll}
(k;\ell).\mathit{putl}(z,(c'_k,c'_\ell)) & = & \mathsf{let}\ (y',c''_\ell) = \ell.\mathit{putl}(z,c'_\ell)\ \mathsf{in} & (2.3.1) \\
 & & \mathsf{let}\ (x',c''_k) = k.\mathit{putl}(y',c'_k)\ \mathsf{in} & \\
 & & (x',(c''_k,c''_\ell)) & \\
 & = & \mathsf{let}\ (y',c''_\ell) = (y,c'_\ell)\ \mathsf{in} & (2.3.2) \\
 & & \mathsf{let}\ (x',c''_k) = k.\mathit{putl}(y',c'_k)\ \mathsf{in} & \\
 & & (x',(c''_k,c''_\ell)) & \\
 & = & \mathsf{let}\ (x',c''_k) = k.\mathit{putl}(y,c'_k)\ \mathsf{in} & (2.3.3) \\
 & & (x',(c''_k,c'_\ell)) & \\
 & = & \mathsf{let}\ (x',c''_k) = (x,c'_k)\ \mathsf{in} & (2.3.4) \\
 & & (x',(c''_k,c'_\ell)) & \\
 & = & (x,(c'_k,c'_\ell)) & (2.3.5)
\end{array}$$

Equation 2.3.1 comes from expanding the definition of $(k;\ell).\mathit{putl}$; equation 2.3.2 from applying PutRL to ℓ; equation 2.3.3 from substituting let-bound variables; equation 2.3.4 from applying PutRL to k; and equation 2.3.5 from again substituting let-bound variables. Moreover, this last equation is exactly what is demanded from applying PutRL to $k;\ell$, so we are done. □

This definition specifies what it means to compose two lenses. To show that this definition lifts to equivalence classes of lenses, we need to check the following congruence property.

2.3.3 Lemma [Composition preserves equivalence]: If $k \equiv k'$ and $\ell \equiv \ell'$, then $k;\ell \equiv k';\ell'$.

2.3.4 Definition: The following function on relations is convenient here:

$$R_1 \times R_2 = \{((c_1, c_2), (c_1', c_2')) \mid (c_1, c_1') \in R_1 \wedge (c_2, c_2') \in R_2\}$$

Proof of 2.3.3: If the simulation R_k witnesses $k \equiv k'$ and R_ℓ witnesses $\ell \equiv \ell'$ then it is straightforward to verify that $R = R_k \times R_\ell$ witnesses $k;\ell \equiv k';\ell'$. There are three things to show.

1. We wish to show the first line:

$$\begin{aligned} & (k;\ell).\mathit{missing}\ R\ (k';\ell').\mathit{missing} \\ \iff & (k.\mathit{missing}, \ell.\mathit{missing})\ R\ (k'.\mathit{missing}, \ell'.\mathit{missing}) \\ \iff & k.\mathit{missing}\ R_k\ k'.\mathit{missing} \wedge \ell.\mathit{missing}\ R_\ell\ \ell'.\mathit{missing} \end{aligned}$$

   But the final line is certainly true, since R_k and R_ℓ are simulation relations.

2. We must show that $(k;\ell).\mathit{putr} \sim_R (k';\ell').\mathit{putr}$. So take $c_k, c_\ell, c_{k'}, c_{\ell'}$ such that $(c_k, c_\ell)\ R\ (c_{k'}, c_{\ell'})$ and choose an input x. Define the following:

$$\begin{aligned} (y, c_k') &= k.\mathit{putr}(x, c_k) \\ (z, c_\ell') &= \ell.\mathit{putr}(y, c_\ell) \\ (y', c_{k'}') &= k'.\mathit{putr}(x, c_{k'}) \\ (z', c_{\ell'}') &= \ell'.\mathit{putr}(y', c_{\ell'}) \end{aligned}$$

   We can then compute:

$$\begin{aligned} (k;\ell).\mathit{putr}(x, (c_k, c_\ell)) &= (z, (c_k', c_\ell')) \\ (k';\ell').\mathit{putr}(x, (c_{k'}, c_{\ell'})) &= (z', (c_{k'}', c_{\ell'}')) \end{aligned}$$

   We need to show that $z = z'$ and that $(c_k', c_\ell')\ R\ (c_{k'}', c_{\ell'}')$. Since $c_k\ R_k\ c_{k'}$, we can conclude that $y = y'$ and $c_k'\ R_k\ c_{k'}'$; similarly, since $c_\ell\ R_\ell\ c_{\ell'}$ and $y = y'$, we know that $z = z'$ (discharging one of our two proof burdens) and $c_\ell'\ R_\ell\ c_{\ell'}'$. Combining the above facts, we find that $(c_k', c_\ell')\ R\ (c_{k'}', c_{\ell'}')$ by definition of R (discharging the other proof burden).

3. The proof that $(k;\ell).\mathit{putl} \sim_R (k';\ell').\mathit{putl}$ is similar to the *putr* case. □

2.3.5 Lemma [Associativity of composition]:

$$j;(k;\ell) \equiv (j;k);\ell$$

(The equivalence is crucial here: $j;(k;\ell)$ and $(j;k);\ell$ are not the same lens because their complements are structured differently.)

Proof: We define a witnessing simulation relation R by

$$R = \{((c_1, (c_2, c_3)), ((c_1, c_2), c_3)) \mid c_1 \in j.C, c_2 \in k.C, c_3 \in \ell.C\}.$$

The verification is then straightforward. □

2.3.6 Lemma [Identity arrows]: The identity lens is a left and right identity for composition:

$$id_X; \ell \equiv \ell; id_Y \equiv \ell$$

Proof: For left identity we use the simulation relation R given by $((), c)\ R\ c$ whenever $c \in \ell.C$. The verification is direct.

The proof of the right-identity law $\ell; id \equiv \ell$ is analogous. □

Thus symmetric lenses form a category, LENS, with sets as objects and equivalence classes of lenses as arrows. The identity arrow for a set X is $[id_X]$. Composition is $[k]; [\ell] = [k; \ell]$.

2.3.7 Proposition [Bijective lenses]: Every bijective function gives rise to a lens:

$$\frac{f \in X \to Y \qquad f \text{ bijective}}{bij_f \in X \leftrightarrow Y}$$

$$\begin{array}{lcl} C & = & Unit \\ missing & = & () \\ putr(x, ()) & = & (f(x), ()) \\ putl(y, ()) & = & (f^{-1}(y), ()) \end{array}$$

(If we were implementing a bidirectional language, we might not want to expose *bij* in its syntax, since we would then need to offer programmers some notation for writing down bijections in such a way that we can verify that they *are* bijections and derive their inverses. However, even if it doesn't appear in the surface syntax, we will see several places where *bij* is useful in talking about the algebraic theory of symmetric lenses.)

Proof of well-formedness: We verify that the PUTRL law holds for bijection lenses; the proof that PUTLR holds is symmetric. Observe that $bij_f.putr(x, ()) = (f(x), ())$. We can therefore compute that $bij_f.putl(f(x), ()) = (f^{-1}(f(x)), ()) = (x, ())$. Thus, after a round-trip, we return to the same x we started from—and the same complement, (), validating the law. □

In fact, any stateless lens is an instance of a bijection lens:

2.3.8 Lemma: If $\ell \in X \leftrightarrow Y$ and $h \in \ell.C \to \mathit{Unit}$ is a bijection, then there exists a bijection $f \in X \to Y$ such that $\ell \equiv \mathit{bij}_f$.

Proof: We define:

$$f(x) = \mathsf{fst}(\ell.\mathit{putr}(x, h^{-1}(())))$$

We must show that f is bijective and that $\mathit{bij}_f \equiv \ell$. For the former, we exhibit its inverse in g:

$$g(y) = \mathsf{fst}(\ell.\mathit{putl}(y, h^{-1}(())))$$

The round-trip law PUTRL guarantees that $g(f(x)) = x$, and the round-trip law PUTLR guarantees that $f(g(y)) = y$.

To show the latter, we argue that h witnesses the equivalence. Clearly

$$h(\mathit{bij}_f.\mathit{missing}) = \ell.\mathit{missing}$$

because all elements of $\ell.C$ are equal (and hence $(\mathit{bij}_f.\mathit{missing}, \ell.\mathit{missing}) \in h$). The definition of f makes it clear that $\mathit{bij}_f.\mathit{putr} \sim_h \ell.\mathit{putr}$; similarly, the definition of f's inverse g makes it clear that $\mathit{bij}_f.\mathit{putl} \sim_h \ell.\mathit{putl}$. □

2.3.9 Corollary: If $\ell.C$ is a singleton set $\{c\}$ and $\mathsf{fst}(\ell.\mathit{putr}(x, c)) = x$ for all x, then $\ell \equiv \mathit{id}$.

This transformation (like several others we will see) respects much of the structure available in our category. Formally, *bij* is a functor. Recall that a *covariant* (respectively, *contravariant*) *functor* between categories $\mathcal{C}$ and $\mathcal{D}$ is a pair of maps—one from objects of $\mathcal{C}$ to objects of $\mathcal{D}$ and the other from arrows of $\mathcal{C}$ to arrows of $\mathcal{D}$—that preserve typing, identities, and composition:

- The image of any arrow $f : X \to Y$ in $\mathcal{C}$ has the type $F(f) : F(X) \to F(Y)$ (respectively, $F(f) : F(Y) \to F(X)$) in $\mathcal{D}$.
- For every object X in $\mathcal{C}$, we have $F(\mathit{id}_X) = \mathit{id}_{F(X)}$ in $\mathcal{D}$.
- If $f; g = h$ in $\mathcal{C}$, then $F(f); F(g) = F(h)$ (respectively, $F(g); F(f) = F(h)$) in $\mathcal{D}$.

Covariant functors are simply called functors. When it can be inferred from the arrow mapping, the object mapping is often elided.

2.3.10 Lemma [Embedding bijections]: The *bij* operator forms a functor from the category ISO, whose objects are sets and whose arrows are isomorphic functions, to LENS—that is, $\mathit{bij}_{\mathit{id}_X} = \mathit{id}_X$ and $\mathit{bij}_f; \mathit{bij}_g = \mathit{bij}_{f;g}$.

Proof: Showing that $bij_{id_X} = id_X$ is a straightforward application of Corollary 2.3.9. Now consider $bij_f; bij_g$. Since its complement is a singleton set, Lemma 2.3.8 tells us that $bij_f; bij_g \equiv bij_h$, where

$$h(x) = \mathsf{fst}((bij_f; bij_g).putr(x, ((), ()))),$$

which can be reduced to:

$$h(x) = g(f(x))$$

Thus $bij_f; bij_g \equiv bij_{f;g}$ as desired. □

Since functors preserve isomorphisms it follows that bijective lenses are isomorphisms in the category of lenses. However, not every isomorphism in LENS is of that form. This is because a bijective lens displays no dependency on the complement at all, whereas an isomorphism in the category of lenses still allows for some limited interaction with the complement as in the following counterexample.

Define the set $Trit = \{-1, 0, 1\}$ and the function $f \in Trit \times Trit \to Trit$ which returns its arguments if they are equal and the third possible value if they are not:

c	x	$f(c,x)$
-1	-1	-1
-1	0	1
-1	1	0
0	-1	1
0	0	0
0	1	-1
1	-1	0
1	0	-1
1	1	1

For any particular c, the partial application $f(c)$ is a bijection and an involution. Thus, we can define the following lens, which is its own inverse but is not equivalent to any bijective lens:

$$strange \in Trit \leftrightarrow Trit$$

$$\begin{array}{lcl} C & = & Unit + Trit \\ missing & = & \mathsf{inl}\ () \\ putr(x, \mathsf{inl}\ ()) & = & (x, \mathsf{inr}\ x) \\ putr(x, \mathsf{inr}\ c) & = & (f(c,x), \mathsf{inr}\ c) \\ putl(x, \mathsf{inl}\ ()) & = & (x, \mathsf{inr}\ x) \\ putl(x, \mathsf{inr}\ c) & = & (f(c,x), \mathsf{inr}\ c) \end{array}$$

We can show, however, that the *putr* and *putl* functions of any invertible lens induce a bijection between the two replicas for any pair of reachable complements. More precisely:

2.3.11 Lemma: Suppose we have lenses $k \in X \leftrightarrow Y$ and $\ell \in Y \leftrightarrow X$ such that $k;\ell \equiv id_X$ and $\ell;k \equiv id_Y$. Then there is a relation $R \subset k.C \times \ell.C$ satisfying the following conditions:

$$(k;\ell).missing \in R \tag{1}$$

$$\frac{(k;\ell).putr(x,c) = (x',c') \qquad c \in R}{x' = x \wedge c' \in R} \tag{2}$$

$$\frac{(k;\ell).putl(x,c) = (x',c') \qquad c \in R}{x' = x \wedge c' \in R} \tag{3}$$

$$\frac{(\ell;k).putr(y,c) = (y',c') \qquad \gamma^{\times}(c) \in R}{y' = y \wedge \gamma^{\times}(c') \in R} \tag{4}$$

$$\frac{(\ell;k).putl(y,c) = (y',c') \qquad \gamma^{\times}(c) \in R}{y' = y \wedge \gamma^{\times}(c') \in R} \tag{5}$$

Here, the function $\gamma^{\times}$ is the symmetry in SET, namely $\gamma^{\times}((x,y)) = (y,x)$.

Proof: We get an R_1 that satisfies 1-3 from the fact that $k;\ell \equiv id_X$, and we get an R_2 that satisfies 1, 4, and 5 from the fact that $\ell;k \equiv id_Y$. Then we can define $R = R_1 \cap R_2$. There are four conditions to check, but we will consider only one of them here, as the others are very similar:

$$\frac{(k;\ell).putr(x,c) = (x',c') \qquad c \in R}{c' \in R_2}$$

Now $c \in R$ means $c = (c_k, c_\ell)$ where $c_k \; R_1 \; c_\ell$ and $c_k \; R_2 \; c_\ell$. We can define

$$\begin{aligned} (y, c'_k) &= k.putr(x, c_k) \\ (x', c'_\ell) &= \ell.putr(y, c_\ell). \end{aligned}$$

Since R_1 satisfies 2, we know $x' = x$, that is, we know

$$\begin{aligned} \ell.putr(y, c_\ell) &= (x, c'_\ell) \\ k.putr(x, c_k) &= (y, c'_k). \end{aligned}$$

Now the fact that R_2 satisfies 4 above tells us that $c'_k \; R_2 \; c'_\ell$, that is, $c' \in R_2$. □

2.3.12 Corollary [Isomorphisms are indexed bijections]: Consider the functions f and g which give the value-only part of a lens' puts:

$$\begin{aligned} f_{\ell,c_\ell}(x) &= \mathsf{fst}(\ell.putr(x, c_\ell)) \\ g_{\ell,c_\ell}(x) &= \mathsf{fst}(\ell.putl(x, c_\ell)) \end{aligned}$$

If $c_k\ R\ c_\ell$ (using the R given by the previous lemma), then f_{k,c_k}, f_{ℓ,c_ℓ}, g_{k,c_k}, and g_{ℓ,c_ℓ} are all bijections.

Proof: For any $x \in X$, we know $f_{\ell,c_\ell}(f_{k,c_k}(x)) = x$ by 2, and for any $y \in Y$, we know $f_{k,c_k}(f_{\ell,c_\ell}(y)) = y$ by 4. Thus, not only is f_{k,c_k} a bijection, we actually have its inverse: $f^{-1}_{k,c_k} = f_{\ell,c_\ell}$! Similarly, $g^{-1}_{k,c_k} = g_{\ell,c_\ell}$. □

2.3.13 Definition [Dual of a lens]:

$$\frac{\ell \in X \leftrightarrow Y}{\ell^{op} \in Y \leftrightarrow X}$$

$$\begin{array}{lcl} C & = & \ell.C \\ \mathit{missing} & = & \ell.\mathit{missing} \\ \mathit{putr}(y,c) & = & \ell.\mathit{putl}(y,c) \\ \mathit{putl}(x,c) & = & \ell.\mathit{putr}(x,c) \end{array}$$

Proof of well-formedness: We observe that saying ℓ^{op} satisfies PutRL is an identical condition to saying ℓ satisfies PutLR, and likewise having ℓ^{op} satisfy PutLR is identical to having ℓ satisfy PutRL. □

It is easy to see that $(-)^{op}$ is involutive—that is, that $(\ell^{op})^{op} = \ell$ for every ℓ—and that $\mathit{bij}_{f^{-1}} = \mathit{bij}_f^{op}$ for any bijective f. Recalling that an endofunctor is a functor whose source and target categories are identical, we can easily show the following lemma.

2.3.14 Lemma: The $(-)^{op}$ operation can be lifted to a contravariant endofunctor on the category LENS, mapping each arrow $[\ell]$ to $[\ell^{op}]$.

Proof: We must show three things:

1. The mapping $[\ell] \mapsto [\ell^{op}]$ is well-defined, that is, that if $k \equiv \ell$, then $k^{op} \equiv \ell^{op}$.
2. The mapping respects identities, that is, that $id^{op} \equiv id$.
3. The mapping respects composition, that is, $(k;\ell)^{op} \equiv \ell^{op}; k^{op}$.

We sketch the proofs in that order.

1. If $k \equiv \ell$ is witnessed by R then $k^{op} \equiv \ell^{op}$ is also witnessed by R;
2. In fact, $id^{op} = id$; and

3. The relation $(c_k, c_\ell)\ R\ (c_\ell, c_k)$ whenever $c_k \in k.C$ and $c_\ell \in \ell.C$ witnesses the equivalence. □

The existence of $(-)^{op}$ is one of the two canonical constructions that motivate the name "symmetric lenses" (the other being *disconnect*, which we discuss below). Before we formalize this intuition, we review two standard constructions from category theory.

2.3.15 Definition: The *opposite* of a category $\mathcal{C}$, denoted $\mathcal{C}^{op}$, has backwards composition compared to $\mathcal{C}$. That is, whenever $f; g = h$ in $\mathcal{C}$, we have $g; f = h$ in $\mathcal{C}^{op}$. This induces the remaining components of $\mathcal{C}^{op}$:

Objects The objects of $\mathcal{C}^{op}$ are exactly the objects of $\mathcal{C}$.

Arrows The arrows $f : X \to Y$ of $\mathcal{C}^{op}$ are the arrows $f : Y \to X$ of $\mathcal{C}$.

Identities The identities of $\mathcal{C}^{op}$ are exactly the identities of $\mathcal{C}$.

That is, forming the opposite of a category means formally reversing the "direction" of each arrow. In general, a category and its opposite can have very different structure. What we want to show is that the directionality of arrows in LENS is not important; we can formalize this by saying that LENS and LENS^{op} have the same structure, provided we can formalize what it means for two categories have the same structure. There are many ways to define equivalence between categories; we give a particularly strong one here.

2.3.16 Definition: Categories $\mathcal{C}$ and $\mathcal{D}$ are isomorphic if there are functors $F : \mathcal{C} \to \mathcal{D}$ and $G : \mathcal{D} \to \mathcal{C}$ for which $F; G$ is the identity on $\mathcal{C}$ and $G; F$ is the identity on $\mathcal{D}$.

2.3.17 Corollary: The category LENS is self dual, i.e., isomorphic to LENS^{op}. (Note that this does not mean that each arrow is its own inverse!)

Proof: The arrow part of $(-)^{op}$ is bijective. □

The lenses we have discussed so far maintain all the information in the domain and codomain. It is sometimes useful to discard some information in one direction of the lens. The terminal lens does this, recording the discarded information in the complement so that the other direction of the lens can restore it.

2.3.18 Definition [Terminal lens]:

$$\frac{x \in X}{term_x \in X \leftrightarrow Unit}$$

$$\begin{array}{lcl} C & = & X \\ missing & = & x \\ putr(x', c) & = & ((), x') \\ putl((), c) & = & (c, c) \end{array}$$

Proof of well-formedness: The PUTLR law is trivially true, since

$$putr(putl((), c)) = putr(c, c) = ((), c)$$

and in particular since c does not change at all in this round trip. We also observe:

$$putl(putr(x, c)) = putl((), x) = (x, x)$$

Since the complement x does not change during the *putl* and we arrive back at the value x that we started with, this verifies that PUTRL holds as well. □

2.3.19 Proposition [Uniqueness of terminal lens]: Lenses with the same type as a terminal lens are equivalent to a terminal lens. More precisely, suppose $k \in X \leftrightarrow Unit$ and $k.putl((), k.missing) = (x, c)$. Then $k \equiv term_x$.

Of course, there may be many (pairwise non-equivalent) terminal lenses of a particular type; for any two $x, y \in X$ with $x \neq y$, it's clear that $term_x \not\equiv term_y$. Proposition 2.3.19 tells us that there are exactly as many arrows $\ell : X \Longleftrightarrow Unit$ as there are elements of X.

Proof: The behavior of k is uniquely defined by the given data: *putl* must return x the first time and echo the last *putr* henceforth. Formally, we may define a simulation relation as follows:

$$R = \{(c, y) \mid \mathsf{fst}(k.putl((), c)) = y\}$$

It's clear that $k.missing\ R\ x$, since we have chosen x specifically so that

$$\mathsf{fst}(k.putl((), k.missing)) = x.$$

Let us show next that $k.putl \sim_R term_x.putl$. Choose arbitrary $v \in Unit$ and choose c and y such that $\mathsf{fst}(k.putl((), c)) = y$. Clearly, $v = ()$, so we can compute:

$$k.putl(v, c) = k.putl((), c) = (y, c')$$
$$term_x.putl(v, y) = term_x.putl((), y) = (y, y)$$

Clearly, $y = y$, and law PUTL2 tells us that $k.putl((), c') = (y, c')$, and hence that $c'\ R\ y$.

Finally, we must show that $k.putr \sim_R term_x.putr$. Again, choose c and y such that $\mathsf{fst}(k.putl((), c)) = y$ and arbitrary $z \in X$.

$$k.putr(z, c) = ((), c')$$
$$term_x.putr(z, y) = ((), z)$$

It's clear that $() = ()$, and law PUTRL tells us that $k.putl((), c') = (z, c')$, and hence $c'\ R\ z$. □

2.3.20 Definition [Disconnect lens]:

$$\frac{x \in X \qquad y \in Y}{\mathit{disconnect}_{xy} \in X \leftrightarrow Y}$$

$$\mathit{disconnect}_{xy} \;=\; \mathit{term}_x ; \mathit{term}_y^{op}$$

The disconnect lens does not synchronize its two sides at all. The complement, *disconnect*.C, is $X \times Y$; inputs are squirreled away into one side of the complement, and outputs are retrieved from the other side of the complement.

(Note that we do not need an explicit proof that *disconnect* is a lens: this follows from the fact that *term* is a lens and $(-)^{op}$ and ; construct lenses from lenses.)

2.4 Products

A few additional notions from elementary category theory will be useful for giving us ideas about what sorts of properties to look for and for structuring the discussion of which of these properties hold and which fail for lenses.

The *categorical product* of two objects X and Y is an object $X \times Y$ and arrows $\pi_1 : X \times Y \to X$ and $\pi_2 : X \times Y \to Y$ such that for any two arrows $f : Z \to X$ and $g : Z \to Y$ there is a unique arrow $\langle f, g \rangle : Z \to X \times Y$—the *pairing* of f and g—satisfying $\langle f, g \rangle ; \pi_1 = f$ and $\langle f, g \rangle ; \pi_2 = g$. It is well known that, if a categorical product exists at all, it is unique up to isomorphism. If a category $\mathcal{C}$ has a product for each pair of objects, we say that $\mathcal{C}$ has products.

2.4.1 Theorem [No products]: LENS does not have products.

Proof idea: Suppose we have lenses $k \in Z \Longleftrightarrow X$ and $\ell \in Z \Longleftrightarrow Y$. Informally, the lens k includes a way to take any Z and choose a corresponding X and a way to take any X and find a corresponding Z. Many common categories with products include the former, but the latter is somewhat unique to lens categories, so we focus on the return trip here.

The lenses k and ℓ together mean we have a way to take any X and choose a corresponding Z, and we have a (separate) way to take any Y and choose a corresponding Z. Assume temporarily that the object part of the product of two objects is simply the Cartesian product. To complete the product, we must construct $\langle k, \ell \rangle \in Z \Longleftrightarrow X \times Y$, that is, we must find a way to take an X and a Y and choose a Z that corresponds to both simultaneously. But there may not be any such Z—the Z that k gives us from X may not be the same as the Z that ℓ gives us from Y.

To complete the proof, we simply choose X and Y carefully to rule out the possibility of a corresponding Z, regardless of whether we choose $X \times Y$ to be the Cartesian product or to be some other construction.

Proof: Uniqueness of pairing shows that there is exactly one lens from *Unit* to $\mathit{Unit} \times \mathit{Unit}$ (whatever this may be). Combined with Prop. 2.3.19 this shows that $\mathit{Unit} \times \mathit{Unit}$ is a one-element set. Again by Prop. 2.3.19 this then means that lenses between $\mathit{Unit} \times \mathit{Unit}$ and any other set X are constant which leads to cardinality clashes once $|X| > 1$.

In more detail: Assume, for a contradiction, that LENS does have products, and let W be the product of *Unit* with itself. The two projections are maps into *Unit*. By Proposition 2.3.19 there is exactly one lens from *Unit* to *Unit*. By uniqueness of pairing we can then conclude that there is exactly one map from *Unit* to W. Now for each $w \in W$ the lens $(\mathit{term}_w)^{op}$ is such a map, whence W must be a singleton set, and we can without loss of generality assume $W = \mathit{Unit}$. But now consider the pairing of term_0 and term_1 from $\{0, 1\}$ to *Unit*. Their pairing is a lens from $\{0, 1\}$ to $W = \mathit{Unit}$, hence itself of the form term_x for some $x \in \{0, 1\}$. But each of these violate the naturality laws. □

However, LENS *does* have a similar (but weaker) structure: a *tensor product*—i.e., an associative, two-argument functor. For any two objects X and Y, we have an object $X \otimes Y$, and for any two arrows $f : A \to X$ and $g : B \to Y$, an arrow $f \otimes g : A \otimes B \to X \otimes Y$ such that $(f_1; f_2) \otimes (g_1; g_2) = (f_1 \otimes g_1); (f_2 \otimes g_2)$ and $\mathit{id}_X \otimes \mathit{id}_Y = \mathit{id}_{X \otimes Y}$. Furthermore, for any three objects X, Y, Z there is a natural isomorphism $\alpha_{X,Y,Z} : (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z)$ satisfying certain coherence conditions (which specify that all ways of re-associating a quadruple are equal).

A categorical product is always a tensor product (by defining $f \otimes g = \langle \pi_1; f, \pi_2; g \rangle$), and conversely a tensor product is a categorical product if there are natural transformations $\pi_1, \pi_2, \mathit{diag}$

$$\begin{aligned} \pi_{1,X,Y} &\in X \otimes Y \to X \\ \pi_{2,X,Y} &\in X \otimes Y \to Y \\ \mathit{diag}_X &\in X \to X \otimes X \end{aligned}$$

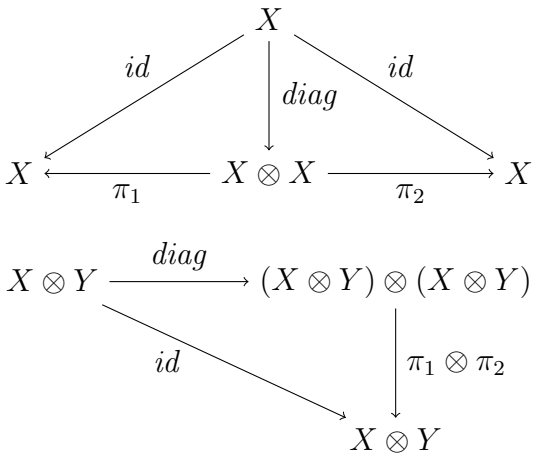
such that (suppressing subscripts to reduce clutter)

$$(f \otimes g); \pi_1 = \pi_1; f \tag{2.4.1}$$

$$(f \otimes g); \pi_2 = \pi_2; g \tag{2.4.2}$$

$$\mathit{diag}; (f \otimes f) = f; \mathit{diag} \tag{2.4.3}$$

for all arrows f and g. Moreover, the following diagrams must commute, in the sense that composite arrows with the same endpoints represent equal arrows:



The former diagram says that the result of applying *diag* is an element whose components are both equal to the original. The latter diagram says that the application of *diag* results in independent copies of the original.

Building a categorical product from a tensor product is not the most familiar presentation, but it can be shown to be equivalent (see Proposition 13 in [3], for example).

In the category LENS, we can build a tensor product and can also build projection lenses with reasonable behaviors. However, these projections are not quite natural transformations—laws 2.4.1 and 2.4.2 above hold only with an additional indexing constraint for particular f and g. More seriously, while it seems we can define some reasonable natural transformations with the type of *diag* (that is, arrows satisfying law 2.4.3), none of them make the additional diagrams commute.

2.4.2 Definition [Tensor product lens]:

$$\frac{k \in X \leftrightarrow Z \qquad \ell \in Y \leftrightarrow W}{k \otimes \ell \in X \times Y \leftrightarrow Z \times W}$$

$$\begin{array}{lcl}
C & = & k.C \times \ell.C \\
\mathit{missing} & = & (k.\mathit{missing}, \ell.\mathit{missing}) \\
\mathit{putr}((x,y),(c_k,c_\ell)) & = & \mathsf{let}\ (z,c'_k) = k.\mathit{putr}(x,c_k)\ \mathsf{in} \\
 & & \mathsf{let}\ (w,c'_\ell) = \ell.\mathit{putr}(y,c_\ell)\ \mathsf{in} \\
 & & ((z,w),(c'_k,c'_\ell)) \\
\mathit{putl}((z,w),(c_k,c_\ell)) & = & \mathsf{let}\ (x,c'_k) = k.\mathit{putl}(z,c_k)\ \mathsf{in} \\
 & & \mathsf{let}\ (y,c'_\ell) = \ell.\mathit{putl}(w,c_\ell)\ \mathsf{in} \\
 & & ((x,y),(c'_k,c'_\ell))
\end{array}$$

Proof of well-formedness: We will show that PUTRL holds; a similar argument shows that PUTLR holds. Suppose

$$k.putr(x, c_k) = (z, c'_k)$$
$$\ell.putr(y, c_\ell) = (w, c'_\ell)$$

so that:

$$(k \otimes \ell).putr((x, y), (c_k, c_\ell)) = ((z, w), (c'_k, c'_\ell))$$

Applying PUTRL to the lenses k and ℓ, we learn that

$$k.putl(z, c'_k) = (x, c'_k)$$
$$\ell.putl(w, c'_\ell) = (y, c'_\ell)$$

so that:

$$(k \otimes \ell).putl((z, w), (c'_k, c'_\ell)) = ((x, y), (c'_k, c'_\ell))$$

But this is exactly what we need to show for rule PUTRL. □

Proof of preservation of equivalence: If R_k is a witness that $k \equiv k'$ and R_ℓ is a witness that $\ell \equiv \ell'$, then $R = R_k \times R_\ell$ witnesses $k \otimes \ell \equiv k' \otimes \ell'$.

Since $k.missing\ R_k\ k'.missing$ and $\ell.missing\ R_\ell\ \ell'.missing$, we know that

$$(k.missing, \ell.missing)\ R\ (k'.missing, \ell'.missing),$$

that is:

$$(k \otimes \ell).missing\ R\ (k' \otimes \ell').missing$$

Choose arbitrary $(x, y) \in X \times Y$ and related complements $(c_k, c_\ell)\ R\ (c_{k'}, c_{\ell'})$. Define:

$$(z, c'_k) = k.putr(x, c_k)$$
$$(z', c'_{k'}) = k'.putr(x, c_{k'})$$
$$(w, c'_\ell) = \ell.putr(y, c_\ell)$$
$$(w', c'_{\ell'}) = \ell'.putr(y, c_{\ell'})$$

Since $c_k\ R_k\ c_{k'}$ and $k.putr \sim_{R_k} k'.putr$, we can conclude that $z = z'$ and $c'_k\ R_k\ c'_{k'}$. Similarly, $w = w'$ and $c'_\ell\ R_\ell\ c'_{\ell'}$. But we can compute

$$(k \otimes \ell).putr((x, y), (c_k, c_\ell)) = ((w, z), (c'_k, c'_\ell))$$
$$(k' \otimes \ell').putr((x, y), (c_{k'}, c_{\ell'})) = ((w', z'), (c'_{k'}, c'_{\ell'}))$$

where $(w, z) = (w', z')$ and $(c'_k, c'_\ell)\ R\ (c'_{k'}, c'_{\ell'})$. Thus, $(k \otimes \ell).putr \sim_R (k' \otimes \ell').putr$.

Showing that $(k \otimes \ell).putl \sim_R (k' \otimes \ell').putl$ is similar. □

2.4.3 Lemma [Functoriality of $\otimes$]: The tensor product operation on lenses induces a bifunctor on the category LENS, that is,

$id_X \otimes id_Y \equiv id_{X \times Y}$, and

$(k_1; \ell_1) \otimes (k_2; \ell_2) \equiv (k_1 \otimes k_2); (\ell_1 \otimes \ell_2)$.

Proof of functoriality: Corollary 2.3.9 implies the former equivalence. The latter has an intricate (but uninteresting) witness:

$$((c_{k_1}, c_{\ell_1}), (c_{k_2}, c_{\ell_2}))\ R\ ((c_{k_1}, c_{k_2}), (c_{\ell_1}, c_{\ell_2}))$$

That is, one state is related to another precisely when it is a rearrangement of the component states. It is clear that this relates the *missing* states of each lens, and the *putr* and *putl* components do identical computations (albeit in a different order), so they are related by $\sim_R$ as necessary. □

2.4.4 Lemma [Product bijection]: For bijections f and g,

$$bij_f \otimes bij_g \equiv bij_{f \times g}.$$

Proof: Write $k = bij_f \otimes bij_g$ and $\ell = bij_{f \times g}$. The total relation $R \in (Unit \times Unit) \times Unit$ is a witness. It's clear that $k.missing\ R\ \ell.missing$, so let's show that the puts are similar. Since all complements are related, this reduces to showing that equal input values yield equal output values.

$$\begin{array}{rcl} k.putr((x,y),((),())) & = & \mathsf{let}\ (x', c_1) = bij_f.putr(x,()) \ \mathsf{in} \\ & & \mathsf{let}\ (y', c_2) = bij_g.putr(y,()) \ \mathsf{in} \\ & & ((x', y'), (c_1, c_2)) \\ & = & ((f(x), g(y)), ((), ())) \\ \ell.putr((x,y),()) & = & ((f(x), g(y)), ()) \end{array}$$

The *putl* direction is similar. □

In fact, the particular tensor product defined above is very well behaved: it induces a *symmetric monoidal category*—i.e., a category with a unit object 1 and the following natural isomorphisms:

$$\begin{array}{rcl} \alpha_{X,Y,Z} & : & (X \otimes Y) \otimes Z \to X \otimes (Y \otimes Z) \\ \lambda_X & : & 1 \otimes X \to X \\ \rho_X & : & X \otimes 1 \to X \\ \gamma_{X,Y} & : & X \otimes Y \to Y \otimes X \end{array}$$

These are known as the *associator*, *left-unitor*, *right-unitor*, and *symmetry*, respectively. In addition to the equations implied by these being natural isomorphisms, they must also satisfy the coherence equations:

$$\begin{aligned} \alpha;\alpha &= (\alpha \otimes id);\alpha;(id \otimes \alpha) \\ \rho \otimes id &= \alpha;(id \otimes \lambda) \\ \alpha;\gamma;\alpha &= (\gamma \otimes id);\alpha;(id \otimes \gamma) \\ \alpha^{-1};\gamma;\alpha^{-1} &= (id \otimes \gamma);\alpha^{-1};(\gamma \otimes id) \\ \gamma;\gamma &= id \end{aligned}$$

2.4.5 Proposition [LENS,$\otimes$ is symmetric monoidal]: In the category SET, the Cartesian product is a bifunctor with *Unit* as unit, and gives rise to a symmetric monoidal category. Let $\alpha^\times, \lambda^\times, \rho^\times, \gamma^\times$ be associator, left-unitor, right-unitor, and symmetry natural isomorphisms. Then the $\otimes$ bifunctor also gives rise to a symmetric monoidal category of lenses, with *Unit* as unit and $\alpha^\otimes = bij \circ \alpha^\times$, $\lambda^\otimes = bij \circ \lambda^\times$, $\rho^\otimes = bij \circ \rho^\times$, and $\gamma^\otimes = bij \circ \gamma^\times$ as associator, left-unitor, right-unitor, and symmetry, respectively.

Knowing that LENS is a symmetric monoidal category is useful for several reasons. First, it tells us that, even though it is not quite a full-blown product, the tensor construction is algebraically quite well behaved. Second, it justifies a convenient intuition where lenses built from multiple tensors are pictured as graphical "wiring diagrams," and suggests a possible syntax for lenses that shuffle product components (which we briefly discuss in §5.1).

Proof: We know $\alpha^\otimes$, $\lambda^\otimes$, $\rho^\otimes$, and $\gamma^\otimes$ are all isomorphisms because every bijection lens is an isomorphism. Showing that they are natural is a straightforward calculation.[2] The five coherence conditions follow from coherence in SET, functoriality of *bij*, and Lemma 2.4.4. □

2.4.6 Definition [Projection lenses]: In LENS, the projection is parametrized by an extra element to return when executing a *putl* with a *missing* complement.

$$\frac{y \in Y}{\pi_{1y} \in X \times Y \leftrightarrow X}$$

$$\pi_{1y} = (id_X \otimes term_y);\rho^\otimes_X$$

[2]For example, showing that $\gamma^\otimes$ is natural requires showing that for any two lenses $k : X \leftrightarrow Z$ and $\ell : Y \leftrightarrow W$,

$$(k \otimes \ell);\gamma^\otimes_{Z,W} \equiv \gamma^\otimes_{X,Y};(\ell \otimes k).$$

The complements for these two lenses are $(k.C \times \ell.C) \times Unit$ and $Unit \times (\ell.C \times k.C)$; the isomorphism that simply rearranges the parts of the complement is a witness to the lenses' equivalence. The story is similar for the other naturality properties.

The other projection is defined similarly.

Returning to the example in the introduction, recall that we wish to create a lens $e : X \times Y \leftrightarrow Y \times Z$ with missing elements $m \in X$ and $M \in Z$. We now have the machinery necessary to construct this lens:

$$e = \pi_{2m} \,;\, \pi^{op}_{1M}$$

The extra parameter to the projection (e.g. m or M above) needs to be chosen with some care. Some sets may have clear neutral elements; for example, a projection from $A \times B^\star \to A$ will likely use the empty list $\langle\rangle$ as its neutral element. Other projections may need additional domain knowledge to choose a good neutral element—for example, a projection $A \times \text{Country} \to A$ might use the country with the most customers as its default.

In some cases, the algebraic laws that one wants the projection to satisfy may guide the choice as well. The extra parameter prevents full naturality from holding, and therefore prevents this from being a categorical product, but the following "indexed" version of the naturality law does hold.

2.4.7 Lemma [Naturality of projections]: Suppose $k \in X_k \leftrightarrow Y_k$ and $\ell \in X_\ell \leftrightarrow Y_\ell$ and choose some initial value $y_i \in Y_\ell$. Define $(x_i, c_i) = \ell.putl(y_i, \ell.missing)$. Then $(k \otimes \ell); \pi_{1y_i} \equiv \pi_{1x_i}; k$.

Proof: We show that the following diagram commutes:

$$\begin{array}{ccc}
X_k \times X_\ell & \xrightarrow{k \otimes \ell} & Y_k \times Y_\ell \\
\Big\downarrow{\scriptstyle id_{X_k} \otimes term_{x_i}} & & \Big\downarrow{\scriptstyle id_{Y_k} \otimes term_{y_i}} \\
X_k \times Unit & \xrightarrow{k \otimes id_{Unit}} & Y_k \times Unit \\
\Big\downarrow{\scriptstyle \rho_{X_k}} & & \Big\downarrow{\scriptstyle \rho_{Y_k}} \\
X_k & \xrightarrow{\qquad k \qquad} & Y_k
\end{array}$$

To show that the top square commutes, we invoke functoriality of $\otimes$ and the property of identities; all that remains is to show that

$$\ell; term_{y_i} \equiv term_{x_i}$$

which follows from the uniqueness of terminal lenses and the definition of x_i. The bottom square commutes because ρ is a natural isomorphism. □

The most serious problem, though, is that there is no diagonal. There are, of course, lenses with the *type* we need for *diag*—for example, *disconnect*. Or, more usefully, the lens that coalesces the copies of X whenever possible, preferring the left one when it cannot coalesce (this is essentially the *merge* lens from [16])

$$diag \in X \to X \times X$$

$$\begin{array}{lcl} C & = & Unit + X \\ missing & = & \mathsf{inl}\ () \\ putr(x, \mathsf{inl}\ ()) & = & ((x,x), \mathsf{inl}\ ()) \\ putr(x, \mathsf{inr}\ x') & = & ((x,x'), eq(x,x')) \\ putl((x,x'),c) & = & (x, eq(x,x')) \end{array}$$

where here the *eq* function tests its arguments for equality:

$$eq(x,x') = \begin{cases} \mathsf{inl}\ () & x = x' \\ \mathsf{inr}\ x' & x \neq x' \end{cases}$$

—$eq(x,x')$ yields $\mathsf{inl}\ ()$ if $x = x'$ and yields x' if not. This assumes that X possesses a decidable equality, a reasonable assumption for the applications of lenses that we know about. However, neither of these proposals satisfy all the required laws.

Proof of well-formedness:
PutLR:

$$\begin{array}{rcll} putr(putl((x,x'),c)) & = & putr(x, eq(x,x')) & \\ & = & \begin{cases} putr(x, \mathsf{inl}\ ()) & x = x' \\ putr(x, \mathsf{inr}\ x') & x \neq x' \end{cases} & \\ & = & \begin{cases} ((x,x), \mathsf{inl}\ ()) & x = x' \\ ((x,x'), \mathsf{inr}\ x') & x \neq x' \end{cases} & \\ & = & ((x,x'), eq(x,x')) & \end{array}$$

PutRL:

$$\begin{array}{rcll} putl(putr(x, \mathsf{inl}\ ())) & = & putl((x,x), \mathsf{inl}\ ()) & \\ & = & (x, \mathsf{inl}\ ()) & \\ putl(putr(x, \mathsf{inr}\ x')) & = & putl((x,x'), eq(x,x')) & \\ & = & (x, eq(x,x')) & \square \end{array}$$

2.5 Sums and Lists

Historically, the status of sums has been even more mysterious than that of products. In particular, the *injection arrows* from A to $A+B$ and B to $A+B$ do not even make sense in the asymmetric setting; as functions, they are not surjective, so they cannot satisfy PutGet.

Before we study the question for LENS, let us formally define a sum. A *categorical sum* of two objects X and Y is an object $X + Y$ and arrows $inl : X \to X + Y$ and $inr : Y \to X + Y$ such that for any two arrows $f : X \to Z$ and $g : Y \to Z$ there is a unique arrow $[f, g] : X + Y \to Z$—the *choice* of f or g—satisfying $inl; [f, g] = f$ and $inr; [f, g] = g$. As with products, if a sum exists, it is unique up to isomorphism.

Since products and sums are dual, Corollary 2.3.17 and Theorem 2.4.1 imply that LENS does not have sums. But we do have a tensor whose object part is a set-theoretic sum—in fact, there are at least two interestingly different ones—and we can define useful associated structures, including a choice operation on lenses. But these constructions are even farther away from being categorical sums than what we saw with products.

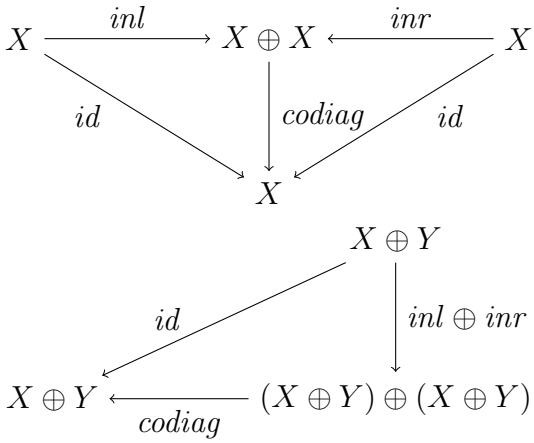
As with products, a tensor can be extended to a sum by providing three natural transformations—this time written *inl*, *inr*, and *codiag*; that is, for each pair of objects X and Y, there must be arrows

$$\begin{aligned} inl_{X,Y} &\in X \to X \oplus Y \\ inr_{X,Y} &\in Y \to X \oplus Y \\ codiag_X &\in X \oplus X \to X \end{aligned}$$

such that

$$\begin{aligned} inl; (f \oplus g) &= f; inl \\ inr; (f \oplus g) &= g; inr \\ (f \oplus f); codiag &= codiag; f \end{aligned}$$

and making the following diagrams commute:



These diagrams are identical to the product diagrams, with the exception that the arrows point in the opposite directions (that is, the sum diagrams are the dual of the product diagrams).

The two tensors, which we called *retentive* and *forgetful* in §2.1, differ in how they handle the complement when the new value being *put* is from a different branch of the sum than the old value that was *put*. The retentive sum keeps complements for *both*

sublenses in its own complement and switches between them as needed. The forgetful sum keeps only one complement, corresponding to whichever branch was last *put*. If the next *put* switches sides, the complement is replaced with *missing*.

2.5.1 Definition [Retentive tensor sum lens]:

$$\frac{k \in X \leftrightarrow Z \qquad \ell \in Y \leftrightarrow W}{k \oplus \ell \in X + Y \leftrightarrow Z + W}$$

$$\begin{array}{lcl}
C & = & k.C \times \ell.C \\
\mathit{missing} & = & (k.\mathit{missing}, \ell.\mathit{missing}) \\
\mathit{putr}(\mathsf{inl}\ x, (c_k, c_\ell)) & = & \mathsf{let}\ (z, c'_k) = k.\mathit{putr}(x, c_k)\ \mathsf{in} \\
 & & (\mathsf{inl}\ z, (c'_k, c_\ell)) \\
\mathit{putr}(\mathsf{inr}\ y, (c_k, c_\ell)) & = & \mathsf{let}\ (w, c'_\ell) = \ell.\mathit{putr}(y, c_\ell)\ \mathsf{in} \\
 & & (\mathsf{inr}\ w, (c_k, c'_\ell)) \\
\mathit{putl}(\mathsf{inl}\ z, (c_k, c_\ell)) & = & \mathsf{let}\ (x, c'_k) = k.\mathit{putl}(z, c_k)\ \mathsf{in} \\
 & & (\mathsf{inl}\ x, (c'_k, c_\ell)) \\
\mathit{putl}(\mathsf{inr}\ w, (c_k, c_\ell)) & = & \mathsf{let}\ (y, c'_\ell) = \ell.\mathit{putl}(y, c_\ell)\ \mathsf{in} \\
 & & (\mathsf{inr}\ y, (c_k, c'_\ell))
\end{array}$$

Proof of well-formedness: We show that PutRL holds; the proof that PutLR holds is similar. Choose arbitrary $c_k \in k.C$ and $c_\ell \in \ell.C$. There are two cases to consider for the starting value: it will be either $\mathsf{inl}\ x$ for some $x \in X$ or $\mathsf{inr}\ y$ for some $y \in Y$. In the former case, define $(z, c'_k) = k.\mathit{putr}(x, c_k)$ so that applying PutRL to k tells us that $k.\mathit{putl}(z, c'_k) = (x, c'_k)$. But now we can compute:

$$\mathit{putl}(\mathit{putr}(\mathsf{inl}\ x, (c_k, c_\ell))) = \mathit{putl}(\mathsf{inl}\ z, (c'_k, c_\ell)) = (\mathsf{inl}\ x, (c'_k, c_\ell)).$$

Thus, the value has round-tripped exactly as $\mathsf{inl}\ x$, and the complement changed only after the *putr* (and not after the *putl*) – exactly what we needed to show.

The other case is similar: define $(w, c'_\ell) = \ell.\mathit{putr}(y, c_\ell)$ so that applying PutRL to ℓ tells us that $\ell.\mathit{putl}(w, c'_\ell) = (y, c'_\ell)$. Computation then shows that:

$$\mathit{putl}(\mathit{putr}(\mathsf{inr}\ y, (c_k, c_\ell))) = \mathit{putl}(\mathsf{inr}\ w, (c_k, c'_\ell)) = (\mathsf{inr}\ y, (c_k, c'_\ell)).\ \square$$

Proof of preservation of equivalence: Suppose $k \equiv k'$ and $\ell \equiv \ell'$, as witnessed by relations R_k and R_ℓ, respectively. Then $R = R_k \times R_\ell$ witnesses the equivalence $k \oplus \ell \equiv k' \oplus \ell'$. Since $k.\mathit{missing}\ R_k\ k'.\mathit{missing}$ and $\ell.\mathit{missing}\ R_\ell\ \ell'.\mathit{missing}$, we have $(k \oplus \ell).\mathit{missing}\ R\ (k' \oplus \ell').\mathit{missing}$.

We now show that $(k \oplus \ell).\mathit{putr} \sim_R (k' \oplus \ell').\mathit{putr}$. Choose arbitrary $v \in X + Y, c_k \in k.C, c_{k'} \in k'.C, c_\ell \in \ell.C, c_{\ell'} \in \ell'.C$ such that $(c_k, c_\ell)\ R\ (c_{k'}, c_{\ell'})$. By the definition of

R, we can conclude that $c_k\ R_k\ c_{k'}$ and that $c_\ell\ R_\ell\ c_{\ell'}$. There are two cases to consider: either $v = \mathsf{inl}\ x$ for some $x \in X$ or $v = \mathsf{inr}\ y$ for some $y \in Y$. In the first case, define

$$(z, c'_k) = k.putr(x, c_k)$$
$$(z', c'_{k'}) = k'.putr(x, c_{k'})$$

Since $c_k\ R_k\ c_{k'}$, we can conclude $z = z'$ and $c'_k\ R_k\ c'_{k'}$. Therefore,

$$(k \oplus \ell).putr(v, (c_k, c_\ell)) = (\mathsf{inl}\ z, (c'_k, c_\ell))$$
$$(k' \oplus \ell').putr(v, (c_{k'}, c_{\ell'})) = (\mathsf{inl}\ z, (c'_{k'}, c_{\ell'}))$$

where $(c'_k, c_\ell)\ R\ (c'_{k'}, c_{\ell'})$ as desired. The second case, where $v = \mathsf{inr}\ y$, is similar.

Showing that $(k \oplus \ell).putl \sim_R (k' \oplus \ell').putl$ is symmetric to the argument for $putr$. □

2.5.2 Lemma [Functoriality of $\oplus$]: The tensor sum operation on lenses induces a bifunctor on LENS.

Proof of functoriality: Corollary 2.3.9 gives us $id_X \oplus id_Y \equiv id_{X+Y}$ with fairly minor computation. We must also show that composition is preserved. Suppose we have four lenses:

$$k \in X \leftrightarrow Y \qquad k' \in X' \leftrightarrow Y'$$
$$\ell \in Y \leftrightarrow Z \qquad \ell' \in Y' \leftrightarrow Z'$$

The obvious isomorphism between complements witnesses the equivalence $(k; \ell) \oplus (k'; \ell') \equiv (k \oplus k'); (\ell \oplus \ell')$, namely:

$$((c_k, c_\ell), (c'_k, c'_\ell))\ R\ ((c_k, c'_k), (c_\ell, c'_\ell))$$

Define abbreviations $a = (k; \ell) \oplus (k'; \ell')$ and $b = (k \oplus k'); (\ell \oplus \ell')$. Expanding definitions,

$$a.missing = ((k.missing, \ell.missing), (k'.missing, \ell'.missing))$$
$$b.missing = ((k.missing, k'.missing), (\ell.missing, \ell'.missing))$$

so $a.missing\ R\ b.missing$. We must also show $a.putr \sim_R b.putr$ and $a.putl \sim_R b.putl$. We will show only the former; the proof of the latter is similar.

Choose arbitrary $v \in X + X', c_a \in a.C, c_b \in b.C$ such that $c_a\ R\ c_b$. This means there are $c_k \in k.C, c_{k'} \in k'.C, c_\ell \in \ell.C, c_{\ell'} \in \ell'.C$ such that $c_a = ((c_k, c_\ell), (c_{k'}, c_{\ell'}))$ and $c_b = ((c_k, c_{k'}), (c_\ell, c_{\ell'}))$. There are two cases to consider: either $v = \mathsf{inl}\ x$ or $v = \mathsf{inr}\ x'$. In the first case, we can define

$$(y, c'_k) = k.putr(x, c_k)$$
$$(z, c'_\ell) = \ell.putr(y, c_\ell),$$

and compute:

$$a.putr(\mathsf{inl}\ x, ((c_k, c_\ell), (c_{k'}, c_{\ell'}))) = (\mathsf{inl}\ z, ((c'_k, c'_\ell), (c_{k'}, c_{\ell'})))$$
$$b.putr(\mathsf{inl}\ x, ((c_k, c_{k'}), (c_\ell, c_{\ell'}))) = (\mathsf{inl}\ z, ((c'_k, c_{k'}), (c'_\ell, c_{\ell'})))$$

Since $\mathsf{inl}\ z = \mathsf{inl}\ z$ and $((c'_k, c'_\ell), (c_{k'}, c_{\ell'}))\ R\ ((c'_k, c_{k'}), (c'_\ell, c_{\ell'}))$, we have finished the first case. The second case, where $v = \mathsf{inr}\ x'$, is nearly identical, and we conclude that $a.putr \sim_R b.putr$. □

2.5.3 Definition [Forgetful tensor sum]:

$$\frac{k \in X \leftrightarrow Z \qquad \ell \in Y \leftrightarrow W}{k \oplus^f \ell \in X + Y \leftrightarrow Z + W}$$

$$\begin{array}{lcl}
C & = & k.C + \ell.C \\
missing & = & \mathsf{inl}\ k.missing \\
putr(\mathsf{inl}\ x, \mathsf{inl}\ c_k) & = & \mathsf{let}\ (z, c'_k) = k.putr(x, c_k)\ \mathsf{in} \\
 & & (\mathsf{inl}\ z, \mathsf{inl}\ c'_k) \\
putr(\mathsf{inl}\ x, \mathsf{inr}\ c_\ell) & = & \mathsf{let}\ (z, c_k) = k.putr(x, k.missing)\ \mathsf{in} \\
 & & (\mathsf{inl}\ z, \mathsf{inl}\ c_k) \\
putr(\mathsf{inr}\ y, \mathsf{inl}\ c_k) & = & \mathsf{let}\ (w, c_\ell) = \ell.putr(y, \ell.missing)\ \mathsf{in} \\
 & & (\mathsf{inr}\ w, \mathsf{inr}\ c_\ell) \\
putr(\mathsf{inr}\ y, \mathsf{inr}\ c_\ell) & = & \mathsf{let}\ (w, c'_\ell) = k.putr(y, c_\ell)\ \mathsf{in} \\
 & & (\mathsf{inr}\ w, \mathsf{inr}\ c'_\ell)
\end{array}$$

putl is similar

Proof of well-formedness: As for the retentive sum, the round-trip laws for k and ℓ guarantee that $k \oplus^f \ell$ round-trips. The only difference is that there are additional cases to consider when the tag on the value and the tag on the complement do not match at the beginning of the trip; however, this poses no real difficulty, as the tags *will* match after the first put. □

Proof of preservation of equivalence: Let $a = k \oplus^f \ell$ and $b = k' \oplus^f \ell'$. If R_k witnesses $k \equiv k'$ and R_ℓ witnesses $\ell \equiv \ell'$ then $a \equiv b$ may be witnessed by

$$R = \{(\mathsf{inl}\ c, \mathsf{inl}\ c') \mid c\ R_k\ c'\} \cup \{(\mathsf{inr}\ c, \mathsf{inr}\ c') \mid c\ R_\ell\ c'\}$$

Since $k.missing\ R_k\ k'.missing$, we know $a.missing\ R\ b.missing$.

We must still show that $a.putr \sim_R b.putr$ and that $a.putl \sim_R b.putl$; for each of these proofs, there are cases to consider where the input is tagged inl and cases

where the input is tagged inr. Below, we will consider only the inl case for *putr*; the remaining cases are similar.

Therefore, consider arbitrary $x \in X, c_a \in a.C, c_b \in b.C$ such that $c_a \; R \; c_b$. Project these complements into $k.C$ and $k'.C$, respectively, as follows:

$$c'_a = \begin{cases} c_k & c_a = \mathsf{inl}\ c_k \\ k.missing & c_a = \mathsf{inr}\ c_\ell \end{cases}$$
$$c'_b = \begin{cases} c_{k'} & c_b = \mathsf{inl}\ c_{k'} \\ k'.missing & c_b = \mathsf{inr}\ c_{\ell'} \end{cases}$$

Since $c_a \; R \; c_b$, we know they have the same tag, and hence that c'_a and c'_b follow the same "branch" in their definition; in either branch, we find that $c'_a \; R_k \; c'_b$, because $c_a \; R \; c_b$ and $k.missing \; R_k \; k'.missing$. But now we can compute:

$$a.putr(x, c_a) = \mathsf{let}\ (z, c'_k) = k.putr(x, c'_a)\ \mathsf{in}\ (\mathsf{inl}\ z, \mathsf{inl}\ c'_k)$$
$$b.putr(x, c_b) = \mathsf{let}\ (z, c'_{k'}) = k'.putr(x, c'_b)\ \mathsf{in}\ (\mathsf{inl}\ z, \mathsf{inl}\ c'_{k'})$$

The desired properties now arise because $k.putr \sim_{R_k} k'.putr$ and $c'_a \; R_k \; c'_b$. □

Proof of functoriality: There are two things to show:

$$id_X \oplus^f id_Y \equiv id_{X+Y}$$

$$(k \oplus^f k'); (\ell \oplus^f \ell') \equiv (k; \ell) \oplus^f (k'; \ell')$$

For identity preservation, we use the total relation:

$$c \; R \; ()$$

Clearly the initial condition $(id \oplus^f id).missing \; R \; id.missing$ holds; we will also show that $(id \oplus^f id).putr \sim_R id.putr$, eliding the similar proof relating the *putl* functions. So, choose arbitrary $v \in X + Y$ and $c \in Unit + Unit$.

$$(id \oplus^f id).putr(v, c) = \begin{cases} \mathsf{let}\ (x', c') = id.putr(x, ()) & \\ \mathsf{in}\ (\mathsf{inl}\ x', \mathsf{inl}\ c') & v = \mathsf{inl}\ x \\ \mathsf{let}\ (y', c') = id.putr(y, ()) & \\ \mathsf{in}\ (\mathsf{inr}\ y', \mathsf{inr}\ c') & v = \mathsf{inr}\ y \end{cases}$$
$$= \begin{cases} (\mathsf{inl}\ x, \mathsf{inl}\ ()) & v = \mathsf{inl}\ x \\ (\mathsf{inr}\ y, \mathsf{inr}\ ()) & v = \mathsf{inr}\ y \end{cases}$$
$$= \left(v, \begin{cases} \mathsf{inl}\ () & v = \mathsf{inl}\ x \\ \mathsf{inr}\ () & v = \mathsf{inr}\ y \end{cases}\right)$$
$$id.putr(v, c) = (v, ())$$

Since $v = v$ and the complements are always related, this shows that

$$(id \oplus^f id).putr \sim_R id.putr.$$

For preservation of composition, we use the relation R defined by:

$$\{((\mathsf{inl}\ c_k, \mathsf{inl}\ c_\ell), \mathsf{inl}\ (c_k, c_\ell)) \mid c_k \in k.C, c_\ell \in \ell.C\} \cup$$
$$\{((\mathsf{inr}\ c_k, \mathsf{inr}\ c_\ell), \mathsf{inr}\ (c_k, c_\ell)) \mid c_k \in k'.C, c_\ell \in \ell'.C\}$$

Abbreviating $a = (k \oplus^f k'); (\ell \oplus^f \ell')$ and $b = (k; \ell) \oplus^f (k'; \ell')$, we can quickly see that $a.missing = (\mathsf{inl}\ k.missing, \mathsf{inl}\ \ell.missing)\ R\ \mathsf{inl}\ (k.missing, \ell.missing) = b.missing$. We will also show that $a.putr \sim_R b.putr$, eliding the similar proof that $a.putl \sim_R b.putl$.

Choose arbitrary $v \in X_0 + X_1, c_a \in a.C, c_b \in b.C$ such that $c_a\ R\ c_b$. There are many cases to consider, but two of them are representative of the remainder. In the first representative case, we have

$$\begin{aligned} v &= \mathsf{inl}\ x_0 \\ c_a &= (\mathsf{inl}\ c_k, \mathsf{inl}\ c_\ell) \\ c_b &= \mathsf{inl}\ (c_k, c_\ell) \end{aligned}$$

Then:

$$\begin{aligned} a.putr(v, c_a) &= \mathsf{let}\ (y_0, c'_k) = k.putr(x_0, c_k)\ \mathsf{in} \\ &\quad \mathsf{let}\ (z_0, c'_\ell) = \ell.putr(y_0, c_\ell)\ \mathsf{in} \\ &\quad (\mathsf{inl}\ z_0, (\mathsf{inl}\ c'_k, \mathsf{inl}\ c'_\ell)) \\ b.putr(v, c_b) &= \mathsf{let}\ (z_0, (c'_k, c'_\ell)) = (k; \ell).putr(x_0, (c_k, c_\ell))\ \mathsf{in} \\ &\quad (\mathsf{inl}\ z_0, \mathsf{inl}\ (c'_k, c'_\ell)) \\ &= \mathsf{let}\ (y_0, c'_k) = k.putr(x_0, c_k)\ \mathsf{in} \\ &\quad \mathsf{let}\ (z_0, c'_\ell) = \ell.putr(y_0, c_\ell)\ \mathsf{in} \\ &\quad (\mathsf{inl}\ z_0, \mathsf{inl}\ (c'_k, c'_\ell)) \end{aligned}$$

Since z_0, c'_k, c'_ℓ are computed identically in the two equations, the relation is preserved in this case.

In the second representative case, we have

$$\begin{aligned} v &= \mathsf{inl}\ x_0 \\ c_a &= (\mathsf{inr}\ c_{k'}, \mathsf{inr}\ c_{\ell'}) \\ c_b &= \mathsf{inr}\ (c_{k'}, c_{\ell'}) \end{aligned}$$

Then:

$$\begin{aligned}
a.putr(v, c_a) &= \mathsf{let}\ (y_0, c_k) = k.putr(x_0, k.missing)\ \mathsf{in} \\
&\quad\ \mathsf{let}\ (z_0, c_\ell) = \ell.putr(x_0, \ell.missing)\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z_0, (\mathsf{inl}\ c_k, \mathsf{inl}\ c_\ell)) \\
b.putr(v, c_b) &= \mathsf{let}\ (z_0, c') = (k;\ell).putr(x_0, (k;\ell).missing)\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z_0, \mathsf{inl}\ c') \\
&= \mathsf{let}\ (y_0, c_k) = k.putr(x_0, k.missing)\ \mathsf{in} \\
&\quad\ \mathsf{let}\ (z_0, c_\ell) = \ell.putr(y_0, \ell.missing)\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z_0, \mathsf{inl}\ (c_k, c_\ell))
\end{aligned}$$

Again, since z_0, c_k, c_ℓ are computed identically in both equations, the relation is preserved. □

2.5.4 Lemma [Sum bijection]: For bijections f and g,

$$bij_f \oplus bij_g \equiv bij_f \oplus^f bij_g \equiv bij_{f+g}$$

Proof: Write $k = bij_f \oplus bij_g$, $k^f = bij_f \oplus^f bij_g$, and $\ell = bij_{f+g}$. The total relation $R \subset (Unit \times Unit) \times Unit$ is a witness that $k \equiv \ell$ and the total relation $R^f \subset (Unit + Unit) \times Unit$ is a witness that $k^f \equiv \ell$. It's clear that $k.missing\ R\ \ell.missing$ and $k^f.missing\ R^f\ \ell.missing$, so let's show that the puts are similar. Since all complements are related, this reduces to showing that equal input values yield equal output values.

$$\begin{aligned}
k.putr(\mathsf{inl}\ x, ((), ())) &= \mathsf{let}\ (z, c_k) = bij_f.putr(x, ())\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z, (c_k, ())) \\
&= \mathsf{let}\ (z, c_k) = (f(x), ())\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z, (c_k, ())) \\
&= (\mathsf{inl}\ f(x), ((), ())) \\
k.putr(\mathsf{inr}\ y, ((), ())) &= (\mathsf{inr}\ g(y), ((), ())) \\
k^f.putr(\mathsf{inl}\ x, c) &= \mathsf{let}\ (z, c_k) = bij_f.putr(x, ())\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z, \mathsf{inl}\ c_k) \\
&= \mathsf{let}\ (z, c_k) = (f(x), ())\ \mathsf{in} \\
&\quad\ (\mathsf{inl}\ z, \mathsf{inl}\ c_k) \\
&= (\mathsf{inl}\ f(x), \mathsf{inl}\ ()) \\
k^f.putr(\mathsf{inr}\ y, c) &= (\mathsf{inr}\ g(y), \mathsf{inr}\ ()) \\
\ell.putr(\mathsf{inl}\ x, ()) &= ((f+g)(\mathsf{inl}\ x), ()) \\
&= (\mathsf{inl}\ f(x), ()) \\
\ell.putr(\mathsf{inr}\ y, ()) &= (\mathsf{inr}\ g(y), ())
\end{aligned}$$

The *putl* direction is similar. □

2.5.5 Proposition [LENS,$\oplus$, $\oplus^f$ are symmetric monoidal]: In SET, the disjoint union gives rise to a symmetric monoidal category with $\emptyset$ as unit. Let $\alpha^+, \lambda^+, \rho^+, \gamma^+$ be associator, left-unitor, right-unitor, and symmetry natural isomorphisms. Then the $\oplus$ and $\oplus^f$ bifunctors each give rise to a symmetric monoidal category of lenses with $\emptyset$ as unit and $\alpha^\oplus = bij \circ \alpha^+$, $\lambda^\oplus = bij \circ \lambda^+$, $\rho^\oplus = bij \circ \rho^+$, and $\gamma^\oplus = bij \circ \gamma^+$ as associator, left-unitor, right-unitor, and symmetry, respectively.

The types of these natural isomorphisms are:

$$\begin{array}{rcl} \alpha^\oplus_{X,Y,Z} & \in & (X+Y)+Z \leftrightarrow X+(Y+Z) \\ \lambda^\oplus_X & \in & \emptyset + X \leftrightarrow X \\ \rho^\oplus_X & \in & X + \emptyset \leftrightarrow X \\ \gamma^\oplus_{X,Y} & \in & X+Y \leftrightarrow Y+X \end{array}$$

Proof: We know $\alpha^\oplus$, $\lambda^\oplus$, $\rho^\oplus$, and $\gamma^\oplus$ are all isomorphisms because every bijection lens is an isomorphism. Showing that they are natural is a straightforward calculation. The only subtlety comes in showing that $(k \oplus^f \ell); \gamma^\oplus \equiv \gamma^\oplus; (\ell \oplus^f k)$. We must be careful to include the *missing* complements in the relation; the following relation will do:

$$\begin{array}{rcl} R & = & \{(\mathsf{inl}\ c, \mathsf{inr}\ c) \mid c \in k.C\} \cup \\ & & \{(\mathsf{inr}\ c, \mathsf{inl}\ c) \mid c \in \ell.C\} \cup \\ & & \{(\mathsf{inl}\ k.\mathit{missing}, \mathsf{inl}\ \ell.\mathit{missing})\} \end{array}$$

The five coherence conditions follow from coherence in SET, functoriality of *bij*, and Lemma 2.5.4. □

Unlike the product unit, there are no interesting lenses whose domain is the sum's unit, so this cannot be used to define the injection lenses; we have to do it by hand.

2.5.6 Definition [Injection lenses]:

$$\frac{x \in X}{\mathit{inl}_x \in X \leftrightarrow X+Y}$$

$$\begin{array}{lcl} C & = & X \times (\mathit{Unit} + Y) \\ \mathit{missing} & = & (x, \mathsf{inl}\ ()) \\ \mathit{putr}(x, (x', \mathsf{inl}\ ())) & = & (\mathsf{inl}\ x, (x, \mathsf{inl}\ ())) \\ \mathit{putr}(x, (x', \mathsf{inr}\ y)) & = & (\mathsf{inr}\ y, (x, \mathsf{inr}\ y)) \\ \mathit{putl}(\mathsf{inl}\ x, c) & = & (x, (x, \mathsf{inl}\ ())) \\ \mathit{putl}(\mathsf{inr}\ y, (x, c)) & = & (x, (x, \mathsf{inr}\ y)) \end{array}$$

We also define $inr_y = inl_y; \gamma^{\oplus}_{Y,X}$.

Proof of well-formedness: For PUTRL, we consider two cases: either the complement has the form $(x_c, \mathsf{inl}\ ())$ or the form $(x_c, \mathsf{inr}\ y)$.

$$\begin{aligned} putl(putr(x, (x_c, \mathsf{inl}\ ()))) &= putl(\mathsf{inl}\ x, (x, \mathsf{inl}\ ())) \\ &= (x, (x, \mathsf{inl}\ ())) \\ putl(putr(x, (x_c, \mathsf{inr}\ y))) &= putl(\mathsf{inr}\ y, (x, \mathsf{inr}\ y)) \\ &= (x, (x, \mathsf{inr}\ y)) \end{aligned}$$

Thus, in each case, the output value is equal to the input value and the complement is unaffected by the *putl*, as required by PUTRL.

To show PUTLR holds, we again consider two cases: either we start with $\mathsf{inl}\ x$ or $\mathsf{inr}\ y$.

$$\begin{aligned} putr(putl(\mathsf{inl}\ x, (x_c, y_c))) &= putr(x, (x, \mathsf{inl}\ ())) \\ &= (\mathsf{inl}\ x, (x, \mathsf{inl}\ ())) \\ putr(putl(\mathsf{inr}\ y, (x_c, y_c))) &= putr(x_c, (x_c, \mathsf{inr}\ y)) \\ &= (\mathsf{inr}\ y, (x_c, \mathsf{inr}\ y)) \end{aligned}$$

In both cases, the value output matches the value input and the complement remains unaffected by *putr*. □

As with the projection lenses for tensor products, we may ask whether the injection lenses for tensor sums are natural. If they were, we would expect a diagram like the following one to commute for all f:

$$\begin{array}{ccc} X & \xrightarrow{\quad f \quad} & Y \\ {\scriptstyle inl_x}\downarrow & & \downarrow{\scriptstyle inl_y} \\ X + Z & \xrightarrow{\quad f \oplus id \quad} & Y + Z \end{array}$$

Now, even if we carefully choose x and y to be related by f as we did for the projection lenses, this diagram may not commute. When running the *putr* function, the path along the top always invokes *f.putr*, whereas the path along the bottom may sometimes invoke *id.putr* instead; at that moment, the complements of f (on the top path) and $f \oplus id$ (on the bottom path) get out of synch. As we show in the following proposition this can be used to produce a subsequent observable difference, i.e., not only at the level of complements.

The situation with the forgetful sum is similar, but offers an additional way to desynchronize the two complements: when resetting f's complement along the bottom path to *missing*.

2.5.7 Proposition: The injection lenses are not natural.

Proof: We first define a lens that counts the number of changes it sees in the *putr* direction, and allows puts of non-numbers to be overridden in the *putl* direction:

$$\frac{x \in X}{count_x \in X \leftrightarrow Unit + \mathbb{N}}$$

$$\begin{array}{lcl}
C & = & X \times Bool \times \mathbb{N} \\
missing & = & (x, \mathsf{true}, 0) \\
putr(x, (x', b, n)) & = & \\
\left\{ \begin{array}{ll}
(\mathsf{inl}\ (), (x, b, n)) & x = x' \wedge \neg b \\
(\mathsf{inr}\ n, (x, b, n)) & x = x' \wedge b \\
(\mathsf{inr}\ (n+1), (x, \mathsf{true}, n+1)) & x \neq x'
\end{array} \right. & & \\
putl(\mathsf{inl}\ (), (x, b, n)) & = & (x, (x, \mathsf{false}, n)) \\
putl(\mathsf{inr}\ n, (x, b, n')) & = & (x, (x, \mathsf{true}, n))
\end{array}$$

We delay the proof that this lens is well-formed temporarily. Contrast the lens $inl_b; (count_{b'} \oplus id_{Unit})$ with $count_{b'}; inl_n$ (where b and b' are arbitrary $Bool$ values and n is an arbitrary $Unit + \mathbb{N}$ value). Consider the put objects

$$\langle \mathsf{inl}\ \mathsf{true}, \mathsf{inr}\ (\mathsf{inr}\ ()), \mathsf{inl}\ \mathsf{false}, \mathsf{inr}\ (\mathsf{inl}\ (\mathsf{inl}\ ())), \mathsf{inl}\ \mathsf{true}, \mathsf{inl}\ \mathsf{false} \rangle$$

The first two put objects in the list are simply initializing the lens: we first put true to the right, getting an inl object out on the right from both lenses, then put back an inr object, switching sides.

The next put of false to the right is where the problem really arises. For the $count_{b'}; inl_n$ lens, the counting lens first registers the change from true to false, then its output gets thrown away. On the other hand, in the $inl_b; (count_{b'} \oplus id_{Unit})$ lens, the false gets thrown away before the counting lens can see it, so the complement in the counting lens doesn't get updated.

The remainder of the objects simply manifest this problem by switching the sum back to the counting side, and getting an output from the counting lenses; one will give a higher count than the other.

The proof for *inr* is symmetric. □

Proof of well-formedness: For completeness, we must also show that $count_x$ satisfies the lens laws.

PUTLR: There are two cases to consider. Both are simple calculations.

$$\begin{aligned} putr(putl(\mathsf{inl}\ (), (x, b, n'))) &= putr(x, (x, \mathsf{false}, n')) \\ &= (\mathsf{inl}\ (), (x, \mathsf{false}, n')) \\ putr(putl(\mathsf{inr}\ n, (x, b, n'))) &= putr(x, (x, \mathsf{true}, n)) \\ &= (\mathsf{inr}\ n, (x, \mathsf{true}, n)) \end{aligned}$$

PUTRL: There are three cases to consider. For the first case, choose distinct $x \neq x'$.

$$\begin{aligned} putl(putr(x, (x', b, n))) &= putl(\mathsf{inr}\ (n+1), (x, \mathsf{true}, n+1)) \\ &= (x, (x, \mathsf{true}, n+1)) \end{aligned}$$

In the remaining cases, both the value and the complement round-trip exactly, which is even more than the PUTRL law requires.

$$\begin{aligned} putl(putr(x, (x, \mathsf{false}, n))) &= putl(\mathsf{inl}\ (), (x, \mathsf{false}, n)) \\ &= (x, (x, \mathsf{false}, n)) \\ putl(putr(x, (x, \mathsf{true}, n))) &= putl(\mathsf{inr}\ n, (x, \mathsf{true}, n)) \\ &= (x, (x, \mathsf{true}, n)) \qquad \square \end{aligned}$$

As with products, where we have a useful lens of type $X \leftrightarrow X \times X$ that is nevertheless not a diagonal lens, we can craft a useful conditional lens of type $X+X \leftrightarrow X$ that is nevertheless not a codiagonal lens. In fact, we define a more general lens $union \in X + Y \leftrightarrow X \cup Y$. Occasionally, a value that is both an X and a Y may be put to the left across one of these union lenses. In this situation, the lens may legitimately choose either an inr tag or an inl tag. Below, we propose two lenses that break this tie in different ways. The *union* lens uses the most recent unambiguous put to break the tie. The *union*′ lens, on the other hand, looks back to the last tagged value that was put to the right that was in both sets.

2.5.8 Definition [Union lens]:

$$union_{XY} \in X + Y \leftrightarrow X \cup Y$$

$$\begin{aligned} C &= Bool \\ missing &= \mathsf{false} \\ putr(\mathsf{inl}\ x, c) &= (x, \mathsf{false}) \\ putr(\mathsf{inr}\ y, c) &= (y, \mathsf{true}) \\ putl(xy, c) & \\ &= \begin{cases} (\mathsf{inl}\ xy, \mathsf{false}) & xy \notin Y \vee (xy \in X \wedge \neg c) \\ (\mathsf{inr}\ xy, \mathsf{true}) & xy \notin X \vee (xy \in Y \wedge c) \end{cases} \end{aligned}$$

Proof of well-formedness:
PUTRL:

$$\begin{array}{rcl} putl(putr(\mathsf{inl}\ x, c)) & = & putl(x, \mathsf{false}) \\ & = & (\mathsf{inl}\ x, \mathsf{false}) \\ putl(putr(\mathsf{inr}\ y, c)) & = & putl(y, \mathsf{true}) \\ & = & (\mathsf{inr}\ y, \mathsf{true}) \end{array}$$

PUTLR: There are six cases to consider, corresponding to which of the sets X, Y, and $X \cap Y$ our value is a member of and to whether the complement is true or false.

$$\begin{array}{rcl} putr(putl(xy, \mathsf{false})) & = & putr(\mathsf{inl}\ xy, \mathsf{false}) \\ & = & (xy, \mathsf{false}) \\ putr(putl(x, \mathsf{false})) & = & putr(\mathsf{inl}\ x, \mathsf{false}) \\ & = & (x, \mathsf{false}) \\ putr(putl(y, \mathsf{false})) & = & putr(\mathsf{inr}\ y, \mathsf{true}) \\ & = & (y, \mathsf{true}) \end{array}$$

The cases for when the complement is true are symmetric. □

2.5.9 Definition [Another union lens]: Given two sets X and Y, let's define a few bijections:

$$\begin{array}{rcl} f & \in & X \to X \setminus Y + X \cap Y \\ g & \in & Y \to X \cap Y + Y \setminus X \\ h & \in & X \setminus Y + X \cap Y + Y \setminus X \to X \cup Y \\ f(x) & = & \begin{cases} \mathsf{inl}\ x & x \notin Y \\ \mathsf{inr}\ x & x \in Y \end{cases} \\ g(y) & = & \begin{cases} \mathsf{inl}\ y & y \in X \\ \mathsf{inr}\ y & y \notin X \end{cases} \\ h(\mathsf{inl}\ x) & = & x \\ h(\mathsf{inr}\ (\mathsf{inl}\ xy)) & = & xy \\ h(\mathsf{inr}\ (\mathsf{inr}\ y)) & = & y \end{array}$$

$$union'_{XY} \in X + Y \leftrightarrow X \cup Y$$

$$\begin{array}{rcl} union'_{XY} & = & bij_{(f+g);\alpha^+;(id+(\alpha^+)^{-1})}; \\ & & (id_X \oplus (union_{X\cap Y, X\cap Y} \oplus id_Y)); \\ & & bij_h \end{array}$$

These definitions are not symmetric in X and Y, because *putl* prefers to return an inl value if there have been no tie breakers yet. Because of this preference, neither *union* nor *union′* can be used to construct a true codiagonal. However, there are two useful related constructions:

2.5.10 Definition [Switch lens]:

$$switch_X \in X + X \leftrightarrow X$$

$$switch_X \;=\; union_{XX}$$

We've used *union* rather than *union′* in this definition, but it actually doesn't matter: the two lenses' tie-breaking methods are equivalent when $X = Y$:

2.5.11 Lemma:

$$union_{XX} \equiv union'_{XX}$$

Proof: The relation that equates the states of the two *union* lenses is a witness: $R = \{(b, (((), (b, ())), ())) \mid b \in Bool\}$. □

2.5.12 Definition [Retentive case lens]:

$$\frac{k \in X \leftrightarrow Z \qquad \ell \in Y \leftrightarrow Z}{case_{k,\ell} \in X + Y \leftrightarrow Z}$$

$$case_{k,\ell} = (k \oplus \ell); switch_X$$

2.5.13 Definition [Forgetful case lens]:

$$\frac{k \in X \leftrightarrow Z \qquad \ell \in Y \leftrightarrow Z}{case^f_{k,\ell} \in X + Y \leftrightarrow Z}$$

$$case^f_{k,\ell} = (k \oplus^f \ell); switch_X$$

Lists We can also define a variety of lenses operating on lists. We only consider mapping here, because in the next section we show how to obtain this and a whole variety of other functions on lists as instances of a powerful generic theorem, but it is useful to see one concrete instance first!

Write $X^\star$ for the set of lists with elements from the set X. Write $\langle\rangle$ for the empty list and $x{:}xs$ for the list with head x and tail xs. Write X^ω for the set of infinite lists over X. When $x \in X$ and $ss \in X^\omega$, write $x{:}ss \in X^\omega$ for the infinite list with head x and tail ss. Write $x^\omega \in X^\omega$ for the infinite list of x's.

2.5.14 Definition [Retentive list mapping lens]:

$$\frac{\ell \in X \leftrightarrow Y}{\mathsf{map}(\ell) \in X^\star \leftrightarrow Y^\star}$$

$$\begin{array}{lcl}
C & = & (\ell.C)^\omega \\
\mathit{missing} & = & (\ell.\mathit{missing})^\omega \\
\mathit{putr}(x,c) & = & \mathsf{let}\ \langle x_1,\ldots,x_m\rangle = x\ \mathsf{in} \\
 & & \mathsf{let}\ \langle c_1,\ldots\rangle = c\ \mathsf{in} \\
 & & \mathsf{let}\ (y_i,c_i') = \ell.\mathit{putr}(x_i,c_i)\ \mathsf{in} \\
 & & (\langle y_1,\ldots,y_m\rangle, \langle c_1',\ldots,c_m',c_{m+1},\ldots\rangle) \\
\mathit{putl} & & (\text{similar})
\end{array}$$

The map lens gives us the machinery we need to complete the first example in the introduction: simply define $e^\star = \mathsf{map}(e)$. Additionally, as we saw in §2.1, there is also a forgetful variant of the list mapping lens. Indeed, this is the one that corresponds to the known list mapping operator on asymmetric, state-based lenses [8, 16].

2.5.15 Definition [Forgetful list mapping lens]:

$$\frac{\ell \in X \leftrightarrow Y}{\mathsf{map}^f(\ell) \in X^\star \leftrightarrow Y^\star}$$

$$\begin{array}{lcl}
C & = & \ell.C^\star \\
\mathit{missing} & = & \langle\rangle \\
\mathit{putr}(x,c) & = & \mathsf{let}\ \langle x_1,\ldots,x_m\rangle = x\ \mathsf{in} \\
 & & \mathsf{let}\ \langle c_1,\ldots,c_n\rangle = c\ \mathsf{in} \\
 & & \mathsf{let}\ \langle c_{n+1},\ldots\rangle = (\ell.\mathit{missing})^\omega\ \mathsf{in} \\
 & & \mathsf{let}\ (y_i,c_i') = \ell.\mathit{putr}(x_i,c_i)\ \mathsf{in} \\
 & & (\langle y_1,\ldots,y_m\rangle, \langle c_1',\ldots,c_m'\rangle) \\
\mathit{putl} & & (\text{similar})
\end{array}$$

Rather than proving that these two forms of list mapping are lenses, preserve equivalence, induce functors, and so on, we show that these properties hold for a generalization of their construction in the next section.

We can make the relationship between the retentive sum and map lenses and the forgetful sum and map lenses precise; the following two diagrams commute:

$$\begin{array}{rcl} \mathit{Unit} + X \times X^\star & \xrightarrow{\mathit{bij}} & X^\star \\ {\scriptstyle id_{\mathit{Unit}} \oplus (\ell \otimes \mathsf{map}(\ell))}\downarrow & & \downarrow{\scriptstyle \mathsf{map}(\ell)} \\ \mathit{Unit} + Y \times Y^\star & \xrightarrow{\mathit{bij}} & Y^\star \end{array}$$

$$\begin{array}{rcl} \mathit{Unit} + X \times X^\star & \xrightarrow{\mathit{bij}} & X^\star \\ {\scriptstyle id_{\mathit{Unit}} \oplus^f (\ell \otimes \mathsf{map}^f(\ell))}\downarrow & & \downarrow{\scriptstyle \mathsf{map}^f(\ell)} \\ \mathit{Unit} + Y \times Y^\star & \xrightarrow{\mathit{bij}} & Y^\star \end{array}$$

2.6 Iterators

In functional programming, mapping functionals are usually seen as instances of more general "fold patterns," or defined by general recursion. In this section, we investigate to what extent this path can be followed in the world of symmetric lenses.

Allowing general recursive definitions for symmetric lenses may be possible, but in general, complements change when unfolding a recursive definition; this means that the structure of the complement of the recursively defined function would itself have to be given by some kind of fixpoint construction. Preliminary investigation suggests that this is possible, but it would considerably clutter the development—on top of the general inconvenience of having to deal with partiality.

Therefore, we choose a different path. We identify a "fold" combinator for lists, reminiscent of the view of lists as initial algebras. We show that several important lenses on lists—including, of course, the mapping combinator—can be defined with the help of a fold, and that, due to the self-duality of lenses, folds can be composed back-to-back to yield general recursive patterns in the style of *hylomorphisms* [31].

We also discuss iteration patterns on trees and argue that the methodology carries over to other polynomial inductive datatypes.

2.6.1 Lists

Let $\mathit{fold} \in \mathit{Unit} + (X \times X^\star) \to X^\star$ be the bijection between "unfolded" lists and lists; *fold* takes $\mathsf{inl}\ ()$ to $\langle\rangle$ and $\mathsf{inr}\ (x, xs)$ to $x{:}xs$. Note that $\mathit{bij}_{\mathit{fold}} \in \mathit{Unit} + (X \times X^\star) \iff X^\star$

is then a bijective arrow in the category LENS.

2.6.1.1 Definition [X-list algebra]: An *X-list algebra* on a set Z is an arrow $\ell \in \mathit{Unit} + (X \times Z) \Longleftrightarrow Z$ and a weight function $w \in Z \to \mathbb{N}$ such that $\ell.\mathit{putl}(z, c) = (\mathsf{inr}\ (x, z'), c')$ implies $w(z') < w(z)$. We write $T^\star_X$ for the functor that sends any lens k to $\mathit{id}_{\mathit{Unit}} \oplus (\mathit{id}_X \otimes k)$.

The function w here plays the role of a termination measure. We will be iterating $\ell.\mathit{putl}$, producing a stream of values of type Z, which we would like to guarantee eventually ends.

2.6.1.2 Theorem [Iteration is well-defined]: For X-list algebra ℓ on Z, there is a unique arrow $\mathit{It}(\ell) \in X^\star \Longleftrightarrow Z$ such that the following diagram commutes:

$$\begin{array}{ccc} T^\star_X(X^\star) & \xrightarrow{\mathit{bij}_{\mathit{fold}}} & X^\star \\ {\scriptstyle T^\star_X(\mathit{It}(\ell))}\downarrow & & \downarrow{\scriptstyle \mathit{It}(\ell)} \\ T^\star_X(Z) & \xrightarrow[\ell]{} & Z \end{array}$$

In the terminology of universal algebra, an algebra for a functor F from some category to itself is simply an object Z and an arrow $F(Z) \to Z$. An arrow between F-algebras (Z, f) and (Z', f') is an arrow $u \in Z \to Z'$ such that $f; u = F(u); f'$. The F-algebras thus form a category themselves. An initial F-algebra is an initial object in that category (an initial object has exactly one arrow to each other object, and is unique up to isomorphism). F-algebras can be used to model a wide variety of inductive datatypes, including lists and various kinds of trees [41]. Using this terminology, Theorem 2.6.1.2 says that $\mathit{bij}_{\mathit{fold}}$ is an initial object in the subcategory consisting of those $T^\star_X$-algebras for which a weight function w is available.

Before we give the proof, let us consider some concrete instances of the theorem. First, if $k \in X \Longleftrightarrow Y$ is a lens, then we can form an X-list algebra ℓ on $Y^\star$ by composing two lenses as follows:

$$\mathit{Unit} + (X \times Y^\star) \xrightarrow{\mathit{id}_{\mathit{Unit}} \oplus (k \otimes \mathit{id}_{Y^\star})} \mathit{Unit} + (Y \times Y^\star) \xrightarrow{\mathit{bij}_{\mathit{fold}}} Y^\star$$

A suitable weight function is given by $w(ys) = \mathit{length}(ys)$. The induced lens $\mathit{It}(\ell) \in X^\star \Longleftrightarrow Y^\star$ is the lens analog of the familiar list mapping function. In fact, substituting the lens $e \in X \times Y \Longleftrightarrow Y \times Z$ (from the introduction) for k in the above diagram, we find that $\mathit{It}(\ell)$ is the sneakier variant of the lens $e^\star$. (Again, we are ignoring the important question of alignment here. A hand-written map lens could perform a more sophisticated alignment analysis to associate "similar" items in a sequence of puts and recover more appropriate data from the complement; the process described above results in a simple positional alignment scheme.)

Second, suppose that $X = X_1 + X_2$ and let Z be $X_1^\star \times X_2^\star$. Writing X_i^+ for $X_i \times X_i^\star$, we can define isomorphisms

$$\begin{array}{rcl} f & \in & (X_1 + X_2) \times X_1^\star \times X_2^\star \\ & \to & (X_1^+ + X_2^+) + (X_1^+ \times X_2^+ + X_1^+ \times X_2^+) \\ g & \in & \mathit{Unit} + ((X_1^+ + X_2^+) + X_1^+ \times X_2^+) \\ & \to & X_1^\star \times X_2^\star \end{array}$$

by distributing the sum and unfolding the list type for f and by factoring the polynomial and folding the list type for g.[3]

$$\begin{array}{rcl} f(\mathsf{inl}\ x_1, xs_1, \langle\rangle) & = & \mathsf{inl}\ (\mathsf{inl}\ (x_1, xs_1)) \\ f(\mathsf{inl}\ x_1, xs_1, x_2{:}xs_2) & = & \mathsf{inr}\ (\mathsf{inl}\ ((x_1, xs_1), (x_2, xs_2))) \\ f(\mathsf{inr}\ x_2, \langle\rangle, xs_2) & = & \mathsf{inl}\ (\mathsf{inr}\ (x_2, xs_2)) \\ f(\mathsf{inr}\ x_2, x_1{:}xs_1, xs_2) & = & \mathsf{inr}\ (\mathsf{inr}\ ((x_1, xs_1), (x_2, xs_2))) \end{array}$$

$$\begin{array}{rcl} g(\mathsf{inl}\ ()) & = & (\langle\rangle, \langle\rangle) \\ g(\mathsf{inr}\ (\mathsf{inl}\ (\mathsf{inl}\ (x_1, xs_1)))) & = & (x_1 : xs_1, \langle\rangle) \\ g(\mathsf{inr}\ (\mathsf{inl}\ (\mathsf{inr}\ (x_2, xs_2)))) & = & (\langle\rangle, x_2 : xs_2) \\ g(\mathsf{inr}\ (\mathsf{inr}\ ((x_1, xs_1), (x_2, xs_2)))) & = & (x_1 : xs_1, x_2 : xs_2) \end{array}$$

Then we can create

$$\ell \in \mathit{Unit} + ((X_1 + X_2) \times Z) \leftrightarrow Z$$

$$\begin{array}{rcl} \ell & = & (\mathit{id}_{\mathit{Unit}} \oplus \mathit{bij}_f); \\ & & (\mathit{id}_{\mathit{Unit}} \oplus (\mathit{id}_{X_1^+ + X_2^+} \oplus \mathit{switch}_{X_1^+ \times X_2^+})); \\ & & \mathit{bij}_g \end{array}$$

A suitable weight function for ℓ is given by

$$w((xs_1, xs_2)) = \mathit{length}(xs_1) + \mathit{length}(xs_2).$$

The lens $\mathit{It}(\ell) \in (X_1 + X_2)^\star \Longleftrightarrow X_1^\star \times X_2^\star$ that we obtain from iteration partitions the input list in one direction and uses a stream of booleans from the state to put them

[3]The bijections f and g can be written in terms of the associators, symmetries, unfolds, folds, and so forth that were already introduced, so the lenses bij_f and bij_g would not have to be defined "out of whole cloth" as they are here, but these definitions get bogged down in syntax without adding much value.

back in the right order in the other direction. Indeed, $It(\ell)$ is exactly the *partition* lens described in the introductory examples. Composing it with a projection yields a filter lens. (Alternatively, the filter lens could be obtained directly by iterating a slightly trickier ℓ.) Consequently, we now have the machinery we need to define *comp* from the introduction:

$$\begin{aligned} \mathit{filter} &= \mathit{partition}; \pi_{1\langle\rangle} \\ \mathit{comp} &= \mathit{filter}; \mathit{filter}^{op} \end{aligned}$$

Proof of 2.6.1.2: We define the lens $It(\ell)$ explicitly.

$$\frac{\ell \in T^{\star}_{X}(Z) \leftrightarrow Z \qquad \exists \text{ suitable } w}{It(\ell) \in X^{\star} \leftrightarrow Z}$$

$$\begin{array}{lcl}
It(\ell).C & = & (\ell.C)^{\omega} \\
It(\ell).\mathit{missing} & = & (\ell.\mathit{missing})^{\omega} \\
It(\ell).\mathit{putr}(\langle\rangle, c{:}cs) & = & \mathsf{let}\ (z, c') = \ell.\mathit{putr}(\mathsf{inl}\ (), c)\ \mathsf{in} \\
 & & (z, c'{:}cs) \\
It(\ell).\mathit{putr}(x{:}xs, c{:}cs) & = & \mathsf{let}\ (z, cs') = It(\ell).\mathit{putr}(xs, cs)\ \mathsf{in} \\
 & & \mathsf{let}\ (z', c') = \ell.\mathit{putr}(\mathsf{inr}\ (x, z), c)\ \mathsf{in} \\
 & & (z', c'{:}cs') \\
It(\ell).\mathit{putl}(z, c{:}cs) & = & \mathsf{match}\ \ell.\mathit{putl}(z, c)\ \mathsf{with} \\
 & & \quad (\mathsf{inl}\ (), c') \rightarrow (\langle\rangle, c'{:}cs) \\
 & & \mid\ \ (\mathsf{inr}\ (x, z'), c') \rightarrow \\
 & & \quad \mathsf{let}\ (xs, cs') = It(\ell).\mathit{putl}(z', cs)\ \mathsf{in} \\
 & & \quad (x{:}xs, c'{:}cs')
\end{array}$$

Note that the first element of the complement list holds *both* the complement that is used when we do a *putr* of an empty list *and* the complement that is used for the first element when we do a *putr* of a non-empty list. Similarly, the second element of the complement list holds both the complement that is used at the end of the *putr* of a one-element list and the complement that is used for the second element when we do a *putr* of a two or more element list.

The recursive definition of $It(\ell).\mathit{putr}$ is clearly terminating because the first argument to the recursive call is always a shorter list; the recursive definition of $It(\ell).\mathit{putl}$ is terminating because the value of w is always smaller on the arguments to the recursive call. The round-trip laws are readily established by induction on xs and on $w(z)$, respectively. So this is indeed a lens.

Commutativity of the claimed diagram is a direct consequence of the defining equations (which have been crafted so as to make commutativity hold).

To show uniqueness, let $k \in X^\star \Longleftrightarrow Z$ be another lens for which the diagram commutes—i.e., such that:

$$\begin{array}{ccc} T^\star_X(X^\star) & \xrightarrow{bij_{fold}} & X^\star \\ {\scriptstyle T^\star_X(k)}\downarrow & & \downarrow{\scriptstyle k} \\ T^\star_X(Z) & \xrightarrow[\ell]{} & Z \end{array}$$

Choose representatives of the equivalence classes k and ℓ—for convenience, call these representatives k and ℓ. Let $R \subseteq k.C \times (k.C \times \ell.C)$ be a simulation relation witnessing the commutativity of this diagram (recalling that equality of LENS-arrows means lens-equivalence of representatives). Notice that $k.C$ is the complement of (a representative of) the upper path through the diagram, and $k.C \times \ell.C$ is the complement of (a representative of) the lower path through the diagram. (Strictly speaking, the complements are $Unit \times k.C$ and $Unit \times Unit \times k.C \times \ell.C$; using these isomorphic forms reduces clutter.) Thus, the commutativity of the diagram means:

$$(k.missing, (k.missing, \ell.missing)) \in R$$

$$\frac{(d, (d', c)) \in R \qquad k.putr(\langle\rangle, d) = (z, d_1) \qquad \ell.putr(\mathsf{inl}\ (), c) = (z', c_1)}{(d_1, (d', c_1)) \in R \ \wedge\ z = z'}$$

$$\frac{(d, (d', c)) \in R \qquad k.putr(x{:}xs, d) = (z, d_1) \qquad k.putr(xs, d') = (z', d'_1) \qquad \ell.putr(\mathsf{inr}\ (x, z'), c) = (z'', c_1)}{(d_1, (d'_1, c_1)) \in R \ \wedge\ z = z''}$$

$$\frac{(d, (d', c)) \in R \qquad k.putl(z, d) = (\langle\rangle, d_1)}{\ell.putl(z, c) = (\mathsf{inl}\ (), c_1) \wedge (d_1, (d', c_1)) \in R}$$

$$\frac{(d, (d', c)) \in R \qquad k.putl(z, d) = (x{:}xs, d_1)}{\begin{array}{rl} & \ell.putl(z, c) = (\mathsf{inr}\ (x, z'), c_1) \\ \wedge & k.putl(z', d') = (xs, d'_1) \\ \wedge & (d_1, (d'_1, c_1)) \in R \end{array}}$$

The variables c_1, z', d'_1 in the last two rules are existentially quantified.

In order to show that $It(\ell) \equiv k$ we define a relation $S \subseteq It(\ell).C \times k.C$ inductively as follows:

$$(It(\ell).missing, k.missing) \in S$$

$$\frac{(d, (d', c)) \in R \qquad (cs, d') \in S}{(c{:}cs, d) \in S}$$

Notice that if $(c{:}cs, d) \in S$ by either one of the rules, then there exists d' such that $(d, (d', c)) \in R$ and $(cs, d') \in S$. In particular, for the first rule, $c{:}cs = It(\ell).missing$ and we choose $d' = k.missing$.

It remains to show that S is compatible with *putl* and *putr*. So assume that $(c{:}cs, d) \in S$, hence $(d, (d', c)) \in R$ and $(cs, d') \in S$ for some d'. We proceed by induction on $length(xs)$ in the *putr* cases and by induction on $w(z)$ in the *putl* cases.

Case for *putr* of empty list: By definition,

$$It(\ell).putr(\langle\rangle, c{:}cs) = (z, c'{:}cs),$$

where $(z, c') = \ell.putr(\mathsf{inl}\ (), c)$. Let $(z_1, d_1) = k.putr(\langle\rangle, d)$. Commutativity of the diagram then tells us that $(d_1, (d', c')) \in R$ and $z_1 = z$. Since $(cs, d') \in S$, we can conclude $(c'{:}cs, d_1) \in S$, as required.

Case for *putr* of nonempty list: This time, the definition gives us

$$It(\ell).putr(x{:}xs, c{:}cs) = (z', c'{:}cs'),$$

where

$$\begin{array}{rcl} (z, cs') & = & It(\ell).putr(xs, cs) \\ (z', c') & = & \ell.putr(\mathsf{inr}\ (x, z), c). \end{array}$$

Let

$$\begin{array}{rcl} (z_1, d_1) & = & k.putr(x{:}xs, d) \\ (z_2, d_2) & = & k.putr(xs, d') \\ (z_3, c_3) & = & \ell.putr(\mathsf{inr}\ (x, z_2), c). \end{array}$$

Inductively, we get $z_2 = z$ and $(cs', d_2) \in S$. Thus, $z_3 = z'$ and $c_3 = c'$. From commutativity we get $z_1 = z'$ and $(d_1, (d_2, c')) \in R$, so $(c'{:}cs', d_1) \in S$ and we are done.

Case where *It.putl* on z returns the empty list: Suppose we have $It(\ell).putl(z, c{:}cs) = (\langle\rangle, c'{:}cs)$, where $(\mathsf{inl}\ (), c') = \ell.putl(z, c)$. Let $k.putr(z, d) = (xs, d_1)$. Commutativity of the diagram asserts that $(d_1, (c', d')) \in R$ and $xs = \langle\rangle$. Now, since $(cs, d') \in S$, we can conclude $(c'{:}cs, d_1) \in S$, as required.

Case where *It.putl* on z returns a non-empty list: Suppose we have

$$\begin{array}{rcl} It(\ell).putl(z, c{:}cs) & = & (x{:}xs, c'{:}cs') \\ (\mathsf{inr}\ (x, z'), c') & = & \ell.putl(z, c) \\ (xs, cs') & = & It(\ell).putl(z', cs). \end{array}$$

Since $\ell.putl(z, c)$ returns an inr we are in the situation of the fourth rule above and we have $k.putl(z, d) = (x{:}xs', d_1)$ for some xs' and d_1. Furthermore, we have $k.putl(z', d') = (xs', d_1')$ and $(d_1, (d_1', c_1)) \in R$. The induction hypothesis applied to z' in view of $w(z') < w(z)$ then yields $xs' = xs$ and also $(cs', d_1') \in S$. It then follows $(c'{:}cs', d_1) \in S$ and we are done. □

2.6.1.3 Corollary [Hylomorphism]: Suppose k^{op} is an X-list algebra on W and ℓ is an X-list algebra on Z. Then there is a lens $Hy(k,\ell) \in W \Longleftrightarrow Z$ such that the following diagram commutes:

$$\begin{array}{ccc} T_X^\star(W) & \xleftarrow{\quad k \quad} & W \\ {\scriptstyle T_X^\star(Hy(k,\ell))}\downarrow & & \downarrow{\scriptstyle Hy(k,\ell)} \\ T_X^\star(Z) & \xrightarrow[\quad \ell \quad]{} & Z \end{array}$$

Proof: Define $Hy(k,\ell)$ as the composition $It(k^{op})^{op}; It(\ell)$. □

One can think of $Hy(k,\ell)$ as a recursive definition of a lens. The lens k tells whether a recursive call should be made, and if so, produces the argument for the recursive call and some auxiliary data. The lens ℓ then describes how the result is to be built from the result of the recursive call and the auxiliary data. This gives us a lens version of the hylomorphism pattern from functional programming [31]. Unfortunately, we were unable to prove or disprove the uniqueness of $Hy(k,\ell)$.

We have not formally studied the question of whether $It(\ell)$ is actually an initial algebra, i.e., whether it can be defined and is unique even in the absence of a weight function. However, this seems unlikely, because then it would apply to the case where Z is the set of finite and infinite X lists and ℓ the obvious bijective lens. The *putl* component of $It(\ell)$ would then have to truncate an infinite list, which would presumably break the commuting square.

2.6.2 Other Datatypes

Analogs of Theorem 2.6.1.2 and Corollary 2.6.1.3 are available for a number of other functors, in particular those that are built up from variables by $+$ and $\times$. All of these can also be construed as containers (see §2.7), but the iterator and hylomorphism patterns provide more powerful operations for the construction of lenses than the mapping operation available for general containers. Moreover, the universal property of the iterator provides a modular proof method, allowing one to deduce equational laws which can be cumbersome to establish directly because of the definition of equality as behavioral equivalence. For instance, we can immediately deduce that list mapping is a functor. Containers, on the other hand, subsume datatypes such as labeled graphs that are not initial algebras.

Iterators with multiple arguments The list iterator allows us to define a lens between $X^\star$ and some other set Z, but Theorem 2.6.1.2 cannot be directly used to define a lens between $X^\star \times Y$ and Z (think of Y as modeling parameters). In standard functional programming, a map from $X^\star \times Y$ to Z is tantamount to a map from $X^\star$ to $Y \to Z$, so iteration with parameters is subsumed by the parameterless

case. Unfortunately, LENS does not seem to have the function spaces required to play this trick.

Therefore, we introduce the functor $T^{\star}_{X,Y}(Z) = Y + X \times Z$ and notice that $T^{\star}_{X,Y}(X^{\star} \times Y) \simeq X^{\star} \times Y$. Just as before, an algebra for that functor is a lens $\ell \in T^{\star}_{X,Y}(Z) \leftrightarrow Z$ together with a function $w : Z \to \mathbb{N}$ such that $\ell.putl(z, c) = (\mathsf{inr}\ (x, z'), c')$ implies $w(z') < w(z)$.

As an example, let $Y = Z = X^{\star}$ and define

$$\boxed{\ell \in X^{\star} + X \times X^{\star} \leftrightarrow X^{\star}}$$

$$\boxed{\begin{array}{lcl} C & = & \mathit{Bool} \\ \mathit{missing} & = & \mathsf{true} \\ \ell.\mathit{putr}(\mathsf{inl}\ xs, b) & = & (xs, \mathsf{true}) \\ \ell.\mathit{putr}(\mathsf{inr}\ (x, xs), b) & = & (x{:}xs, \mathsf{false}) \\ \ell.\mathit{putl}(\langle\rangle, b) & = & (\mathsf{inl}\ \langle\rangle, \mathsf{true}) \\ \ell.\mathit{putl}(x{:}xs, \mathsf{true}) & = & (\mathsf{inl}\ (x{:}xs), \mathsf{true}) \\ \ell.\mathit{putl}(x{:}xs, \mathsf{false}) & = & (\mathsf{inr}\ (x, xs), \mathsf{false}) \end{array}}$$

Iteration yields a lens $X^{\star} \times X^{\star} \leftrightarrow X^{\star}$ that can be seen as a bidirectional version of list concatenation. The commuting square for the iterator corresponds to the familiar recursive definition of concatenation: $\mathit{concat}(\langle\rangle, ys) = ys$ and $\mathit{concat}(x{:}xs, ys) = x{:}\mathit{concat}(xs, ys)$. In the bidirectional case considered here the complement will automatically retain enough information to allow splitting in the *putl*-direction.

We can use a version of Corollary 2.6.1.3 for this data structure to implement tail recursive constructions. Consider, for instance, the $T^{\star}_{\mathit{Unit},X^{\star}}$-algebra $k : X^{\star} + X^{\star} \times X^{\star} \leftrightarrow X^{\star} \times X^{\star}$ where

$$\begin{array}{lcl} k.\mathit{putl}((\mathit{acc}, \langle\rangle), \mathsf{true}) & = & (\mathsf{inl}\ \mathit{acc}, \mathsf{true}) \\ k.\mathit{putl}((\mathit{acc}, x{:}xs), \mathsf{true}) & = & (\mathsf{inr}\ (x{:}\mathit{acc}, xs), \mathsf{true}) \\ k.\mathit{putl}((\mathit{acc}, xs), \mathsf{false}) & = & (\mathsf{inr}\ (\mathit{acc}, xs), \mathsf{false}). \end{array}$$

Together with the $T^{\star}_{\mathit{Unit},X^{\star}}$-algebra $\mathit{switch}_{X^{\star}} : X^{\star} + X^{\star} \leftrightarrow X^{\star}$, this furnishes a bidirectional version of the familiar tail recursive list reversal that sends (acc, xs) to $xs^{\mathit{rev}}\,\mathit{acc}$.

Trees For set X let $\mathit{Tree}(X)$ be the set of binary X-labeled trees given inductively by $\mathit{leaf} \in \mathit{Tree}(X)$ and $x \in X, \ell \in \mathit{Tree}(X), r \in \mathit{Tree}(X) \Rightarrow \mathit{node}(x, \ell, r) \in \mathit{Tree}(X)$. Consider the endofunctor T^{Tree}_X given by $T^{\mathit{Tree}}_X(Z) = \mathit{Unit} + X \times Z \times Z$. Let $c \in T^{\mathit{Tree}}_X(\mathit{Tree}(X)) \leftrightarrow \mathit{Tree}(X)$ denote the obvious bijective lens.

An X-tree algebra is a lens $\ell \in T_X^{Tree}(Z) \leftrightarrow Z$ and a function $w \in Z \to \mathbb{N}$ with the property that if $\ell.putl(z, c) = (\mathsf{inr}\ (x, z_l, z_r), c')$ then $w(z_l) < w(z)$ and $w(z_r) < w(z)$. The bijective lens c is then the initial object in the category of X-tree algebras; that is, every X-tree algebra on Z defines a unique lens in $Tree(X) \leftrightarrow Z$.

Consider, for example, the concatenation lens $concat : X^\star \times X^\star \leftrightarrow X^\star$. Let $concat' : Unit + X \times X^\star \times X^\star \leftrightarrow X^\star$ be the lens obtained from $concat$ by precomposing with the fold-isomorphism and the terminal lens $term_{\langle\rangle}$. Intuitively, this lens sends $\mathsf{inl}\ ()$ to $\langle\rangle$ and x, xs, xs' to $x{:}xs@xs'$, using the complement to undo this operation properly. This lens forms an example of a tree algebra (with number of nodes as weight functions) and thus iteration furnishes a lens $Tree(X) \leftrightarrow X^\star$ which does a pre-order traversal, keeping enough information in the complement to rebuild a tree from a modified traversal.

The hylomorphism pattern can also be applied to trees, yielding the ability to define symmetric lenses by divide-and-conquer, i.e., by dispatching one call to two parallel recursive calls whose results are then appropriately merged.

2.7 Containers

The previous section suggests a construction for a variety of operations on datatypes built from polynomial functors. Narrowing the focus to the very common "map" operation, we can generalize still further, to any kind of *container functor* [1], i.e. a *normal functor* in the terminology of Hasegawa [19] or an *analytic functor* in the terminology of Joyal [27]. (These structures are also related to the *shapely types* of Jay and Cockett [25].)

2.7.1 Definition [Container]: A *container* consists of a set I together with an I-indexed family of sets $B \in I \to Set$.

Each container (I, B) gives rise to an endofunctor $F_{I,B}$ on SET whose object part is defined by $F_{I,B}(X) = \sum_{i \in I} B(i) \to X$. For example, if $I = \mathbb{N}$ and $B(n) = \{0, 1, \ldots, n-1\}$, then $F_{I,B}(X)$ is $X^\star$ (up to isomorphism). Or, if $I = Tree(Unit)$ is the set of binary trees with trivial labels and $B(i)$ is the set of nodes of i, then $F_{I,B}(X)$ is the set of binary trees labeled with elements of X. In general, we can think of I as a set of shapes and, for each shape $i \in I$, we can think of $B(i)$ as the set of "positions" in shape i. So an element $(i, f) \in F_{I,B}(X)$ consists of a shape i and a function f assigning an element $f(p) \in X$ to each position $p \in B(i)$.

The arrow part of $F_{I,B}$ maps a function $u \in X \to Y$ to a function $F_{I,B}(u) \in F_{I,B}(X) \to F_{I,B}(Y)$ given by $(i, f) \mapsto (i, f; u)$.

Now, we would like to find a way to view a container as a functor on the category of lenses. In order to do this, we need a little extra structure.

2.7.2 Definition: A *container with ordered shapes* is a pair (I, B) satisfying these conditions:

1. I is a partial order with binary meets. We say i is a *subshape* of j whenever $i \leq j$.

2. B is a functor from $(I, \leq)$ viewed as a category (with one object for each element and an arrow from i to j iff $i \leq j$) into SET. When B and i are understood, we simply write $b|i'$ for $B(i \leq i')(b)$ if $b \in B(i)$ and $i \leq i'$.

3. If i and i' are both subshapes of a common shape j and we have positions $b \in B(i)$ and $b' \in B(i')$ with $b|j = b'|j$, then there must be a unique $b_0 \in B(i \wedge i')$ such that $b = b_0|i$ and $b' = b_0|i'$. Thus such b and b' are really the same position. In other words, every diagram of the following form is a pullback:

$$\begin{array}{ccc} B(i \wedge i') & \xrightarrow{B(i \wedge i' \leq i)} & B(i) \\ {\scriptstyle B(i \wedge i' \leq i')}\downarrow & & \downarrow{\scriptstyle B(i \leq j)} \\ B(i') & \xrightarrow[B(i' \leq j)]{} & B(j) \end{array}$$

If $i \leq j$, we can apply the instance of the pullback diagram where $i = i'$ and hence $i \wedge i' = i$ and deduce that $B(i \leq j) \in B(i) \to B(j)$ is always injective.

For example, in the case of trees, we can define $t \leq t'$ if every path from the root in t is also a path from the root in t'. The arrow part of B then embeds positions of a smaller tree canonically into positions of a bigger tree. The meet of two trees is the greatest common subtree starting from the root.

2.7.3 Definition [Container mapping lens]:

$$\frac{\ell \in X \leftrightarrow Y}{F_{I,B}(\ell) \in F_{I,B}(X) \leftrightarrow F_{I,B}(Y)}$$

$$\begin{array}{l} C = \\ \quad \{t \in \prod_{i \in I} B(i) \to \ell.(C) \mid \\ \quad\quad \forall i, i'.\ i \leq i' \supset \forall b \in B(i).\ t(i')(b|i') = t(i)(b)\} \\ \mathit{missing}(i)(b) \ = \ \ell.\mathit{missing} \\ \mathit{putr}((i, f), t) \ = \\ \quad \mathsf{let}\ f'(b) \quad = \ \mathsf{fst}(\ell.\mathit{putr}(f(b), t(i)(b)))\ \mathsf{in} \\ \quad \mathsf{let}\ t'(j)(b) \ = \\ \quad\quad \mathsf{if}\ \exists b_0 \in B(i \wedge j).\ b_0|j = b \\ \quad\quad \mathsf{then}\ \mathsf{snd}(\ell.\mathit{putr}(f(b_0|i), t(j)(b)))\quad \mathsf{in} \\ \quad\quad \mathsf{else}\ t(j)(b) \\ \quad ((i, f'), t') \\ \mathit{putl} \qquad\qquad \text{(similar)} \end{array}$$

(Experts will note that C is the limit of the contravariant functor $i \mapsto (B(i) \to \ell.(C))$. Alternatively, we can construe C as the function space $D \to \ell.(C)$ where D is the colimit of the functor B. Concretely, D is given by $\sum_{i \in I} B(i)$ modulo the equivalence relation $\sim$ generated by $(i, b) \sim (i', b')$ whenever $i \leq i'$ and $b' = B(i \leq i')(b)$.)

Proof of well-formedness: To show that this definition is a lens, we should begin by checking that it is well typed—i.e., that the t' we build in *putr* really lies in the complement (the argument for *putl* will be symmetric). So suppose that $j \leq j'$ and $b \in B(j)$. There are two cases to consider:

1. $b = b_0|j$ for some (unique) $b_0 \in B(i \wedge j)$. Then $b|j' = b_0|j'$ so we are in the "then" branch in both $t'(j')(b|j')$ and $t'(j)(b)$, and the results are equal by the fact that $t \in C$.

2. b is not of the form $b_0|j$ for some (unique) $b_0 \in B(i \wedge j)$. We claim that then $b|j'$ is not of the form $b_1|j'$ for any $b_1 \in B(i \wedge j')$, so that we are in the "else" branch in both applications of t'. Since $t \in C$, this will conclude the proof of this case. To see the claim, assume for a contradiction that $b|j' = b_1|j'$ for some $b_1 \in B(i \wedge j')$. Applying the pullback property to the situation $i \wedge j \leq j \leq j'$ and $i \wedge j \leq i \wedge j' \leq j'$ yields a unique $b_0 \in B(i \wedge j)$ such that $b = b_0|j$ and $b_1 = b_0|(i \wedge j')$, contradicting the assumption.

It now remains to verify the lens laws. We will check PutRL; the PutLR law can be checked similarly. Suppose that

$$\begin{aligned} F_{I,B}(\ell).putr((i, f), t) &= ((i, f_r), t_r) \\ F_{I,B}(\ell).putl((i, f_r), t_r) &= ((i, f_{rl}), t_{rl}) \end{aligned}$$

We must check that $f_{rl} = f$ and $t_{rl} = t_r$.

Let us check that $f_{rl} = f$. Choose arbitrary $b \in B(i)$. Then

$$f_{rl}(b) = \mathsf{fst}(\ell.putl(f_r(b), t_r(i)(b))).$$

Inspecting the definition of t_r, we find that $t_r(i)(b) = \mathsf{snd}(\ell.putr(f(b), t(i)(b)))$, and from the definition of f_r, we find that $f_r(b) = \mathsf{fst}(\ell.putr(f(b), t(i)(b)))$. Together, these two facts imply that

$$f_{rl}(b) = \mathsf{fst}(\ell.putl(\ell.putr(f(b), t(i)(b))))$$

Applying PutRL to ℓ, this reduces to $f_{rl}(b) = f(b)$, as desired.

Finally, we must show that $t_{rl} = t_r$. Choose arbitrary $j \in I$ and $b \in B(j)$. There are two cases: either we have $b_0|j = b$ or not.

- Suppose $b_0|j = b$. Then we find that

$$t_{rl}(j)(b) = \mathsf{snd}(\ell.putl(f_r(b_0|i), t_r(j)(b)))$$

Now, inspecting the definitions of f_r and t_r, we find that this amounts to saying

$$t_{rl}(j)(b) = \mathsf{snd}(\ell.putl(\ell.putr(f(b_0|i), t(j)(b))))$$

Furthermore, we have $t_r(j)(b) = \mathsf{snd}(\ell.putr(f(b_0|i), t(j)(b)))$, so the PUTRL law applied to ℓ tells us that $t_{rl}(j)(b) = t_r(j)(b)$, as desired.

- Otherwise, there is no b_0 with that property. Then we find that $t_{rl}(j)(b) = t_r(j)(b)$ immediately from the definition of t_{rl}. □

Proof of preservation of equivalence: If R witnesses $k \equiv \ell$, then we relate functions that yield related outputs for each possible input:

$$R_{I,B} = \{(t_k, t_\ell) \mid \forall i, b.\ t_k(i)(b)\ R\ t_\ell(i)(b)\}$$

For any i and b, we can show

$$\begin{array}{rcl} F_{I,B}(k).missing(i)(b) & = & k.missing \\ k.missing & R & \ell.missing \\ \ell.missing & = & F_{I,B}(\ell).missing(i)(b) \end{array}$$

so the *missing* elements are related by $R_{I,B}$. Now suppose the following relationships hold:

$$t_k\ R_{I,B}\ t_\ell$$

$$\begin{array}{rcl} F_{I,B}(k).putr((i,f), t_k) & = & ((i, f_k), t'_k) \\ F_{I,B}(\ell).putr((i,f), t_\ell) & = & ((i, f_\ell), t'_\ell) \end{array}$$

We must show that $f_k = f_\ell$ and that $t'_k\ R_{I,B}\ t'_\ell$. The former follows directly; for any b, we have $f_k(b) = f_\ell(b)$ because $t_k(i)(b)\ R\ t_\ell(i)(b)$. For the latter, consider an arbitrary j and b. There are two cases. If $b_0|j = b$ for some $b_0 \in B(i \wedge j)$, then $t'_k(j)(b)\ R\ t'_\ell(j)(b)$ because k and ℓ preserve R-states; otherwise, $t'_k(j)(b)\ R\ t'_\ell(j)(b)$ because $t'_k(j)(b) = t_k(j)(b)$ and $t'_\ell(j)(b) = t_\ell(j)(b)$. □

Proof of functoriality: The complete relation (which has only one element) witnesses the equivalence $F_{I,B}(id_X) \equiv id_{F_{I,B}(X)}$. The relation

$$\{(t, (t_l, t_r)) \mid \forall i, b.\ t(i)(b) = (t_l(i)(b), t_r(i)(b))\}$$

witnesses the equivalence $F_{I,B}(k; \ell) \equiv F_{I,B}(k); F_{I,B}(\ell)$. □

For the case of lists, this mapping lens coincides with the retentive map that we obtained from the iterator in §2.6. In general, two pieces of data synchronized by one of these mapping lenses will have exactly the same shape; any shape change to one of the sides will be precisely mirrored in the other side. For example, the tree version

of this lens will transport the deletion of a node by deleting the node in the same position on the other side. We believe it should also be possible to define a forgetful version where the complement is just $F_{I,B}(\ell.C)$.

The notion of *combinatorial species* provides an alternative to the container framework. One of their attractions is that there are species corresponding to containers whose $B(i) \to X$ family is quotiented by some equivalence relation; we can obtain multisets in this way, for example. However, we have not explored this generalization in the case of lenses, because it is then not clear how to match up positions.

2.8 Asymmetric Lenses as Symmetric Lenses

The final step in our investigation is to formalize the connection between symmetric lenses and the more familiar asymmetric ones, and to show how known constructions on asymmetric lenses correspond to the constructions we have considered.

Write $X \overset{a}{\leftrightarrow} Y$ for the set of asymmetric lenses from X to Y (using the first presentation of asymmetric lenses from §2.1, with *get*, *put*, and *create* components).

2.8.1 Definition [Symmetrization]: Every asymmetric lens can be embedded in a symmetric one.

$$\frac{\ell \in X \overset{a}{\leftrightarrow} Y}{\ell^{sym} \in X \leftrightarrow Y}$$

$$\begin{array}{lcl} C & = & \{f \in Y \to X \mid \forall y \in Y.\ \ell.get(f(y)) = y\} \\ missing & = & \ell.create \\ putr(x, f) & = & (\ell.get(x), f_x) \\ putl(y, f) & = & \mathsf{let}\ x = f(y)\ \mathsf{in}\ (x, f_x) \end{array}$$

(Here, $f_x(y)$ means $\ell.put(y, x)$.) Viewing X as the source of an asymmetric lens (and therefore as having "more information" than Y), we can understand the definition of the complement here as being a value from X stored as a closure over that value. The presentation is complicated slightly by the need to accommodate the situation where a complete X does not yet exist—i.e. when defining *missing*—in which case we can use *create* to fabricate an X value out of a Y value if necessary.

Proof of well-formedness: The CreateGet law guarantees that $\ell.create \in C$ and the PutGet law guarantees that $f_x \in C$ for all $x \in X$, so we need merely check the round-trip laws.

PUTRL:

$$\begin{array}{rcl} putl(putr(x,c)) & = & putl(\ell.get(x), f_x) \\ & = & \mathsf{let}\ x' = f_x(\ell.get(x))\ \mathsf{in}\ (x', f_{x'}) \\ & = & \mathsf{let}\ x' = \ell.put(\ell.get(x), x)\ \mathsf{in}\ (x', f_{x'}) \\ & = & (x, f_x) \end{array}$$

PUTLR:

$$\begin{array}{rcl} putr(putl(y,f)) & = & putr(\mathsf{let}\ x = f(y)\ \mathsf{in}\ (x, f_x)) \\ & = & putr(f(y), f_{f(y)}) \\ & = & (\ell.get(f(y)), f_{f(y)}) \\ & = & (y, f_{f(y)}) \end{array}$$

□

2.8.2 Definition [Asymmetric lenses]: Here are several useful asymmetric lenses (based on string lenses from [8]).

$$copy_X \in X \overset{a}{\leftrightarrow} X$$

$$\begin{array}{rcl} get(x) & = & x \\ put(x,x') & = & x \\ create(x) & = & x \end{array}$$

$$\frac{k \in X \overset{a}{\leftrightarrow} Y \qquad \ell \in Y \overset{a}{\leftrightarrow} Z}{k;\ell \in X \overset{a}{\leftrightarrow} Z}$$

$$\begin{array}{rcl} get(x) & = & \ell.get(k.get(x)) \\ put(z,x) & = & k.put(\ell.put(z, k.get(x)), x) \\ create(z) & = & k.create(\ell.create(z)) \end{array}$$

$$\frac{v \in X}{aconst_v \in X \overset{a}{\leftrightarrow} Unit}$$

$$\begin{array}{lcl} get(x) & = & () \\ put((), x) & = & x \\ create(()) & = & v \end{array}$$

$$\frac{k \in X \overset{a}{\leftrightarrow} Y \qquad \ell \in Z \overset{a}{\leftrightarrow} W}{k \cdot \ell \in X \times Z \overset{a}{\leftrightarrow} Y \times W}$$

$$\begin{array}{lcl} get(x, z) & = & (k.get(x), \ell.get(z)) \\ put((y, w), (x, z)) & = & (k.put(y, x), \ell.put(w, z)) \\ create((y, w)) & = & (k.create(y), \ell.create(w)) \end{array}$$

$$\frac{k \in X \overset{a}{\leftrightarrow} Y \qquad \ell \in Z \overset{a}{\leftrightarrow} W}{k | \ell \in X + Z \overset{a}{\leftrightarrow} Y \cup W}$$

$$\begin{array}{lcl} get(\mathsf{inl}\ x) & = & k.get(x) \\ get(\mathsf{inr}\ z) & = & \ell.get(z) \\ put(yw, \mathsf{inl}\ x) & = & \begin{cases} \mathsf{inl}\ k.put(yw, x) & yw \in Y \\ \mathsf{inr}\ \ell.create(yw) & yw \in W \setminus Y \end{cases} \\ put(yw, \mathsf{inr}\ z) & = & \begin{cases} \mathsf{inr}\ \ell.put(yw, z) & yw \in W \\ \mathsf{inl}\ k.create(yw) & yw \in Y \setminus W \end{cases} \\ create(yw) & = & \begin{cases} \mathsf{inl}\ k.create(yw) & yw \in Y \\ \mathsf{inr}\ \ell.create(yw) & yw \in W \setminus Y \end{cases} \end{array}$$

$$\frac{\ell \in X \overset{a}{\leftrightarrow} Y}{\ell^\star \in X^\star \overset{a}{\leftrightarrow} Y^\star}$$

$$\begin{array}{rcl}
get(\langle x_1, \ldots, x_n \rangle) & = & \langle \ell.get(x_1), \ldots, \ell.get(x_n) \rangle \\
put(\langle y_1, \ldots, y_m \rangle, \langle x_1, \ldots, x_n \rangle) & & \\
& = & \langle x'_1, \ldots, x'_m \rangle \\
\text{where } x'_i & = & \begin{cases} \ell.put(y_i, x_i) & i \leq \min(m, n) \\ \ell.create(y_i) & n+1 \leq i \end{cases} \\
create(\langle y_1, \ldots, y_n \rangle) & = & \langle \ell.create(y_1), \ldots, \ell.create(y_n) \rangle
\end{array}$$

2.8.3 Theorem: The symmetric embeddings of these lenses correspond nicely to definitions from earlier in this paper:

$$copy_X^{sym} \equiv id_X \tag{2.8.1}$$
$$(k; \ell)^{sym} \equiv k^{sym}; \ell^{sym} \tag{2.8.2}$$
$$aconst_x^{sym} \equiv term_x \tag{2.8.3}$$
$$(k \cdot \ell)^{sym} \equiv k^{sym} \otimes \ell^{sym} \tag{2.8.4}$$
$$(k|\ell)^{sym} \equiv (k^{sym} \oplus^f \ell^{sym}); union \tag{2.8.5}$$
$$(\ell^\star)^{sym} \equiv \mathsf{map}^f(\ell^{sym}) \tag{2.8.6}$$

The first two show that $(-)^{sym}$ is a functor.

Proof: Throughout the proofs, we will use a to refer to the left-hand side of the equivalence, and b to refer to the right-hand side.

1. Defining f to be the identity function $f(x) = x$, the singleton relation $f\ R\ ()$ witnesses the equivalence. Since $a.missing(x) = x$, we have $a.missing\ R\ b.missing$. Furthermore:

$$\begin{array}{rcl}
a.putr(x, f) & = & (x, x' \mapsto copy_X.put(x', x)) \\
& = & (x, x' \mapsto x') \\
& = & (x, f) \\
b.putr(x, ()) & = & (x, ()) \\
a.putl(x, f) & = & (f(x), x' \mapsto copy_X.put(x', x)) \\
& = & (x, f) \\
b.putl(x, ()) & = & (x, ())
\end{array}$$

This establishes that $a.putr \sim_R b.putr$ and that $a.putl \sim_R b.putl$.

2. The relation

$$R = \{(f_{k\ell}, (f_k, f_\ell)) \mid f_{k\ell} = f_\ell; f_k\}$$

witnesses the equivalence. The fact that $a.missing\ R\ b.missing$ is immediate from the definitions.

Now, to show that $a.putr \sim_R b.putr$, suppose $f_{k\ell}\ R\ (f_k, f_\ell)$. We first compute $a.putr(x, f_{k\ell})$.

$$\begin{array}{rcl} a.putr(x, f_{k\ell}) & = & ((k;\ell).get(x), z \mapsto (k;\ell).put(z, x)) \\ & = & (\ell.get(k.get(x)), \\ & & z \mapsto k.put(\ell.put(z, k.get(x)), x)) \\ & = & (x_a, f'_{k\ell}) \end{array}$$

And now $b.putr(x, (f_k, f_\ell))$:

$$\begin{array}{rcl} k^{sym}.putr(x, f_k) & = & (k.get(x), y \mapsto k.put(y, x)) \\ \ell^{sym}.putr(k.get(x), f_\ell) & = & (\ell.get(k.get(x)), \\ & & z \mapsto \ell.put(z, k.get(x))) \\ b.putr(x, (f_k, f_\ell)) & = & (x_b, (f'_k, f'_\ell)) \end{array}$$

It's now clear that

$$\begin{array}{rcl} f'_k(f'_\ell(z)) & = & f'_k(\ell.put(z, k.get(x))) \\ & = & k.put(\ell.put(z, k.get(x)), x) \\ & = & f'_{k\ell}(z) \end{array}$$

and that $x_a = x_b$, so $a.putr \sim_R b.putr$.

Finally, to show that $a.putl \sim_R b.putl$, suppose again that $f_{k\ell}\ R\ (f_k, f_\ell)$.

$$\begin{array}{rcl} a.putl(z, f_{k\ell}) & = & \mathsf{let}\ x = f_{k\ell}(z)\ \mathsf{in} \\ & & (x, z' \mapsto (k;\ell).put(z', x)) \\ & = & \mathsf{let}\ x = f_{k\ell}(z)\ \mathsf{in} \\ & & (x, z' \mapsto k.put(\ell.put(z', k.get(x)), x)) \end{array}$$

Similarly,

$$\begin{array}{rcl} \ell^{sym}.putl(z, f_\ell) & = & \mathsf{let}\ y = f_\ell(z)\ \mathsf{in} \\ & & (y, z' \mapsto \ell.put(z', y)) \\ k^{sym}.putl(f_\ell(z), f_k) & = & \mathsf{let}\ x = f_k(f_\ell(z))\ \mathsf{in} \\ & & (x, y' \mapsto k.put(y', x)) \\ b.putl(z, (f_k, f_\ell)) & = & (f_k(f_\ell(z)), \\ & & (y' \mapsto k.put(y', f_k(f_\ell(z))), \\ & & z' \mapsto \ell.put(z', f_\ell(z)))) \end{array}$$

Now, we want to show that the first parts of the outputs are equal, that is, that $f_{kl}(z) = f_k(f_\ell(z))$, which is immediate from $f_{kl}\ R\ (f_k, f_\ell)$, and that the second parts of the outputs are related:

$$\begin{array}{rcl} f'_k(f'_\ell(z')) & = & f'_k(\ell.put(z, f_\ell(z))) \\ & = & k.put(\ell.put(z, f_\ell(z)), f_k(f_\ell(z))) \end{array}$$

Observing that

$$\begin{array}{rll} k.get(f_k(f_\ell(z))) = f_\ell(z) & & \text{because } f_k \in k^{sym}.C \\ f_k(f_\ell(z)) = f_{k\ell}(z) & & \text{because } f_{k\ell}\ R\ (f_k, f_\ell), \end{array}$$

that last line becomes

$$\begin{array}{rcl} f'_k(f'_\ell(z')) & = & k.put(\ell.put(z, k.get(f_{k\ell}(z))), f_{k\ell}(z)) \\ & = & f'_{k\ell}(z') \end{array}$$

so the second parts of the outputs are related after all, and $a.putl \sim_R b.putl$.

3. The relation

$$R = \{(() \mapsto c, c) \mid c \in X\}$$

witnesses the equivalence. Since $a.missing = () \mapsto x$ and $b.missing = x$, we see $a.missing\ R\ b.missing$.

To show that $a.putr \sim_R b.putr$, choose arbitrary $x, c \in X$ and define $f_c(()) = c$:

$$\begin{array}{c} a.putr(x, f_c) = ((), () \mapsto x) \\ b.putr(x, c) = ((), x) \end{array}$$

These clearly satisfy $() = ()$ and $(() \mapsto x)\ R\ x$, so we can conclude that $a.putr \sim_R b.putr$.

To show that $a.putl \sim_R b.putl$, choose arbitrary $c \in X$ and define $f_c(()) = c$ as

before. Then:

$$\begin{aligned} a.putl((), f_c) &= (f_c(()), u \mapsto aconst_x.put(u, f_c(()))) \\ &= (c, u \mapsto c) \\ &= (c, () \mapsto c) \\ b.putl((), c) &= (c, c) \end{aligned}$$

These again clearly satisfy $c = c$ and $(() \mapsto c)\ R\ c$, so $b.putl \sim_R b.putl$.

4. The relation

$$R = \{(f_{k\ell}, (f_k, f_\ell)) \mid \forall y, w. f_{k\ell}(y, w) = (f_k(y), f_\ell(w))\}$$

witnesses the equivalence. We can compute

$$\begin{aligned} a.missing &= (y, w) \mapsto (k.create(y), \ell.create(w)) \\ b.missing &= (y \mapsto k.create(y), w \mapsto \ell.create(w)), \end{aligned}$$

so clearly $a.missing\ R\ b.missing$.

Let us show that $a.putr \sim_R b.putr$. Choose $(x, z) \in X \times Z$ and arbitrary $f_{k\ell}, f_k, f_\ell$ (we will not need the assumption that $f_{k\ell}\ R\ (f_k, f_\ell)$). Then:

$$\begin{aligned} a.putr((x, z), f_{k\ell}) &= ((k.get(x), \ell.get(z)), \\ &\qquad (y, w) \mapsto (k.put(y, x), \ell.put(w, z))) \\ b.putr((x, z), (f_k, f_\ell)) &= ((k.get(x), \ell.get(z)), \\ &\qquad (y \mapsto k.put(y, x), w \mapsto \ell.put(w, z)) \end{aligned}$$

It's clear that the first elements of these tuples are equal, and the second elements are just as clearly related by R, so it is indeed true that $a.putr \sim_R b.putr$.

Similarly, choose $(y, w) \in Y \times W$ and suppose $f_{k\ell}\ R\ (f_k, f_\ell)$ – which in particular means that $f_{k\ell}(y, w) = (f_k(y), f_\ell(w))$. Then we can define a few things:

$$\begin{aligned} (v_a, f_a) &= a.putl((y, w), f_{k\ell}) \\ &= \mathsf{let}\ (x, z) = f_{k\ell}(y, w)\ \mathsf{in} \\ &\quad ((x, z), (y', w') \mapsto (k.put(y', x), \ell.put(w', z)) \\ &= \mathsf{let}\ (x, z) = (f_k(y), f_\ell(w))\ \mathsf{in} \\ &\quad ((x, z), (y', w') \mapsto (k.put(y', x), \ell.put(w', z)) \\ &= ((f_k(y), f_\ell(w)), \\ &\quad (y', w') \mapsto (k.put(y', f_k(y)), \ell.put(w', f_\ell(w)))) \end{aligned}$$

$$\begin{aligned}(v_b, f_b) &= b.putl((y, w), (f_k, f_\ell)) \\ &= \mathsf{let}\ x = f_k(y)\ \mathsf{in} \\ &\quad \mathsf{let}\ z = f_\ell(w)\ \mathsf{in} \\ &\quad ((x, z), (y' \mapsto k.put(y', x), w' \mapsto \ell.put(w', z))) \\ &= ((f_k(y), f_\ell(w)), \\ &\quad (y' \mapsto k.put(y', f_k(y)), w' \mapsto \ell.put(w', f_\ell(w))))\end{aligned}$$

So $v_a = v_b$ and $f_a\ R\ f_b$ – that is, $a.putl \sim_R b.putl$.

5. Suppose $k \in X \overset{a}{\leftrightarrow} Y$ and $\ell \in Z \overset{a}{\leftrightarrow} W$. Define the following functions:

$$g \in ((Y \to X) + (W \to Z)) \times (Y \cup W) \to X + Z$$

$$g(\mathsf{inl}\ f_k, yw) = \begin{cases} \mathsf{inl}\ f_k(yw) & yw \in Y \\ \mathsf{inr}\ \ell.create(yw) & yw \in W \setminus Y \end{cases}$$
$$g(\mathsf{inr}\ f_\ell, yw) = \begin{cases} \mathsf{inr}\ f_\ell(yw) & yw \in W \\ \mathsf{inl}\ k.create(yw) & yw \in Y \setminus W \end{cases}$$

$$\mathsf{tag} \in (Y \to X) + (W \to Z) \to \mathit{Bool}$$

$$\begin{aligned}\mathsf{tag}(\mathsf{inl}\ f_k) &= \mathsf{false} \\ \mathsf{tag}(\mathsf{inr}\ f_\ell) &= \mathsf{true}\end{aligned}$$

Then we can define the relation

$$R = \{(g(f), (f, \mathsf{tag}(f))) \mid f \in (k^{sym} \oplus^f \ell^{sym}).C\}.$$

It is tedious but straightforward to verify that this witnesses the equivalence.

6. $(\ell^*)^{sym}.C$ comprises functions $f : Y^* \to X^*$ such that whenever $f([y_1, \ldots, y_n]) = [x_1, \ldots, x_m]$ we can conclude $m = n$ and $\ell.get(x_i) = y_i$.

   The complement $\mathsf{map}^f(\ell^{sym}).C$ on the other hand comprises lists of functions $[f_1, \ldots, f_n]$ where $f_i : Y \to X$ and $\ell.get(f_i(y)) = y$. Relate two such complements f and $[f_1, \ldots, f_n]$ if $f([y_1, \ldots, y_m]) = [x_1, \ldots, x_m]$ implies $x_i = f_i(y_i)$ when $i \leq n$ and $x_i = \ell.create(y_i)$ otherwise.

   Clearly, the two "missings" are thus related and it is also easy to see that *putr* is respected. As for the *putl* direction consider that f and $[f_1, \ldots, f_n]$ are related and that $ys = [y_1, \ldots, y_m]$ is do be *putl*-ed. Let $[x_1, \ldots, x_k]$ be the result in the $(f^*)^{sym}$ direction. It follows $k = m$ and $[x_1, \ldots, x_m] = f([y_1, \ldots, y_m])$. If $[x'_1, \ldots, x'_m]$ is the result in the $\mathsf{map}^f(\ell^{sym})$ direction then $x'_i = f_i(y_i)$ if $i \leq n$ and $x'_i = \ell.create(y_i)$ otherwise. Now $x_i = x'_i$ follows by relatedness.

The new $(\ell^*)^{sym}$ complement then is $\lambda ys.(\ell^*).put(ys, xs)$. The new $\mathsf{map}^f(\ell^{sym})$ complement is $[g_1, \ldots, g_m]$ where $g_i(y) = \ell.put(x_i, y)$. These are clearly related again. □

We suspect that there might be an asymmetric *fold* construction similar to our iteration lens above satisfying an equivalence like

$$fold(\ell)^{sym} \equiv It(\ell^{sym}),$$

but have not explored this carefully.

The $(-)^{sym}$ functor is not *full*—that is, there are some symmetric lenses which are not the image of any asymmetric lens. Injection lenses, for example, have no analog in the category of asymmetric lenses, nor do either of the example lenses presented in the introduction. However, we *can* characterize symmetric lenses in terms of asymmetric ones in a slightly more elaborate way.

2.8.4 Theorem [Lenses are spans]: Given any arrow ℓ of LENS, there are asymmetric lenses k_1, k_2 such that

$$(k_1^{sym})^{op}; k_2^{sym} \equiv \ell.$$

This suggests that the category LENS could be constructed from spans in ALENS. A full account of the machinery necessary to realize this approach is given by Johnson and Rosebrugh [26]. It is quite involved for two reasons: first, composition of spans is typically given via a pullback construction, but pullbacks in the appropriate category do not always exist, and second, one must develop a span-based analog for our lens equivalence to retain associativity of composition.

To see this, we need to know how to "asymmetrize" a symmetric lens.

2.8.5 Definition [Asymmetrization]: We can view a symmetric lens as a pair of asymmetric lenses joined "tail to tail" whose common domain is consistent triples. For any lens $\ell \in X \leftrightarrow Y$, define

$$S_\ell = \{(x, y, c) \in X \times Y \times \ell.C \mid \ell.putr(x, c) = (y, c)\}.$$

Now define:

$$\frac{\ell \in X \leftrightarrow Y}{\ell_r^{asym} \in S_\ell \overset{a}{\leftrightarrow} X}$$

$$\begin{array}{lcl} get((x,y,c)) & = & x \\ put(x',(x,y,c)) & = & \mathsf{let}\ (y',c') = \ell.putr(x',c) \\ & & \mathsf{in}\ (x',y',c') \\ create(x) & = & \mathsf{let}\ (y,c) = \ell.putr(x,\ell.missing) \\ & & \mathsf{in}\ (x,y,c) \end{array}$$

$$\frac{\ell \in X \leftrightarrow Y}{\ell_l^{asym} \in S_\ell \overset{a}{\leftrightarrow} Y}$$

$$\begin{array}{lcl} get((x,y,c)) & = & y \\ put(y',(x,y,c)) & = & \mathsf{let}\ (x',c') = \ell.putl(y',c) \\ & & \mathsf{in}\ (x',y',c') \\ create(y) & = & \mathsf{let}\ (x,c) = \ell.putl(y,\ell.missing) \\ & & \mathsf{in}\ (x,y,c) \end{array}$$

Proof of well-formedness: We show only that ℓ_r^{asym} is well-formed; the proof for ℓ_l^{asym} is similar.
GetPut:

$$\begin{array}{lcl} put(get((x,y,c)),(x,y,c)) & = & put(x,(x,y,c)) \\ & = & \mathsf{let}\ (y',c') = \ell.putr(x,c) \\ & & \mathsf{in}\ (x,y',c') \\ & = & (x,y,c) \end{array}$$

The final equality is justified because (x,y,c) is a consistent triple.
PutGet:

$$\begin{array}{lcl} get(put(x',(x,y,c))) & = & \mathsf{let}\ (y',c') = \ell.putr(x',c) \\ & & \mathsf{in}\ get((x',y',c')) \\ & = & x' \end{array}$$

CreateGet:

$$\begin{aligned} get(create(x)) &= \mathsf{let}\ (y,c) = \ell.putr(x, \ell.missing) \\ &\quad\ \mathsf{in}\ get((x,y,c)) \\ &= x \end{aligned}$$

In addition to the three round-trip laws, we must show that *put* and *create* yield consistent triples. But this is clear: the PutR2 law is exactly what we need. □

Proof of 2.8.4: Given arrow $[\ell]$, choose $k_1 = \ell_r^{asym}$ and $k_2 = \ell_l^{asym}$. Writing ℓ_r for $((\ell_r^{asym})^{sym})^{op}$ and ℓ_l for $(\ell_l^{asym})^{sym}$, we then need to show that $\ell_r; \ell_l \equiv \ell$. Define two functions:

$$\begin{aligned} f_c(x) &= \mathsf{let}\ (y,c') = \ell.putr(x,c)\ \mathsf{in}\ (x,y,c') \\ g_c(y) &= \mathsf{let}\ (x,c') = \ell.putl(y,c)\ \mathsf{in}\ (x,y,c') \end{aligned}$$

Then the relation $R = \{((f_c, g_c), c) \mid c \in C\}$ witnesses the equivalence. We can check the definitions to discover that

$$\ell_r.missing = \ell_r^{asym}.create = f_{\ell.missing}$$

$$\ell_l.missing = \ell_l^{asym}.create = g_{\ell.missing}$$

and hence that $(\ell_r; \ell_l).missing\ R\ \ell.missing$.

We also need to show that $(\ell_r; \ell_l).putr$ and $\ell.putr$ are well-behaved with respect to R. Suppose $\ell.putr(x,c) = (y,c')$; then we need to show that

$$(\ell_r; \ell_l).putr(x, (f_c, g_c)) = (y, (f_{c'}, g_{c'})).$$

First we compute $\ell_r.putr(x, f_c)$:

$$\begin{aligned} \ell_r.putr(x, f_c) &= ((\ell_r^{asym})^{sym})^{op}.putr(x, f_c) \\ &= (\ell_r^{asym})^{sym}.putl(x, f_c) \\ &= \mathsf{let}\ t = f_c(x)\ \mathsf{in}\ (t, x' \mapsto \ell_r^{asym}.put(x', t)) \\ &= \mathsf{let}\ (y,c') = \ell.putr(x,c)\ \mathsf{in} \\ &\quad\ ((x,y,c'), x' \mapsto \ell_r^{asym}.put(x', (x,y,c'))) \\ &= ((x,y,c'), x' \mapsto \ell_r^{asym}.put(x', (x,y,c'))) \\ &= ((x,y,c'), f_{c'}) \end{aligned}$$

We then compute $\ell_l.putr((x, y, c'), g_c)$:

$$\begin{aligned}
\ell_l.putr((x, y, c'), g_c) &= (\ell_l^{asym})^{sym}.putr((x, y, c'), g_c) \\
&= (\ell_l^{asym}.get((x, y, c')), \\
&\quad\ y' \mapsto \ell_l^{asym}.put(y', (x, y, c'))) \\
&= (y, y' \mapsto \ell_l^{asym}.put(y', (x, y, c'))) \\
&= (y, g_{c'})
\end{aligned}$$

We conclude from this that $(\ell_r; \ell_l).putr(x, (f_c, g_c)) = (y, (f_{c'}, g_{c'}))$ as desired.

The argument that $(\ell_r; \ell_l).putl$ and $\ell.putl$ are well-behaved with respect to R is almost identical. □

2.9 Conclusion

We have proposed the first notion of symmetric bidirectional transformations that supports composition. Composability opens up the study of symmetric bidirectional transformations from a category-theoretic perspective. The category of symmetric lenses is self-dual and has the category of bijections and that of asymmetric lenses each as full subcategories. We have surveyed the structure of this category and found it to admit tensor product structures that are the Cartesian product and disjoint union on objects. We have also investigated data types both inductively and as "containers" and found the category of symmetric lenses to support powerful mapping and folding constructs. In the next chapter, we will extend this approach to address performance—significantly reducing the amount of information a lens must process—and alignment—giving precise details about the correspondence between old and new copies of a complex repository.