Electronic Communications of the EASST
Volume 49 (2012)

Proceedings of the First International Workshop on Bidirectional Transformations (BX 2012)

Lens put-put laws: monotonic and mixed

Michael Johnson and Robert Rosebrugh

13 pages

Guest Editors: Frank Hermann, Janis Voigtländer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Lens put-put laws: monotonic and mixed

Michael Johnson[1] **and Robert Rosebrugh**[2]

[1] http://www.cs.mq.edu.au/~mike
Department of Computing
Macquarie University, Australia

[2] http://www.mta.ca/~rrosebru
Department of Mathematics and Computer Science
Mount Allison University, Canada

Abstract: Many authors have argued, for good reasons, that in a range of applications the lens put-put law is too strong. On the other hand, the present authors have shown that very well behaved lenses, which do satisfy the put-put law by definition, are algebras for a certain monad, and that this viewpoint admits fruitful generalisations of the lens concept to a variety of base categories. In the algebra approach to lenses, the put-put law corresponds to the associativity axiom, and so is fundamentally important.

Thus we have a dilemma. The put-put law seems inappropriate for many applications, but is fundamental to the mathematical development that can support an extended range of applications.

In this paper we resolve this dilemma. We outline monotonic put-put laws and introduce a new mixed put-put law that appears to be immune to many of the objections to the classical put-put law, and still supports a very satisfactory mathematical foundation.

Keywords: Lens, view update, algebra, monad, span

1 Introduction

Lenses are an appropriate formalism for a wide range of bidirectional transformations. One of the laws that a lens is sometimes required to satisfy is called the put-put law.

The put-put law as normally stated (see below) is widely seen as too strong as it is simply not satisfied in a variety of important applications.

Separately from this concern, and following from a mathematical analysis of very well behaved lenses, the put-put law is indeed, even theoretically, too strong whenever the category of view states is non-trivial (not just a set of states). The put-put law as normally stated ignores the extra structure and can be seen to be an unreasonable requirement.

In this paper we describe put operations that respect the extra structure and then monotonic put-put laws — those in which successive puts are, loosely speaking, both along deletes, or both along inserts, and note that such laws do not suffer the criticisms of the classical put-put law, whether practical or theoretical. Furthermore these new put-put laws provide a counterexample to the mistaken belief that the presence of put-put laws implies constant complement updating.

Of course, of much greater interest than monotonic put-put laws are mixed put-put laws that treat together both inserts and deletes. The main contribution of this paper is the development and analysis of a new mixed put-put law that is very general, is indeed weaker than classical put-put, does not seem to be subject to the extant criticisms of the classical put-put law, and is sufficient to support the monad based mathematical treatment of generalised lenses.

2 Background

Well-behaved lenses

Lenses ([BVP06], [FGM07], and many other sources) are a particular kind of bidirectional transformation that properly captures the following asymmetric situation that arises in a wide range of applications (although it is worth noting that there has also been recent work on developing a symmetric notion of lens [HPW11]). In one direction, (one half of) the bidirectional transformation presents a "view" of a global state, usually losing information. That transformation is usually called a *get*, frequently abbreviated g. In the reverse direction, the transformation can't reasonably be expected to regenerate the missing information, so the *put* transformation, p, will take as input both a view state and a global state and generate a new global state. In symbols, if S is the set of global states, and V is the set of view states, then

$$g : S \to V, \qquad p : V \times S \to S.$$

Usually the two directions, g and p are expected to be mutually inverse in the sense that

$$g(p(v,s)) = v,$$

(the put-get law) and

$$p(g(s),s) = s,$$

(the get-put law) for all $s \in S$ and $v \in V$. Such a lens is called *well-behaved*.

In some cases the functions, especially the put function, might not be total (fully defined), and the above equations need to be adjusted to take into account the domain of definition. We will say no more about totality here since the extra structure introduced in Section 3 is frequently what is required to restore totality.

Put-put laws

A *very well-behaved lens* is a well-behaved lens that also satisfies the put-put law

$$p(v,p(v',s)) = p(v,s)$$

for all $v,v' \in V$ and $s \in S$. In other words, in a very well-behaved lens iterated puts can be achieved by simply putting the *last* view state.

The three laws of very well-behaved lenses have arisen in a wide range of circumstances from at least the work of Oles [Ole82], [Ole86], through Hoffman and Pierce [HP96], to the range of applications in the modern Bx community.

Many times over the years, and more recently with the very detailed study of lenses and their laws instigated by the Bx community, the put-put law has been seen as problematic. Indeed, there are in applications many reasonable examples that do not seem to satisfy the put-put law. See [DXC10] Section 2.4 for a recent written out example. Perhaps it is not surprising that the put-put law turns out to be too strong for many applications. In the put-put law above, any view state v' whatsoever can appear on the left hand side. Indeed, we have chosen to call that state v' because, although it is the first state to be put, in the context of the put-put law it is just an intermediate state and, according to the law, of no importance. Except in relatively trivial cases like constant complement updating [BS81] it seems a lot to expect that extreme choices of v' should not affect the overall result. Indeed, it has been shown that if a well-behaved lens also satisfies the put-put law then the lens is merely a representation of a constant complement updating strategy. (This result can easily be misremembered as "If a lens satisfies a put-put law then its updating is constant complement" — an important point to be taken from this paper is that such a statement may not be true for generalised put-put laws.)

This is a good time to point out that classical lenses, as defined so far, are based on *sets* of states S and V. More usually states are represented by a state transition graph or even a category of states with, as morphisms, state transitions. Once transitions are explicitly included, v' is no longer arbitrary (since it must at least be reachable from $g(s)$), and there are new put-put laws that take the transitions into account. This paper arose from a study of such generalised put-put laws.

Lenses as algebras

In this section we will say simply "lens" in place of "very well-behaved lens".

Recent work of the authors, and their colleague Wood, [JRW10], has sought to put the study of lenses on a concrete category theoretic (see for example, Pierce [Pie91]) foundation. The main result of that work is that the category of very well-behaved lenses for a fixed set of view states V is equivalent to the category of algebras for a well-known monad M on the slice category $\mathbf{set}/V$. Any particular lens will correspond to an algebra. The carrier of that algebra, being an object of $\mathbf{set}/V$, will be a function with codomain V — the get function, g. The action for that algebra, also known as the structure map, will be the put function, p.

For readers who would like to reconstruct the details, the monad M takes an object $g : C \to V$ of $\mathbf{set}/V$ to the product projection $\pi_0 : V \times C \to V$. A structure map is then an arrow $Mg \to g$ in $\mathbf{set}/V$, that is a function $p : V \times C \to C$ that commutes with g and π_0. The commutativity just mentioned is an instance of the put-get law. The two algebra axioms correspond to the get-put law (the identity axiom) and the put-put law (the associativity axiom).

We will not revisit in detail the equivalence of the category of lenses and the category of algebras for M here, but that result does have two noteworthy implications.

First, the put-put law can be seen in a new context. It is not some unreasonable law that may have arisen from some special applications and should be discarded immediately if it seems not to apply in a new application. Rather it corresponds to the associativity law for the algebra — it is mathematically fundamental and has important implications for the tractability of the mathematical treatment.

Second, the treatment of lenses as algebras admits immediate generalisation by replacing

set$/V$ by other categories including for example **ord**$/V$ and **cat**$/V$ (the construction of the monad M on arbitrary $\mathbf{C}/V$ depends only on the category $\mathbf{C}$ having products). In the former case we have generalised lenses that incorporate ordered sets, rather than mere sets, of states, and recover and simplify the ordered view update work of Hegner [Heg04]. In the latter case we are able to deal with arbitrary categories of states (which may include distinguishable parallel pairs of transitions between the same two states).

Thus we can treat coherently a wide range of generalised lenses that take account of the fact that sets of states almost always have extra structure (graphs or categories of transitions).

3 Monotonic put-put

Generalising lenses to algebras for the monad M on **cat**$/V$ stays quite close to the classical notion of lens, apart of course from treating the collections of states V and S as categories rather than mere sets. In particular, the structure maps p for algebras, corresponding to the put operations for lenses, still take as input a pair of states (v,s).

Let's review what's really happening here. We start with a global state s, find its view $g(s)$, modify that view along a transition say $t : g(s) \to v$, and then complete the "view update" by calculating $p(v,s)$ to find a new state in S. Notice that the transition t has been ignored in calculating the view update. We are failing to treat transitions as first class citizens.

Very recent work [JRW12] remedies this failure by replacing the monad M by a comma category monad R. In that paper the objects of the domain of Rg are s indexed arrows $t : g(s) \to v$, and so the input for a structure map is no longer a pair (v,s), but instead an arrow $t : g(s) \to v$, indexed by an object $s \in S$. Transitions are promoted to first class citizens.

Again we don't need to study the details of the monad R for the purposes of this paper. Rather, we take R as given from [JRW12]. Nevertheless, to aid readers who might want to refer to that paper we note that R is there called $(-, 1_V)$ because, for an object $g : S \to V$ of **cat**$/V$, the domain of Rg is the comma category $(g, 1_V)$. Furthermore, Rg itself is the projection functor $(g, 1_V) \to V$ from the comma category to V. (So, for g an object of **cat**$/V$, Mg is a projection from a product built using the domain of g, and Rg is a projection from a comma category built using the domain of g.) Perhaps it is also worth recalling that objects of the comma category $(g, 1_V)$ are triples $(s, t : g(s) \to v, v)$ — that is why we said above an arrow t *indexed by* s. Even if g is not injective on objects, s remains known.

What we seek to do here is explore the associativity law for an algebra for R, because the associativity law is the rule that corresponds to the put-put law. Let's see what it says.

Given a state $s \in S$, and two state transitions in V, $t' : g(s) \to v'$ and $t : v' \to v$, associativity amounts to the equation

$$p(t : g(p(t' : g(s) \to v')) \to v) = p(tt' : g(s) \to v)$$

In this equation $p(t' : g(s) \to v')$ is a state, s' say, with the property that the equation $g(s') = v'$ is satisfied because of the put-get law.

The displayed equation is called a *monotonic put-put law*. Like the classical put-put it says that iterated puts can be achieved by a single put. In the classical case, that put uses the original global state s and the final view state v. In this new case the single put again uses the original

global state s and the final view state v, but the domain of a put must be a transition, and the transition $g(s) \to v$ is obtained by composing the two transitions t' and t. The new law is called a *monotonic* put-put law because, for the two transitions to be composable, they must both go in the same direction, in this case to the right from an image of g to some state $v \in V$. We call such transitions *R-transitions*. Finally, this "rightwardness" is the reason for using R as the name of the monad here.

There is another monotonic put-put law which is the associativity law for algebras for a monad L where L takes an object g in $\mathbf{cat}/V$ to another object in $\mathbf{cat}/V$ whose domain is the comma category $(1_V, g)$. This other put-put law deals with a pair of transitions $(v \to v', v' \to g(s))$ from the *left*, from V states into images of g. We call these *L-transitions*.

In traditional relational database examples, the arrows in the category of view states are chosen to point in the direction of *increased information*. Thus an arrow $t : a \to b$ is an insert — b has all the tuples of a, and perhaps more. A delete operation is really travelling backwards, leftwards, along such an arrow, from the state b to a state a with fewer tuples. Of course, this "travelling backwards", or "leftwards", might sound imprecise, but the monad L makes it completely precise and recovers all the information that might have been included in the state space by adding arrows $b \to a$.

We conjecture that the difficulties with put-put laws do not arise with monotonic put-put laws, even if both monotonic put-put laws are satisfied. Certainly the routine examples against put-put laws involve "mixed" operations — a delete followed by an insert for example (an L-transition followed by an R-transition), and the monotonic put-put laws do not say anything about such cases.

Furthermore, the monotonic put-put laws are satisfied in universal view updating (see [JR07]) and universal view updating is *more* general than constant complement updating. In other words, a generalised lens, in the form of an algebra for the monad R or L, or even both together, need not be a representation of a constant complement updating strategy despite satisfying monotonic put-put law(s).

4 A running example: Labelled graphs

The preceding section has illustrated a new kind of put-put law, one that depends upon taking successive puts in the same "direction" (both inserts or both deletes in the standard database view examples). These generalised very well-behaved lenses are not instances of constant complement updates and seem to avoid the usual objections to put-put laws.

In this section we illustrate this with an example based on labelled graphs. We will return to this example later in the paper to illustrate the mixed put-put law too.

Let S be the category whose objects are directed graphs with nodes labelled by elements of a fixed alphabet Ω_N and edges labelled by elements of a fixed alphabet Ω_E, and whose morphisms are label preserving graph inclusions. Let V be a similar category of directed graphs and graph inclusions, but in which the graphs have nodes labelled by elements of Ω_N and edges are unlabelled. Finally let $g : S \to V$ be the functor which "forgets" the labels on edges (so a view of a node- and edge-labelled graph is the same graph but without its edge labels).

Now we will construct two put operations p_R and p_L which will be structure maps for an R-

algebra and an L-algebra respectively. To do so we suppose that there is a distinguished element $e \in \Omega_E$ which we will call the *default edge label*.

The structure map p_R will send, as in the previous section, a transition $t' : g(s) \to v'$ to an object $p_R(t' : g(s) \to v')$ in S. In other words, given a node- and edge-labelled graph s, and a graph inclusion embedding $g(s)$ in a larger graph v', applying p_R will give a new node- and edge-labelled graph s' say. We know the shape of s'. Because of the put-get law, s' must have the same underlying graph as v'. Furthermore for the same reason s' must have the same node labelling as v'. For edge labelling s' can inherit the edge labels from s because t shows how edges of s, all be they unlabelled as edges of $g(s)$, correspond to some of the edges of v', and the remaining edges can be labelled with e, the default edge label.

In short, p_R will add the nodes and edges to s that t' added to $g(s)$, and label the new edges e.

Similarly, but even more easily, p_L will take a node- and edge-labelled graph s and a graph inclusion $t'' : v'' \to g(s)$. The transition t'' can be thought of as deleting some nodes and edges from $g(s)$ and p_L will delete the corresponding nodes and edges from s to obtain a new object of S, $s'' = p_L(t'' : v'' \to g(s))$.

It is easy to see that p_R and p_L so defined are the structure maps for an R- and L-algebra respectively — it is simply a matter of checking the get-put, put-get and monotonic put-put laws. (And we have included all those primes and double primes in the notation here so that the reader checking monotonic put-put laws can add for example a simple $t : v' \to v$ as in the previous section.)

The put operations p_R and p_L can be melded to form a single put operation p which works on any transition, leftward or rightward, but p does not satisfy the put-put law. Notice that starting at $g(s)$ and deleting an edge and then reinserting it returns us to $g(s)$, but applying p_L and then p_R will not return us to s but rather to s with the relevant edge relabelled as e.

The kind of update we have here is a generalised very well behaved lens – it satisfies put-get, get-put and both monotonic put-puts; it has a representation as an R-algebra and an L-algebra; but the updating is not constant complement; and as in many practical examples, the classical put-put law is not satisfied.

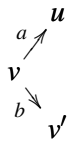
There are similar examples that are based on the important database problem of missing attribute information and view updating, but they require a little more overhead to describe. We are grateful to an anonymous referee for suggesting this simple and elegant example.

5 Mixed operations: Spans

We turn now to the question of whether there is a mixed put-put law that incorporates the two monotonic put-put laws, but also puts limitations on pairs of mixed operations (deletes followed by inserts for example, and vice versa) without being so strong as to collapse into constant complement updating, and to be inapplicable in common examples.

So we need to study mixed operations. We will take as a basic operation in V not a single

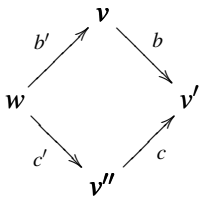
arrow of V, but rather a *span* of arrows in V:



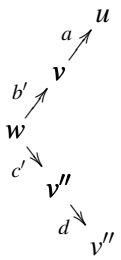
We think of this span as a mixed operation *from u to v'* (of course there is another mixed operation from v' to u via b and a, sometimes called the *reverse* of the span from u to v'). We will think of such spans as elementary mixed operations. These elementary mixed operations don't exclude from our consideration ordinary unmixed operations, arrows of V, since they can be treated as spans in which one "leg", a or b, is an identity arrow.

Assume now that V has pullbacks. Then adjacent pairs of spans

can be *composed* by first forming a pullback of b and c, w say, with projections b' and c',



and then composing the two monotonic legs ab' and dc' as shown



to obtain a single elementary mixed operation.

More generally, arbitrary mixed operations, that is arbitrary strings of arrows of V pointing in either direction, can be reduced to an elementary mixed operation by composing the monotonic operations to get a finite zig-zag, that is a finite string of successive composable spans, and then repeatedly composing adjacent pairs of spans. That reduces the zig-zag to a single elementary mixed operation.

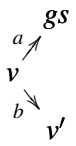
Perhaps we should remark that because pullbacks are defined up to isomorphism, the composition of mixed operations is only determined up to isomorphism. But in fact, this is not an issue at all here, as we will see in the next section.

6 Mixed put-put

Suppose through the remainder of this paper that we have a functor $g : S \to V$ that has L- and R-algebra structures p_L and p_R. In other words, g is the get function for a generalised lens that satisfies both monotonic put-put laws. To ease our notation we will write

$$p_L(a : v \to g(s)) = p_L(a, s)$$

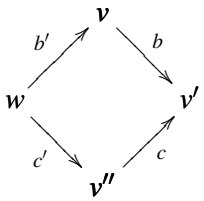
and similarly for p_R. Then there is a put for elementary mixed operations



which we will denote by $p(a,b,s)$ given by $p_R(b, p_L(a,s))$.

We propose the following mixed put-put law. Given composable spans

along with a pullback w, b', c' of b and c,



the **mixed put-put law** requires that

$$p(ab', dc', s) = p(c, d, p(a, b, s))$$

We revisit now our labelled graphs example introduced in Section 4 and illustrate the mixed put-put law with an example which satisfies mixed put-put but does not satisfy classical put-put.

Suppose in our notation above that s is the graph $f : A \to B$ (so s is the node- and edge-labelled graph whose single edge is labelled f). Then $g(s)$ is the node-labelled graph $A \to B$. Let v''' be

the same node labelled graph $A \to B$, let v and v'' be the node-labelled graph A, let v' be the node labelled graph $A \to C$ and let the transitions be the evident inclusions (a and d embed A in $A \to B$ and b and c embed A in $A \to C$). It follows that the pullback w is again the node-labelled graph A, and that b' and c' are both the identity graph morphism $A \to A$.

Notice that in this instance, the mixed put-put law stated above is satisfied (both sides evaluate to $e : A \to B$). Furthermore the classical put-put law is not in general satisfied ($g(s) = v'''$, and so a put that ignores transitions must take the update of $g(s)$ to v''' to s, that is to $f : A \to B$, but the iterated updates forget the edge label f when that edge is deleted and then presumably reinstate the edge with the default edge label $e : A \to B$).

In fact, as we will see in the next section, all instances of this example (p_R and p_L, and so p, applied to any graph s in S and all appropriate transitions a, b, c and d in V) satisfy the mixed put-put law. Here ends, for now, the example.

The mixed put-put law, along with the two monotonic put-put laws, provides a kind of associativity on our three put operations so that arbitrary mixed operations (as defined in Section 5) can be *put* in any order using any of p, p_L and p_R.

In particular,

Proposition 1 *Suppose that p satisfies the mixed put-put law, that w, b', c' is a pullback of b and c as above, that w', b'', c'' is another pullback of b and c, and that $v = g(s')$ for some s' in S, then $p(b', c', s') = p(b'', c'', s')$.*

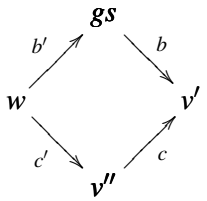
This is why we said at the end of the last section that we needn't be concerned about pullbacks being defined only up to isomorphism.

Proof. The proposition is established by taking a and d in the statement of the put-put law to be identity arrows. In that case both $p(b', c', s')$ and $p(b'', c'', s')$ are equal to $p(c, 1, p(1, b, s'))$. Furthermore, if p_L and p_R satisfy get-put laws then all three are equal to $p_L(c, p_R(b, s'))$. □

7 A sufficient condition

How does one check easily that the mixed put-put law is satisfied in a particular example only known to satisfy the two monotonic put-put laws?

We say that *condition* (*) *is satisfied* if for any object s in S and any pullback (in V)



it is the case that $p_R(c', p_L(b', s)) = p_L(c, p_R(b, s))$.

Proposition 2 *If p_R and p_L satisfy monotonic put-put laws and condition (*) then p defined by*

$$p(a, b, s) = p_R(b, p_L(a, s))$$

satisfies the mixed put-put law.

Proof. The proof is a straightforward application of property (*) along with two applications of monotonic put-put laws. In the notation of Section 6

$$\begin{aligned} p(c,d,p(a,b,s)) &= p_R(d,p_L(c,p_R(b,p_L(a,s)))) \\ &= p_R(d,p_R(c',p_L(b',p_L(a,s)))) \\ &= p_R(d,p_R(c',p_L(ab',s))) \\ &= p_R(dc',p_L(ab',s)) \\ &= p(ab',dc',s) \end{aligned}$$

□

This result makes checking the mixed put-put law relatively easy in many applications. For example, in the traditional database examples pullback is intersection so we only need to check that the result of putting an insert followed by a delete is always the same as putting the "minimal" corresponding delete followed by insert.

Similarly, in the node- and edge-labelled directed graphs example presented in Section 4, the pullback w above is simply the intersection of the node-labelled graphs gs and v'' calculated as subgraphs of v'. Thus w tells us precisely which edges of v' inherited, on updating along b, labels from s that survived the deletions along c. So inheriting those edge labels along b' and labelling the other edges of v'' with the default label when updating along c' will have the same result. We have verified that our example satisfies property (*), and since we already know that the p_R and p_L of that example satisfy the monotonic put-put laws, the proposition assures us that the entire example satisfies the mixed put-put law.

In lectures on this work we have sometimes called condition (*) the mixed put-put law because it is the condition that we first discovered. In fact, it is equivalent to the put-put law as it is easy to see that the mixed put-put law as stated here implies condition (*). We have settled on the version of the mixed put-put law presented in Section 6 since it is more immediately apparent that it has the form that might be expected of a put-put law.

8 General Discussion

Our first general remark is that determining an appropriate put-put law depends upon the assumed nature of the state space of views V. If we consider the collection of all view states to be merely a set, then there is little else that can be done except to propose "the" (meaning the classical) put-put law — the law that many people have argued is unreasonably strong (and we agree). More precise notions of the category of view states lead us to monotonic and ultimately, via spans, to mixed put-put laws. A paper developing a more detailed treatment of state spaces of V and a monadic view of spans is in preparation.

Of course, however detailed the state space of views, if we do not know what transition was used to move from one view to another we cannot use the transition as a component of the domain of *put*. This is the problem of *alignment* which is not taken up in this paper. We have

sought to develop new put-put laws on the assumption that the transition, sometimes called the *delta*, is known.

The treatment developed here sheds further light on the delta-lenses described by Diskin et al [DXC10]. In particular the comma categories that are the domains of Rg and Lg make precise the appropriate (delta-based) domain of *put*, and it seems that with that extra precision only the usual three laws — put-get, get-put and put-put — are required. And we remind readers that put-put is no longer onerous. The appropriate version of put-put is determined by the kinds of updates that the state space of V supports. The monotonic and mixed versions of put-put described here are much less strong (read "less onerous", or indeed "less unreasonable") than classical put-put.

The question of the breadth of applicability of the monotonic, and particularly of the mixed, put-put laws, is an empirical one. It is clear that they hold in many of the circumstances when classical put-put fails, and they have been satisfied in all of the examples that we have had to work with so far. And it has been interesting to maintain the distinction between the monotonic and the mixed put-put laws. In the former case we have not noticed circumstances in which even classical put-put would fail (largely because so many examples have partially ordered state spaces V). In the latter case the circumstances that appear contrary to classical put-put become distinguished because there are many distinct parallel spans from $g(s)$ to v — thus two updates that might unreasonably have been required to be equal under classical put-put are independent, incomparable, and so never arise on the two sides of the mixed put-put equation.

Returning to delta-lenses, the examples presented in the first half of [DXC10] are based on databases and have partial functions as deltas. In the usual treatment of databases the state space of V is a partial order on sets of tuples, with transitions given by tuple inclusion. In this case spans are partial functions (indeed partial injections) and span composition is the same as the composition of the corresponding partial functions. Thus it seems that the examples in [DXC10] are captured by our span based updates of Section 5.

And a final remark about delta-lenses. Very reasonably, they require the put of a delta, of a transition in V, to be a delta, a transition, in the state space S. The reader might be surprised then to see that our put takes an element s of S, and an arrow, say $t : g(s) \to v$, to an *object*, $p_R(t,s)$. In fact, because p_R is functorial it turns out that this does fully determine an appropriate arrow $s \to p_R(t,s)$ (and similarly for p_L and p).

There are significant opportunities for further work on, especially, mixed put-put laws. As category theorists we mention one obvious generalisation — there is no need to take equality in, for example, $p(ab',dc',s) = p(c,d,p(a,b,s))$. In a category of states we would expect isomorphism of objects to suffice. Similarly with property (*). Of course, when the category is a partial order, isomorphism reduces to equality.

Much more interesting is the detailed relationship between mixed put-put laws and algebras for further monads, and the relationships of all of these to universal characterisations via fibrations and op-fibrations. These will be taken up in detail elsewhere.

9 Conclusions

We have proposed monotonic and mixed put-put laws that, in the context of a category of view states V, are weaker than the classical put-put law, but still provide a generalised associativity. In

the examples that we have explored to date, these new put-put laws are easy to check, powerful to use, and not subject to the usual objections against the classical put-put law.

It is noteworthy that all put-put laws discussed here are of the form $p(ba,s) = p(b,p(a,s))$ for suitable notions of composition "ba" — in the monotonic cases it is normal category theoretic composition rightwards or leftwards, in the mixed case it is span composition, and in the classical case it is projection onto the last component.

In ongoing work we have shown that the operation p is also an algebra for a monad, and we are exploring what condition (*) really means for the resultant multiple fibration. We also note that condition (*) is related to the famous Beck-Chevalley condition, and we plan to explore the relationship further.

Acknowledgements: The work presented here benefited from the support of the Australian Research Council and NSERC Canada. The authors are grateful for the generous advice of the editors and the anonymous referees.

Bibliography

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

[BVP06] Bohannon, A., Vaughan, J. and Pierce, B. (2006) Relational Lenses: A language for updatable views. *Proceedings of Principles of Database Systems (PODS) 2006*, 338–347.

[DXC10] Diskin, Z., Yinfei Xiong, and Czarnecki, K. (2011) From state- to delta-based bidirectional model transformations: The assymmetric case. *J. Obj. Technol.* **10**, 1–25.

[FGM07] Foster, J., Greenwald, M., Moore, J., Pierce, B. and Schmitt, A. (2007) Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. *ACM Transactions on Programming Languages and Systems* **29**, 1–65.

[Heg04] Hegner, S. J. (2004) An order-based theory of updates for closed database views. *Ann. Math. Artif. Intell.* **40**, 63–125.

[HP96] Hofmann, M. and Pierce, B. (1995) Positive subtyping. *SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, 186–197.

[HPW11] Hofmann, M., Pierce, B. and Wagner, (2011) Symmetric lenses. *SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)*, 371–384.

[JR07] Johnson, M. and Rosebrugh, R. (2007) Fibrations and universal view updatability. *Theoret. Comput. Sci.* **388**, 109–129.

[JRW10] Johnson, M., Rosebrugh, R. and Wood, R. J. (2010) Algebras and Update Strategies. *J. Univ. Comput. Sci.* **16**, 729–748.

[JRW12] Johnson, M., Rosebrugh, R. and Wood, R. J. (2012) Lenses, Fibrations, and Universal Translations. *Math Structures in Comp Sci*, **22**, 25–42.

[Ole82] Oles, F. J. (1982) A category-theoretic approach to the semantics of programming languages. PhD Thesis, Syracuse University.

[Ole86] Oles, F. J. (1986) Type algebras, functor categories and block structure. In *Algebraic methods in semantics*, 543–573. Cambridge University Press.

[Pie91] Pierce, B. (1991) *Basic category theory for computer scientists*. MIT Press.