Compositionality of Update Propagation: Laxed PutPut

Zinovy Diskin

McMaster University, Hamilton, Canada
diskinz@mcmaster.ca

Abstract

Compatibility of update propagation with update composition (the infamous Putput law) is fundamental for the mathematical modelling of bx, but is not easy to ensure in practical scenarios. This severely restricts practical applicability of elegant algebraic models based on Putput, while leaving practical bx without solid algebraic support is also unsatisfactory. The paper aims to mitigate these problems, and presents the following findings. It is known that PutPut trivially holds for the sequential composition of two inserts or two deletes (what Johnson and Rosebrugh called the monotonic Putput); it also holds for the relational (pullback based) composition of a delete followed by an insert (the mixed Putput). In the present paper, we will see that relational Putput can fail for the relational composition of an insert followed by a delete, which represents a wide class of practically interesting examples. We will also analyze different ways of update composition, and discuss their interaction with update propagation. We will see that update propagation and Putput need a 2-categorical setting, formulate a notion of lax Putput, and show that it is much wider applicable to practical situations than the ordinary strict Putput.

1 Introduction

Compatibility of update propagation with update composition (the infamous Putput) is fundamental for the mathematical modeling of bx, but is not easy to ensure in practical scenarios if we treat compatibility in a straightforward way. This severely restricts practical applicability of elegant algebraic models based on Putput, while leaving practical bx without solid algebraic support is also unsatisfactory (for, at least, some members of the bx community). Putput needs a careful analysis and investigation, but seems to be not a very popular research subject in bx.

Problems with Putput were noticed since the very early work on lenses [9], and continued to be remarked later in, e.g., [8, 15], but the first (and seemingly the only so far) careful and mathematically solid analysis of Putput basics was undertaken by Johnson and Rosebrugh in [12]. They distinguished "easy" *monotonic* Putput, when two consecutive inserts or two consecutive deletes are to be propagated, from more problematic *mixed* Putput, when operation `put` must propagate an interleaving sequence of inserts and deletes. For the later case, they found an equivalent but more elementary and easier to verify condition for the mixed Putput to hold. The present paper continues Putput analysis and presents the following findings.

We will first consider different ways of update composition, and see that in whatever way an algebraic operation of delta composition is defined, the real composition appearing in practice could be different, and, in general, a user input is required to correct an automatically produced delta. Thus, we have a 2-delta between the

In: R. Eramo, M. Johnson (eds.): Proceedings of the Sixth International Workshop on Bidirectional Transformations (Bx 2017), Uppsala, Sweden, April 29, 2017, published at http://ceur-ws.org

automatically produced and the real delta. However, the former always subsets the latter, if deltas are composed as binary (multi)relations. This subsetting is in a sense preserved, when deltas are propagated backward to the source side, and gives rise to a *lax* compositionality of update propagation. Then we will consider a simple example of relational composition, in which an insert is followed by a delete, and see that Putput fails, but, again, some lax version of Putput still holds; we will also argue that this example is a representative of a sufficiently large class of practically interesting and important cases of update propagation.

Thus, different deltas, their compositions and relationships between them form the very core of update propagation and Putput. This observation makes categorical models of bx essentially 2-categorical: deltas are 1-arrows, and relationships between them are 2-arrows. The 2-categorical setting allows us to distinguish the *strict* Putput, which requires equality of the two deltas (and hence their target models), from the *lax* Putput, in which the two deltas and their target models are not equal nor isomorphic, but comparable via a uniquely defined mediating 2-construct. We will see that in many interesting cases, including the insert-delete example mentioned above, while the strict Putput fails, the lax Putput holds. In a sense, problems of Putput mentioned in the literature can be seen as the problems of forcing an essentially 2-categorical lax phenomenon into a 1-categorical strict framework.

Our plan for the paper is as follows. We will begin in Section 2 with a very simple Example 1 to fix the terminology, basic concepts, and issues to be discussed. Specifically, we consider relational, match-based and mixed update compositions (in this paper, updates are spans a.k.a structural deltas between models), and discuss how they coexist with Putput; the last subsection 2.3 builds a more abstract and general framework and presents our first schema/pattern of the lax Putput. In Section 3, we consider Example 2, in which relational Putput fails, discuss how general such type of examples could be, and build our second schema of the lax Putput; the last subsection 3.3 integrates the two schemas into a general lax Putput pattern. Section 4 outlines a formalization in terms of lax delta lenses. Remarks on related work and some historical sentiments can be found in Section 5, which also concludes the paper.

2 Putput and Update Composition (relational, match-based, and mixed)

We will begin with a very simple example explaining the main notions and notations of the delta framework. Then we introduce three types of update composition: relational, match-based, and mixed, and consider how backward update propagation interacts with them.

2.1 Example 1: Basics of update propagation

Models, metamodels, and views

Metamodels are class diagrams with constraints. For example, suppose a metamodel M consisting of the only class Person with two attributes, name1 and name2, both of type String. A *view* of this metamodel is another metamodel N together with a mapping w: $M \leftarrow N$ called the *view definition.* Suppose, for example, that N consists of a class Person with the only attribute name of type String, and the view definition maps Person in N to Person in M, and name in N to name1 in M. It means that in the w-view of M-models, attribute name2 is ignored while values of attribute name1 are called names. Thus, any model instantiating M is mapped to a model instantiating N, and we have a view execution function between the respective model spaces $\mathsf{get}_w\colon \boldsymbol{M} \to \boldsymbol{N}$ (note the reversal of the direction), whose name get abbreviates the command "get the view" (more interesting examples of view definitions via mappings can be found in, e.g., [10, 2]). In the lens framework for bx, the view mapping generating get_w is forgotten, and the index w is omitted.

A simple model A instantiating metamodel M is shown in Fig. 1 (at the left top corner of the grey-border rectangle labelled as Example 1a). It has two objects $p1$, $p2$ with attribute values as shown. As models are stored in computers, $p1$, $p2$ should be understood as object identifiers (OIDs), and name1, name2 as attribute *slots* that hold values, e.g., John and Henry. Function get applied to this model results in model B shown in the figure. Although B has semantically the same objects as A, their OIDs are different — think, for example, about model B stored in another computer. Moreover, mapping get is typically a non-deterministic function as the view execution engine may not control OIDs in the view model stored in a different computer, and thus OIDs may be different when the same get is applied to the same A at different time moments. If we do not want to consider a family of gets indexed by time moments, it makes sense to consider models up to their isomorphisms caused by isomorphic OID replacement; the more so that OIDs are invisible to the user, who sees isomorphic models as equal. Thus, pi, qi should be seen as representatives of the respective equivalence classes, which makes get a well-defined single-valued function (we say "get is defined up to isomorphism"), and we can write

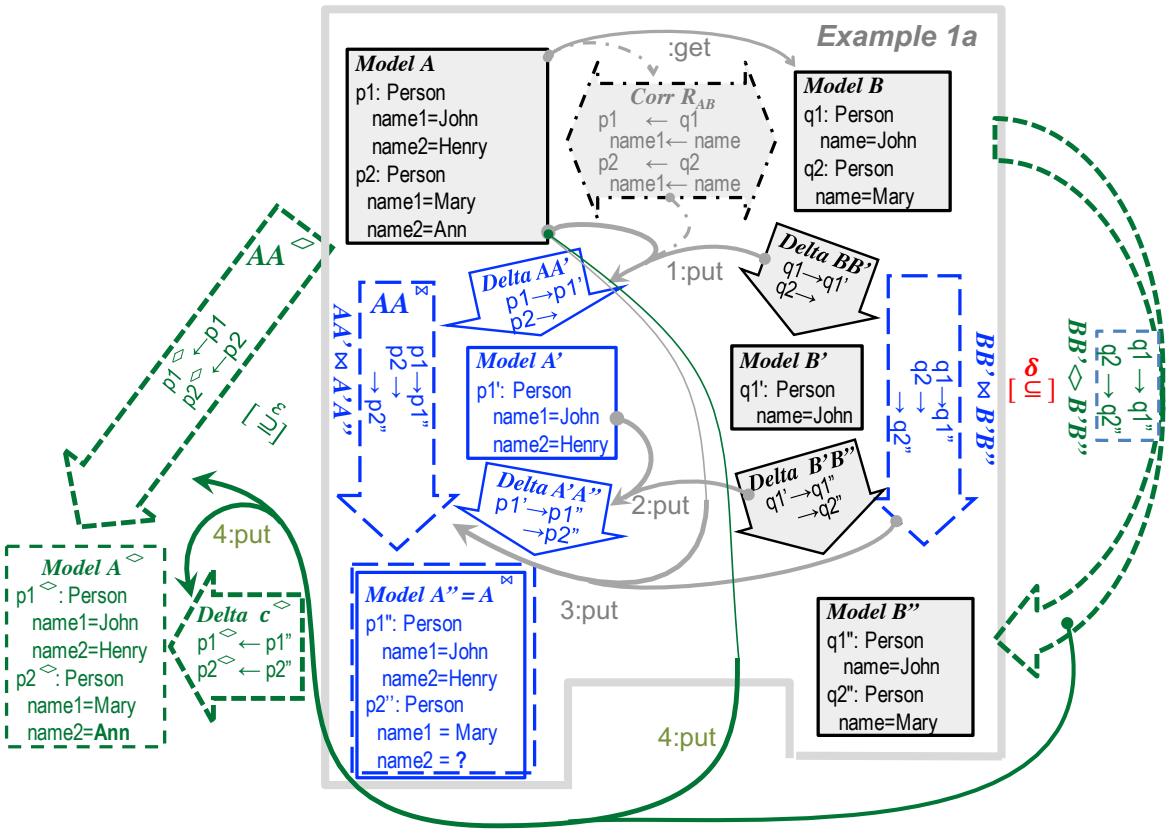


Figure 1: Two examples of backward update propagation: 1a (framed) and 1b (the entire figure)

$B = \mathsf{get}(A)$ or sometime $A.\mathsf{get} = B$ when this notation makes reading formulas easier. In this paper, we will not make "up-to-iso" considerations formally accurate (which is a rather laborious endeavour) and will work with a semi-formal understanding of equality of two models in the hope (supported by the previous experience of managing similar issues) that with a suitable formal definitions everything would work smoothly.

In the figure, the grey arrow labeled :get denotes an application of function get to model A: its bulleted tail is attached to the argument/input object, and the arrow head points to the result – model B. The second arrow head (note the dashed-dotted branch) points to the second object produced by the view execution – a traceability/correspondence mapping R_{AB} that maps elements of B to their origin in A. This mapping is essential for update propagation, but as it can be computed from A (for a given view definition), in the asymmetric lens framework, the correspondence is implicit and officially does not occur into the output of get– hence our dash-dotted notation for it.

Deltas

Consider the following scenario further referred to as *Example 1a*. In the first step, model B is updated to model B'. Imagine a user manually editing the model with a computer, who deletes object $q2$, and does nothing with object $q1$. Such an update can be specified by a delta $\Delta_{BB'}$ (note the respective block-arrow in the figure), which consists of a single link $(q1, q1')$ (in fact, an ordered pair) showing that the object $q1$ is preserved in B'. Placing into the delta a targetless arrow from $q2$ is just a syntactic sugar to show that set of links $\Delta_{BB'}$ does not contain any link from $q2$, which means that object $q2$ is deleted in B'. Thus, formally, $\Delta_{BB'}$ is a singleton $\{(q1, q1')\}$. Of course, in the case of a user manually editing model B with a computer, OIDs $q1$ and $q1'$ would be identical (which is not excluded by our notation). However, if model B' is copied to another computer, having a different OID would be a useful option for our update modelling (with all reservations about OIDs and up-to-iso applicable again).

We assume that our view metamodel says that the multiplicity of attribute name is exactly 1, i.e., any Person object has one and only one name slot. Hence, if two objects are the same, their name slots are also the same,

and we may safely omit links between such slots. However, for attributes with multiplicity more than one (e.g., think of an attribute phone of multiplicity 0..3), data about which phone-slot in model B corresponds to which phone-slot in model B' is to be included into the delta. Linking slots in B and B' declares them to be the same slot, but it does not prohibit changing the value held in the slot. E.g., despite that $q1$ and $q1'$ are linked, which implies their name-slots are linked, the values held in these slots would be different if Person $q1$ changed his name from John in B to, say, Jo in B'. In this paper we do not consider attribute modifications, but some details can be found in [2, Section 3].

Importantly, for the same two models, there can generally be multiple deltas between them. E.g., for models B and B', besides the delta shown in the figure, there is an empty delta that specifies object $q1'$ as a new object different from $q1$, which just happens to have the same name. In this sense, the notation $\Delta_{BB'}$ could be misleading as pairs of models do not actually index deltas, but within our fixed scenario, this notation is convenient.

Update propagation

As updates are specified by deltas, update propagation amounts to delta propagation. In general, there are many source models A' and deltas $\Delta_{AA'}$, whose get-views are equal to B' and $\Delta_{BB'}$, hence we need some *update policy* to ensure uniqueness. A reasonable update policy should propagate updates along the correspondence R_{AB}, i.e., in our case, delete object $p2$ corresponding to $q2$, and keep object $p1$ untouched—the result is the model A' and delta $\Delta_{AA'}$ as shown in the figure. Note that these *computed* model and delta are blank (and blue with a colour display) to distinguish them from *given* (specified by the user rather than computed) models and deltas, which are shaded (and black. The blue colour aims to refer to *mechanical* computation.) The propagation procedure is modelled as an application of operation put to model A and delta $\Delta_{BB'}$ (note the group of arrows 1:put). The correspondence R_{AB} is essentially employed in the propagation, but as it can be computed for a given model A, in the asymmetric lens framework the explicit input for put is a pair $(A, \Delta_{BB'})$ rather than $(R_{AB}, \Delta_{BB'})$ (note the dashed input arrow from R_{AB} to 1:put illustrating the trick).

Now we are ready to approach the main issue of our analysis.

2.2 Example 1 continued: Update composition and Putput

Suppose that after update $\Delta_{BB'}$ was propagated, model B' is again updated by adding a new object $q2''$, which is specified by delta $\Delta_{B'B''}$ (note the arrow without source). Applying put to this delta results in a model A'' with a new object $p2''$ corresponding to $q2$", and the respective delta $\Delta_{A'A''}$. Note that model A' is uncertain, as the name2 value for the newly inserted object is not known, and the attribute slot contains a null value ?.

Our main question is what happens if we compose updates $\Delta_{BB'}$ and $\Delta_{B'B''}$ into a "jump" update $\Delta_{BB''} = \Delta_{B,B'}\,;\Delta_{B',B''}$ (where ; denotes the operation of update composition), and propagate the result to the source side into delta $\Delta^{;}_{AA;}$ from model A to some model $A^{;}$: whether $A^{;} = A''$ and $\Delta^{;}_{AA;} = \Delta_{AA'};\Delta_{A'A''}$, or not?

Two main actors determine the answer:

(a) How operation put restores missing information on the source side;

(b) How much of the information about the component updates is forgotten in the composed (jump) update. We will call this information *mediating*.

In our simple example, restoration (a) is very simple – if a Person in the view is newly inserted, his/her name2 is set to a null. But update composition is non-trivial even in this simple example, and it is the subjects of our analysis below. We will consider two somewhat opposite ways of composing updates. One is *relational* composition, which strives to respect the mediating information; we will also say that relational composition is *history respecting*. The other is *match-based* composition, which entirely ignores the mediating information, and thus is *history ignoring*.

Relational composition and Putput

As deltas consist of links, it's natural to compose deltas by composing links. Consider again deltas $\Delta_{BB'}$ and $\Delta_{B'B''}$ in Fig. 1. The first delta has a link $q1 \xrightarrow{\ell'} q1'$ targeted at $q1'$, and the second one has a link $q1' \xrightarrow{\ell''} q1''$ started from $q1'$; we will call such links *joinable*. By composing these links, we obtain a link $q1 \xrightarrow{\ell' \bowtie \ell''} q1''$ from the source of ℓ' to the target of ℓ'' (symbol $\bowtie$ means to remind about the composition mechanism based on composing joinable links). If the component deltas would have more links, we repeat the procedure and create a jump link for each pair of joinable links. In this way, we obtain a delta $\Delta_{BB'} \bowtie \Delta_{B'B''}: B \to B''$ from the

source of the first delta to the target of the second. Such composition is basically the well-known binary relation composition generalized for multirelations. (And indeed, symbol $\bowtie$ is often used in the database literature to denote the operation of relational join.) We will call this way of composing deltas *relational*.

Note that if even the component deltas are ordinary (not multi) relations (i.e., every possible pair of elements either occurs into the delta or not), their composition described above can result in a multirelation. Indeed, if we have two pairs of joinable links between the same source and target, say, $q \xrightarrow{\ell'_1} q'_1 \xrightarrow{\ell''_1} q''$ and $q \xrightarrow{\ell'_2} q'_2 \xrightarrow{\ell''_2} q''$, the composed delta $\Delta_{BB'} \bowtie \Delta_{B'B''}$ would have two different links, $q \xrightarrow{\ell'_1 \bowtie \ell''_1} q''$ and $q \xrightarrow{\ell'_2 \bowtie \ell''_2} q''$ between the same pair of OIDs; we will call such links *parallel*.

Thus, relational composition treats mediating information rather accurately except the case when an object is inserted in model B', and then deleted in model B''. We will consider such a case later in Sect. 3, but in our Example 1a, relational composition is accurate, restoration of the source information is not problematic, and hence it is not surprising that Putput holds: it is easy to verify that in Fig. 1,

$$\mathsf{put}(A, \Delta_{BB'}) \bowtie \mathsf{put}(A', \Delta_{B'B''}) = \mathsf{put}(A, \Delta_{BB'} \bowtie \Delta_{B'B''}). \tag{1}$$

We will phrase this pleasant result by saying that *relational Putput* holds for Example 1a.

Match-based composition and Putput

Consider the case, when deltas between models are computed by matching based on a key attribute rather than independently set by the user as in the scenario above. Suppose that attribute name on the view side is a key to class (table) Person, i.e., for any two objects $p1, p2$ instantiating the class, equality $p1.\mathsf{name} = p2.\mathsf{name}$ implies $p1 = p2$. For example, deltas $\Delta_{BB'}$ and $\Delta_{B'B''}$ can exactly be computed by matching objects by their name attributes. In this view of deltas and updating, the composition of two deltas would be the delta determined by matching the end models, B and B'', while the mediating model B' is entirely ignored. In a sense, this way of composition is opposite to relational, and we denote it by an "opposite" symbol $\diamond$: $\Delta_{BB'} \diamond \Delta_{B'B''} \stackrel{\text{def}}{=} \Delta^{\diamond}_{BB''}$, where $\Delta^{\diamond}_{BB''}$ denotes the delta produced by matching.

Thus, while in the relationally composed delta $\Delta^{\bowtie}_{BB''} = \Delta_{BB'} \bowtie \Delta_{B'B''}$, object $q2''$ is a newly inserted Mary, the match-based delta $\Delta^{\diamond}_{BB''}$ has an additional link $(q2, q2'')$ that makes object $q2''$ the same Mary earlier existing in model B, and models B and B'' are thus equal (up to iso). Hence, delta $\Delta^{\diamond}_{BB''}$ is propagated to the isomorphism/identity delta $\Delta_{AA^{\diamond}}$ as shown in Fig. 1 by dashed elements outside the frame (green with a colour display), and Putput fails. However, despite Putput stating equality (we say *strict* Putput) fails, we can still find some discipline in the propagation results. To wit: note the horizontal green delta $c^{\diamond}$ from A'' to $A^{\diamond}$, which simply completes model A'' by replacing the null for "new" Mary's name2 by its value Ann for the previously existing Mary. This delta is uniquely determined by our update composition and propagation procedures. Moreover, the discipline also includes relationships between deltas discussed below.

Mixed composition and Putput: 2-deltas

We have considered two possibilities of update composition: $\Delta^{\bowtie}_{BB''} = \Delta_{BB'} \bowtie \Delta_{B'B''}$ for relational composition (*r-composition*), and $\Delta^{\diamond}_{BB''} = \Delta_{BB'} \diamond \Delta_{B'B''}$ for match-based *m-composition*. However, which of these two variants is right, i.e., correctly models the reality? We instantly find than neither one does. It may happen that the "new Mary" is indeed a new person that just happens to have the same name as Mary from model B, and then delta $\Delta^{\diamond}_{BB''}$ is wrong. (That is, while attribute name is a good key for a given state of the model, it does not work across the states, i.e., states at different time moments. We will phrase this by saying that name is a valid *static* key, but invalid *dynamic* key.) Or it may happen that new Mary is the same Mary from model B, which left the domain at the moment specified by model B' but came back at the moment of model B''.[1]Then delta $\Delta^{\bowtie}_{BB''}$ is wrong.

Moreover, for models containing multiple classes, there are many formal ways of composing deltas, which differ by for which classes composition is relational, and for which is match-based. For example, suppose that our models contain two classes, Person and Car, and for objects of class Person we use r-composition, while for Car, m-composition is used. Then, we would have four ways of computing composed deltas: entirely relational,

[1]Think, for example, about class Person as a class of employees of some company, and of Mary as an employee who quit the company at the time of model B' and was hired again at the time of B''. Or, even simpler, the user who was editing model B, deleted Mary and saved the result as B', but later recognized that it was an error, and so in model B'' the same object Mary is to be restored with all its attributes.

entirely match-based, and two mixed ways depending on which of the classes is managed relationally and which is match-based. Thus, there is a whole variety of formal automatic compositions, but often neither of them can always be semantically correct as we have just discussed. That is, for a particular pair of deltas $\Delta_{BB'}$, $\Delta_{B'B''}$, we can find a formal composition $\Delta^{\#}_{BB''}$ ($\# \in \{\bowtie, \diamond, \bowtie\diamond, ...\}$) equal to the correct delta $\Delta_{BB''}$, but chances that it works for all pairs are not significant. For instance, in Example 1, we can have $\Delta_{BB''} = \Delta^{\diamond}_{BB''}$, but $\Delta_{B'B'''} = \Delta_{B'B''}\bowtie\Delta_{B''B'''}$ for a consecutive series of updates from B' to B'' to B'''. We need to find a way of managing this diversity.

As we have different types of deltas between models, we need a way to compare them. The simplest one is to compare deltas as sets of links, which works well under condition that the source and the target models of deltas to be compared are the same (i.e., deltas are parallel). For such deltas, their sets of links can coincide, be disjoint, intersected, or one be included into the other. In the latter case, we call the inclusion mapping a *2-delta* and denote it by a double arrow, or sometimes by the set-inclusion symbol $\subseteq$. (Then deltas between models can be called 1-deltas.) For instance, in Fig. 1 we have two 2-delta declarations labelled by δ on the right, and ε on the left. The content of these declarations is specified in the equations below:

$$\delta: \quad \Delta_{BB'}\bowtie\Delta_{B'B''} \quad \subseteq \quad \Delta_{BB'}\diamond\Delta_{B'B''} \tag{2}$$

$$\varepsilon: \quad (\Delta_{AA'}\bowtie\ \Delta_{A'A''})\bowtie c^{\diamond} \quad \subseteq \quad \Delta_{AA\diamond} \tag{3}$$

Note that mapping $c^{\diamond}$, which fixes the gap between models A'' and $A^{\diamond}$, is a necessary ingredient in (3) as 2-delta subsetting can only be declared for parallel 1-deltas.

Now we can summarize our analysis of Example 1 in the following way. If the real composed delta coincides with the relationally composed, $\Delta_{BB''} = \Delta^{\bowtie}_{BB''}$, then *strict* Putput stating the equality of two deltas holds — see eq. (1). If the real composed delta is match-based, $\Delta_{BB''} = \Delta^{\diamond}_{BB''}$, and hence 2-delta δ gives us a comparison between the real and the relationally composed 1-deltas, then eq. (3) gives us the following *lax* form of Putput:

$$\mathsf{put}(A, \Delta_{BB'})\bowtie\ \mathsf{put}(A', \Delta_{B'B''})\bowtie \mathsf{put}^2(A, \delta) \quad \subseteq \quad \mathsf{put}(A, \Delta_{BB''}) \tag{4}$$

where delta $c^{\diamond}$ from (3) is considered as the result of the 2-activity of operation put, or 2-projection of put, and is denoted by $\mathsf{put}^2(A, \delta)$ (we have also used associativity of $\bowtie$).

Our next goal is to see how much the content of the story described above for Example 1 can be extended for more general cases of update propagation.

2.3 Generalizing Example 1

Relational vs. match-based composition

First, we need to make the notion of (structural) delta more accurate via the notion of *span*. That is, an update u of model X to model Y is modelled by a diagram $X \xleftarrow{u_{\mathsf{del}}} H_u \xrightarrow{u_{\mathsf{ins}}} Y$, in which H_u is the model consisiting of all sameness links declared in u, which is called the *head* of the span, and u_{del}, u_{ins} are injective mappings that specify, resp., deletions and insertions caused by u. To wit: X's elements beyond the range of u_{del} are deleted, and Y's elements beyond the range of u_{ins} are inserted. For example, the head of span $\Delta_{BB'}$ in Fig. 1 is a model consisting of one object $q1q1'$, whose attribute name has value John by default. If this update along with deletion of $q2$ would change the name of $q1$, say, to Jo, then the name-slot of $q1q1'$ would hold a null, and mapping u_{del} would map this null to John, while u_{ins} would map it to Jo (details and formalization can be found in [2, Sect.3]) Clearly, update u is a pure deletion iff u_{ins} is the identity of Y (up to iso), and u is a pure insertion iff u_{del} is the identity of X (up-to-iso). Spans will be denoted by stroked arrows $u: X \nrightarrow Y$ in text, or bulleted arrows $X \overset{u}{\longrightarrow\!\!\bullet} Y$ in diagrams.

Now suppose we have two consecutive updates on the view side: $v': B \nrightarrow B'$, $v'': B' \nrightarrow B''$, and let $v^{\bowtie}: B \nrightarrow B''$ and $v^{\diamond}: B \nrightarrow B''$ denote their r- and m-compositions resp. The r-composition is given by the pullback (PB) of mappings v'_{ins} and v''_{del} as shown in the diagram Fig. 2 (note the label $[pb_1]$), which precisely describes the operation of joinable link composition that we considered above. Thus, $H_{v^{\bowtie}} \overset{\text{def}}{=} H_1$, $v^{\bowtie}_{\mathsf{del}} \overset{\text{def}}{=} p'_1; v'_{\mathsf{del}}$, and $v^{\bowtie}_{\mathsf{ins}} \overset{\text{def}}{=} p''_1; v''_{\mathsf{ins}}$. The m-composition is specified by another PB: a key set of attributes is modeled by a set of functions from the object set of a model to some domain of values (note arrows key and key'', which actually are tuples of functions), to which we apply PB (note the label $[pb_2]$) and obtain span (H_2, p_2, p''_2). Now $H_{v^{\diamond}} \overset{\text{def}}{=} H_2$, $v^{\diamond}_{\mathsf{del}} \overset{\text{def}}{=} p_2$, and $v^{\bowtie}_{\mathsf{ins}} \overset{\text{def}}{=} p''_2$.

If the diamond $H_1 B \mathsf{Val} B''$ is commuttaive (we will discuss it shortly), i.e.,

$$p_1'; v'_{\mathsf{del}}; \mathsf{key} = p_1''; v''_{\mathsf{ins}}; \mathsf{key}'' \tag{5}$$

then by the universality of $[pb_2]$, we have a uniquely defined mapping ! as shown in the diagram. Note that although both spans, (p_1', p_1'') and (p_2, p_2''), are jointly monic (as produced by PB), span $\boldsymbol{H} = (p_1'; v'_{\mathsf{del}},\ p_1''; v''_{\mathsf{ins}})$ is not necessarily monic, and hence mapping ! is not necessary injective (and indeed, matching does not care about by how many ways an object in B can be linked to its match in B'' via a mediating object in B"). But if span $\boldsymbol{H}$ is jointly monic, which is quite normal (e.g., in our Example 1), then mapping ! is injective, and becomes a set inclusion when we consider heads of spans up to iso – this is the inclusion δ in Fig. 1.

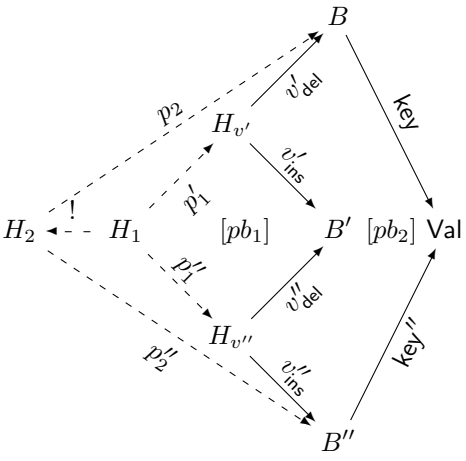


Figure 2: Relational vs. match-based composition

Commutativity (5) is essential for the existence of mapping !. This condition requires that an object does not change its key attribute values. For instance, in the example in Fig. 1, if object $q1$ changes its name to, say, Jo in B' and will keep it in B'', then semantically true link $q1q1'' \in H_{v^{\bowtie}}$ will be lost in $H_{v^{<>}}$. On the other hand, if there was some Jo in model B, who is deleted from B' and B'', matching B and B'' would mistakenly link two different objects (named Jo and John in B) as the same. Note that in both these cases of erroneous match, attribute name remains a valid static key, i.e., no two different objects at a model state have the same name. Thus, the match-based composition is semantically flawed and can do both: create invalid links and miss valid links.

Relational composition behaves much better: although it can miss valid links (and treat returned Mary from model B as a new Mary), it cannot create an invalid link. It allows us to postulate a semantically valid inclusion $\delta\colon v^{\bowtie} \hookrightarrow v$ for any consecutive pair of updates $v'\colon B \nrightarrow B'$, $v''\colon B' \nrightarrow B''$ and their r-composition $v^{\bowtie} = v' \bowtie v''$ as shown in the right part of diagram in Fig. 3 (note the label [=] declaring commutativity of the triangle). This observation is formalized in Sect. 4.1 via the notion of a model as a lax functor. Laxity is also fundamental for Putput as discussed next.

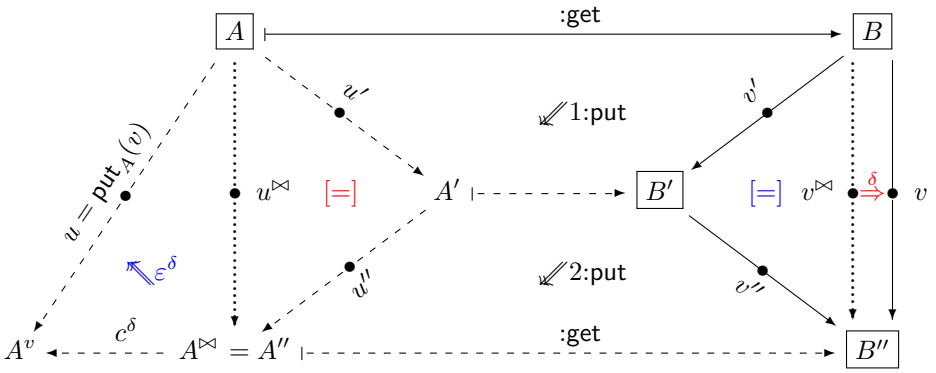


Figure 3: Towards general Putput, I: Backward propagation of 2-deltas

Lax Putput

Consider the diagram in Fig. 3, in which all vertical and inclined arrows are spans (note bulleted bodies of those arrows) denoting deltas. Deltas on the view side are denoted by v with superscripts, their backward propagations on the source side are denoted by u with the respectrive superscripts, e.g., $u' = \mathsf{put}_A v'$ and $u'' = \mathsf{put}_{A'} v''$, where we use a bit more compact notation, in which delta $\mathsf{put}(A, v)$ is denoted by $\mathsf{put}_A v$. We assume that Putput holds for r-composition, and thus $u^{\bowtie} \stackrel{\text{def}}{=} \mathsf{put}_A(v^{\bowtie})$ equals to $u' \bowtie u''$ (note the left label [=]).

As discussed above, 2-arrow δ says that delta v contains additional sameness links, and thus there are objects in B'', which are new according to delta $v^{\bowtie}$ but came from B according to v. As backward propagation of newly created objects leads to source models with nulls, in general, model $A^{\bowtie}$ has nulls where model A^v has

certain values. Thus, models A^v and $A^{\bowtie}$ are different, and deltas $u = \mathsf{put}_A v$ and $u^{\bowtie}$ are also different and even non-parallel. However, as model $A^{\bowtie}$ has nulls where model A^v has certain values, there is a uniquely defined (partial) completion delta c^{δ} as shown in the diagram.[2] Finally, it's reasonable to assume that update propagation is monotonic wrt. inclusion of sets of links, and if delta $v^{\bowtie}$ has less links than v, then delta $u^{\bowtie}\,;c^{\delta}$ has less links than u, which is shown by 2-delta ε^{δ}.

The schema looks plausible, but it assumes that relational Putput holds strictly. In the next section we consider a simple example showing that this assumption can fail if update v' is insertion, while v'' is the respective deletion.

3 Problems of Relational Putput

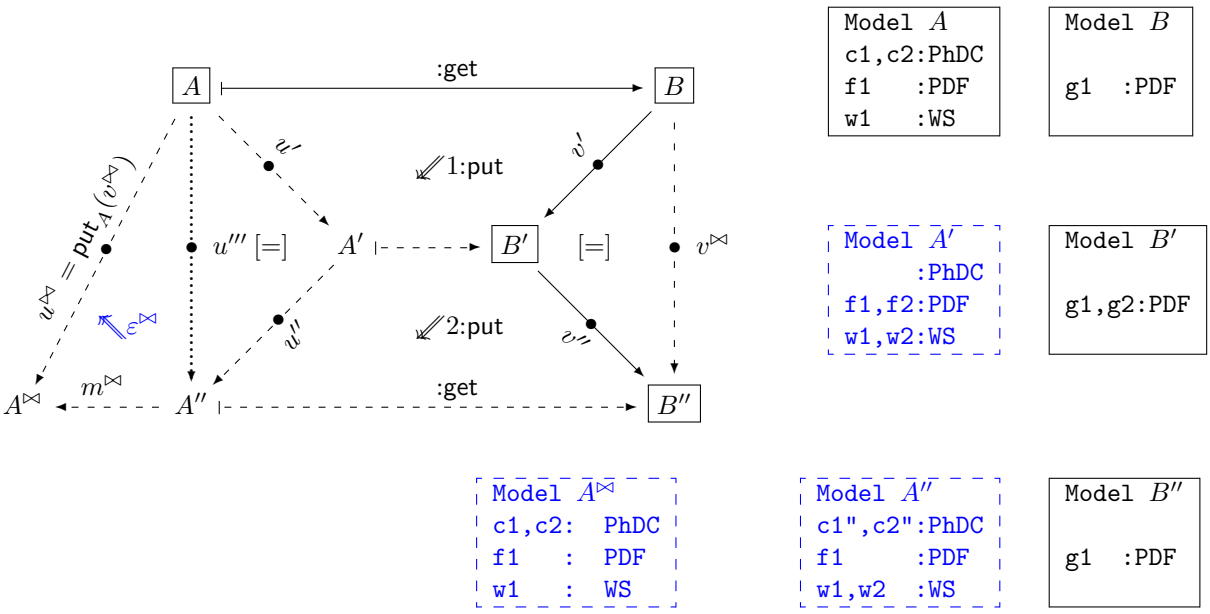


Figure 4: Towards general Putput, II: Lax relational Putput

3.1 Example 2: Relational Putput can fail

We consider yet another synchronization scenario. Suppose that the source models specify states of some research project consisting of two tasks. Each of them is to be performed by either a post-doctoral fellow (PDF), or by two PhD candidates (PhDC), but due to budgetary constraints, only one of these options is allowed. In addition, a PDF is to be provided with one work station (WS). For example, model A in the top right part of Fig. 4 shows some state of the project, in which the workforce consists of two PhDC and one PDF with his workstation. On the view side, only PDFs involved in the project are shown. This gives us the view B shown in the figure.

Now suppose that an updated view B' shows a new PDF $g2$ added to PDF $g1$. Hence, due to budgetary constraints, two PhDC should be assigned to another project—see model A', whose dashed frame (blue with a color display) shows that this model is computed. In the next step of the scenario, the recent PDF $g2$ quits, and so two PhDC should be hired on the source side to ensure performance, while $g2$'s workstation may be kept to minimize the amount of changes — these changes result in model A''. Whether PhDC $c1'', c2''$ are fresh for the project, or the previous PhDCs got involved again, this information cannot be specified with delta $v'' : A' \nrightarrow A''$ and is lost for update v''. On the other hand, relational composition $v^{\bowtie} = v' \bowtie v''$ is identity, which is propagated to identity on the source side. Thus, the state $A^{\bowtie}$ is the same as A (up to iso), and delta $u^{\bowtie}$ is identity. Note the essential difference between updates $u''' = u' \bowtie u''$ and $u^{\bowtie}$: not only model A'' has an extra WS in comparison with $A^{\bowtie}$, but according to update u''', model A'' has two newly hired PhDC, while two PhDC in model $A^{\bowtie}$ are the same as in state A according to update $u^{\bowtie}$. Thus, relational Putput fails.

However, the good news is that some lax discipline of update propagation still holds. Indeed, there is a uniquely defined *mediating* delta $m^{\bowtie} : A^{\bowtie} \nleftarrow A''$ (consisting of a set of horizontal links in Fig. 4), which map

[2] A rather complicated theory of uncertain models and completions can be found in [5], but it is much more general than our needs in the present paper.

(a) newly created objects in A'' to their counterparts in $A^{\bowtie}$, (b) objects in A'' that were preserved from A (via composition u''') to their counterparts in $A^{\bowtie} \cong A$, and, finally, (c) deletes extra objects in A''. Of course, there are two bijections of set $\{c1'', c2''\}$ to set {c1,c2}, but if roles of the two PhDCs are indistinguishable, then our up-to-iso approach to models equalizes these two bijections, while if the two PhDCs play different roles specified in the metamodel, then there is only one role-preserving bijection: we map $c1''$ to that ci, which plays the same role.

Now composing u''' with $m^{\bowtie}$ results in a delta parallel to $u^{\bowtie}$, which does not have any extra links due to (c), but misses some of $u^{\bowtie}$'s links due to broken links in B'. For instance, in Fig. 4, there are links $c1 \to c1$ and $c2 \to c2$ in $u^{\bowtie}$, but there are no links $c1 \to c1''$, $c2 \to c2''$ in u'''. This explains the 2-delta $\varepsilon^{\bowtie} : u''' \bowtie m^{\bowtie} \Rightarrow u^{\bowtie}$.

Despite the simplicity of the example, it illustrates a sufficiently general mechanism of violating strict Putput, but keeping a lax Putput—we discuss this in the next section.

3.2 Generalizing Example 2

In simple cases, a view definition mapping w: $\boldsymbol{M} \leftarrow \boldsymbol{N}$ is just a projection that cuts out a piece of $\boldsymbol{M}$ (e.g., in our Examples 1 and 2). In complex cases, we first augment $\boldsymbol{M}$ with derived elements defined by queries (in fact, such elements are query definitions), and then cut out a required part in the augmented $\boldsymbol{M}$, i.e., a view definition is a Kleisli mapping w: $\mathcal{Q}(\boldsymbol{M}) \leftarrow N$ wrt. the monad specifying the query language. Then computing the w-view of model $A \in \boldsymbol{M}$, $\mathsf{get}_w(A)$, amounts to, first, executing the queries in the image of w, which results in an augmented model $Q(A)$, then cutting out the respective piece in $Q(A)$, and finally retyping the result to metamodel $\boldsymbol{N}$. Details of how this works can be found in [7]. Now suppose that model $B = \mathsf{get}_w(A)$ is updated by inserting a piece Y, which we informally encode by writing $B' = B + Y$. To restore consistency, a respective piece X should be added to $Q(A)$, $Q(A') = Q(A) + X$, and if the queries used in the view definition are monotonic (which is anyway a basic assumption underlying functoriality of get), then some piece Z should be added to A so that $A' = A + Z$ and $Q(A') = Q(A) + Q(Z)$.

However, normally models are bound by constraints, and compliance with some constraints, say, R, may force to complement the addition of Z to A by adding to A another related piece, say, Z_R (e.g., workstations for PDFs in our example). Moreover, other constraints (like budgetary requirements in our example), say Σ, may force to complement the addition of Z by deleting from A some piece Z_Σ (removal of PhDC in our example). We encode this complex interplay of different constraints by writing

$$A' = A + Z + Z_R - Z_\Sigma = (A - Z_\Sigma) + Z + Z_R \tag{6}$$

The next step of our scenario is deletion of Y so that $B'' = B + Y - Y = B$. Consistency restoration leads to deleting X from $Q(A')$, which for many update policies would cause deleting Z from A'. However, deletion of Z does not necessarily imply deletion of Z_R like in our example with workstations, i.e., constraints R may work in only one direction. Also, the metamodel may have constraints Π dual to Σ in the sense that deletion of Z from A' causes addition to A" of some piece Z_Π (performance requirements in our example forcing to have either two PhDC or one PDF for the task). Encoding this interplay in our "algebra" gives us equation

$$A'' = (A - Z_\Sigma) + Z_R + Z_\Pi \tag{7}$$

Importantly, if even "fighting" constraints Σ and Π are of equal power (like budgetary and performance requirements in our example), and pieces Z_Σ and Z_Π are isomorphic (like in our example, where the deleted piece Z_Σ amounts to two PhDCs, and the added piece Z_Π amounts to two PhDC again), the total of $(A - Z_\Sigma) + Z_\Pi$ is not A (i.e., update u''' is not an identity) as relational composition does not create links between the newly added piece Z_Π and its counterpart Z_Σ (exactly like in our example, when two PhDCs are fired and then hired again).

Now we notice that model A'' can be updated to model $A^{\bowtie} \cong A$ in the following canonic way $m^{\bowtie}$: the piece $(A - Z_\Sigma)$ is identically mapped to itself in A, the new piece Z_Π (two new PhDCs) is mapped to its counterpart Z_Σ (two previous PhDCs), and finally the piece Z_R (workstation w2) generated by "unidirectional" contraints R is deleted. Composing just specified delta with vertical composed delta from A to A'' via A' (i.e., u''') still lacks links from Z_Σ to Z_Π, which gives us 2-delta $\varepsilon^{\bowtie}$ as shown in the diagram Fig. 4.

The considerations above although quite informal still demonstrate the generality of the mechanism underlying the example. A mandatory relationship R regulating additions of new source elements, and mutually opposite constraints Σ and Π (resources vs. performance) seem to be quite general notions appearing in many view definition cases, and their interaction between themselves and update propagation were specified in rather

general terms. Obviously, if relational Putput fails in a simple example, it will fail in many other more or less similar and practically interesting cases. What is less clear is how stable the idea of having a lax discipline for relational Putput, i.e., having a unique 1-delta $m^{\bowtie}$ and a unique 2-delta $\varepsilon^{\bowtie}$ medaiting the two results of update propagation. Here an accurate formalization of the generalization above would be very helpful, and it hopefully could be done if we enrich the model spaces with some additive structure suggested by equations (6,7).

3.3 Lax Putput

Diagram in Fig. 3 specifies a lax Putput schema in the assumption that relational Putput holds strictly. As we have just seen, relational Putput is also lax, and thus the schema should be modified as shown by diagram in Fig. 5. This diagram integrates the two previous diagrams, and we again slightly simplify notation to make the meaning of the diagram more transparent. The right-most triangle commutativity means that $v^{\bowtie} = v' \bowtie v''$. The real update delta v supersets $v^{\bowtie}$ which is shown by a 2-arrow δ.

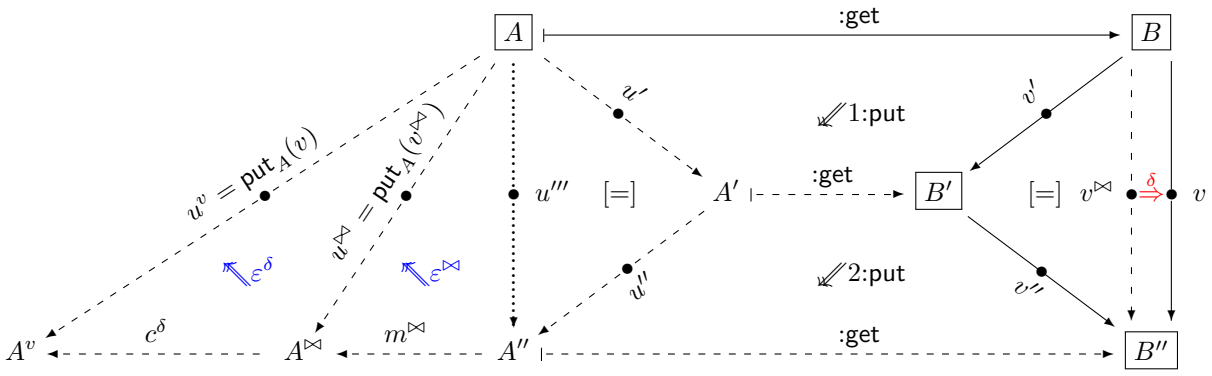


Figure 5: Towards general Putput, III: diagrams in Fig. 3 and Fig. 4 are integrated

Deltas v' and v'' are propagated to u' and u'' resp, and $u''' = u' \bowtie u''$ (note the second commutativity declaration). However, the backward propagation of $v^{\bowtie}$, $\mathsf{put}_A v^{\bowtie}$, is not equal to u''' as relational Putput can fail if v' is an insertion and v'' is a deletion. Thus, $\mathsf{put}_A v^{\bowtie}$ is a different delta $u^{\bowtie}$, and mediating delta $m^{\bowtie}$ together with 2-delta $\varepsilon^{\bowtie}$ provide a comparison between the two. In this way, the laxity of relational Putput is provided by the triangle around $\varepsilon^{\bowtie}$. Similarly, the leftmost triangle around ε^{δ} provides the mediating constructs for propagating back 2-delta δ. An important difference between these two triangles is that c^{δ} is a partial completion mapping that does nothing besides filling some null slots with values (recall our Example 1). In contrast, while delta $m^{\bowtie}$ also includes a significant completion part, it also can delete objects (recall our Example 2). However, neither c^{δ} nor $m^{\bowtie}$ insert objects in our examples.

Now by standard arguments of 2-category theory, we conclude that tiling the two comparison triangles results in a comparison triangle with base $m^{\bowtie} \bowtie c^{\delta}$ and comparison 2-arrow $(\varepsilon^{\bowtie}; c^{\delta}); \varepsilon^{\delta}$, where ; denotes composition of a 2-arrow with 1-arrow (actually with the respective 2-identity) in the standard 2-categorical way.

4 Towards Formalization: Lax Lenses

We will outline a formal framework, in which our preceding discussion can be specified. This work is in progress, some constructs behave well and settled, but there are also several ad hoc definitions, whose clean categorical elaboration is still to be found.

4.1 Models as (lax) trajectories

As discussed in [6], model synchronization is a dynamic rather than static notion: what is to be synchronized is trajectories of models in their model spaces rather than discrete model states. For the asymmetric case, we may assume the same tree I of version numbers for both the source and the view models, which are interpreted as mappings $A\colon I \to \boldsymbol{M}$ and $B\colon I \to \boldsymbol{N}$ considered consistent if $A_i.\mathsf{get} = B_i$ for all $i \in I$ (we write A_i, B_i for $A(i)$, $B(i)$ respectively). In the delta setting, we have a richer picture in which tree I is considered a graph, model

spaces are also graphs, and A, B are graph morphisms considered consistent if $u_{ij}.\mathsf{get} = v_{ij}$ for all pairs $i \le j$ in I with i being the parent of j if $i < j$, and $u_{ij} = A(ij)$, $v_{ij} = B(ij)$ being the respective updates (deltas). If the source update u_{ij} is the result of backward propagation of the view update v_{ij}, then the equality above is the PutGet law for delta lenses.

As update composition is a key part of the picture, we consider model spaces as categories, and any tree I can be seen as a poset and hence a category as well. To fully employ the categorical machinery, we would like to make mappings A, B functors, i.e., graph morphisms preserving composition. However, as our discussion in Section 2 shows, it would exclude many practically important cases. Fortunately, relational composition always subsets semantically valid composition, and then declaring mappings A, B to be *lax* functors into respective model spaces considered as 2-categories, would not be practically restrictive. Thus, for any pair of composable arrows in I, i.e., for any triple $i \le j \le k$, we require existence of mediating 2-arrows: $\varepsilon_{ijk} : u_{ij} \bowtie u_{jk} \Rightarrow u_{ik}$ for A and $\delta_{ijk} : v_{ij} \bowtie v_{jk} \Rightarrow v_{ik}$ for B[3], which make A, B lax functors. Hence, the slogan: *models are lax trajectories*. Lax delta lenses are intended to be a gadget that synchronizes such lax trajectories by propagating updates back and forth.

4.2 Lax lenses

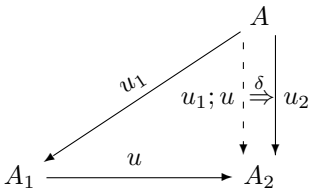
The forward propagation half of the lens structure is easy. Model spaces are categories with a posetal 2-structure (i.e., categories enriched over posets), and $\mathsf{get}: \boldsymbol{M} \to \boldsymbol{N}$ is a strict 2-functor. It is easy to see that 2-functoriality is just an extension of get's monotonicity for 2-arrows. Two models (as trajectories) $A: I \to \boldsymbol{M}$, $B: I \to \boldsymbol{N}$ are called *consistent* if $A; \mathsf{get} = B$ as 2-functors. This equality implies (o) state-based consistency $A_i.\mathsf{get} = B_i$ for any version number $i \in I$, (i) delta-based consistency $u_{ij}.\mathsf{get} = v_{ij}$ for any pair $i \le j$ in I, and (ii) 2-delta consistency $\varepsilon_{ijk}.\mathsf{get} = \delta_{ijk}$ for any triple $i \le j \le k$ in I.

The second half of the lens structure—backward propagation with operation put—is much more complicated because we need to describe two lax phenomena: (a) put's action on 2-deltas, and (b) lax Putput as such. We will consider the former in Sect. 4.2.2 and the latter in Sect. 4.2.3, but first we need some preliminaries.

4.2.1 Preliminaries

Triangle functors

Let $\boldsymbol{C}$ be a small 2-category. We will write $\boldsymbol{C}_i$, $i = 0, 1, 2$ for, resp., its set of objects (0-cells), arrows (1-cells) and 2-arrows (2-cells). If u is an arrow (1- or 2-cell), then ${}^\bullet u$ denotes its source cell, and $u^\bullet$ its target cell. For 1-arrows this notation is especially suggestive as bulleted symbols denote objects, and we will mainly use this notation for 1-arrows.



Let $A \in \boldsymbol{C}_0$ be an object. The set of all 1-arrows from A is denoted by $\boldsymbol{C}_1(A, *)$, and can be converted into a category $\boldsymbol{T}A$ in the following way. Objects of $\boldsymbol{T}A$ are 1-arrows from A, i.e., $\mathsf{Obj}\ \boldsymbol{T}A = \boldsymbol{C}_1(A, *)$. A $\boldsymbol{T}A$-arrow from 1-arrow $u1: A \to A_1$ to 1-arrow $u2: A \to A_2$ is a pair $\triangle = (u, \delta)$ as shown in the inset figure with 1-arrow $u: A_1 \to A_2$ called the *base* of $\triangle$, and 2-arrow $\delta: u1; u \Rightarrow u2$ called the *delta* of $\triangle$. We will often write $\triangle_{u,\delta} : u1 \Rightarrow u2$.

We define composition of triangles by their tiling: given $\triangle = (u, \delta) : u_1 \Rightarrow u_2$ and $\triangle' = (u', \delta') : u_2 \Rightarrow u_3$, composition $\triangle; \triangle'$ is the pair $(u; u', \delta; u'; \delta')$. It is easy to check that such composition is associative, and pair $(\mathsf{id}_{A_2}, \mathsf{id}_{u_2})$ is the identity of u_2 wrt. this composition. Thus, $\boldsymbol{T}A$ is a category.

There are two special subcategories in $\boldsymbol{T}A$, which have all objects that $\boldsymbol{T}A$ has, but fewer arrows. The first one, $\boldsymbol{T}^{\Rightarrow}A$, has arrows with the base being identity. Then triangles degenerates into 2-arrows between 1-arrows from A, and non-parallel 1-arrows in $\boldsymbol{T}^{\Rightarrow}A$ are not connected. The second special category has arrows with the delta being 2-identity. It is easy to see that its triangle-arrows are nothing but commutative triangles with vertex A and we denote this category by $\boldsymbol{T}^{=}A$ (that is, $\boldsymbol{T}^{=}A$ is nothing but the slice category $\boldsymbol{C}\backslash A$).

Any arrow $f: A \leftarrow A'$ gives rise to a *reindexing functor* $f^*: \boldsymbol{T}A \to \boldsymbol{T}A'$ in an obvious way by precomposition. In more detail, an arrow $u_1: A \to *$ is mapped to $f^*(u_1) \stackrel{\text{def}}{=} f; u_1: A' \to *$, and triangle $u_1 \xrightarrow{u,\delta} u_2$ is mapped to triangle $(f; u_1) \xrightarrow{u, f; \delta} (f; u_2)$ with the same base u but a different (composed) delta. Thus, any category $\boldsymbol{C}$ is equipped with a *triangle* functor $\boldsymbol{T}: \boldsymbol{C} \to \boldsymbol{Cat}^{\mathsf{op}}$. We will write $\boldsymbol{T}_{\boldsymbol{C}}$ when we need to make the carrier category

[3] in more detail, we have mappings between the heads of the spans, which commutes with their legs

explicit. It is easy to see that reindexing preserves the two special types of arrows so that we also have functors $\boldsymbol{T}^{\Rightarrow}\colon \boldsymbol{C} \to \boldsymbol{Cat}^{\mathsf{op}}$ and $\boldsymbol{T}^{=}\colon \boldsymbol{C} \to \boldsymbol{Cat}^{\mathsf{op}}$.

Finally, suppose $F\colon \boldsymbol{C} \to \boldsymbol{D}$ is a 2-functor (and hence a 1-functor as well). As any 1-functor preserves 1-arrow parallelism and composition, a 2-functor F gives rise to a family of functors between triangle categories $F_A\colon \boldsymbol{T}_{\boldsymbol{C}}A \to \boldsymbol{T}_{\boldsymbol{D}}B$ with $B = F(A)$ indexed by objects $A \in \boldsymbol{C}_0$. Below we will omit subindex A in F_A.

Lax lenses – a brief sketch

Now we can describe a view mechanism as a 2-functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ from a 2-category (model space) $\boldsymbol{M}$ to a 2-category $\boldsymbol{N}$. This functor gives rise to a family of functors $\mathsf{get}_A\colon \boldsymbol{T}_{\boldsymbol{M}}A \Rightarrow \boldsymbol{T}_{\boldsymbol{N}}B$ with $B = \mathsf{get}(A)$ indexed by objects $A \in \boldsymbol{C}_0$. Backward propagation is modelled by two families of mappings inverse to get, both indexed by models A from $\boldsymbol{M}$. The first family, $\mathsf{put}^{\Leftarrow}_A\colon \boldsymbol{T}_{\boldsymbol{M}}A \leftarrow \boldsymbol{T}^{\Rightarrow}_{\boldsymbol{N}}B$, manages backward propagation of 2-deltas. The second, $\mathsf{put}^{\triangle}_A\colon \boldsymbol{T}_{\boldsymbol{M}}A \leftarrow \boldsymbol{T}^{=}_{\boldsymbol{N}}B$, provides lax Putput. However, Putput laxity is special: the comparison is provided by a pair consisting of a 1-arrow and a 2-arrow, while an ordinary laxity only involves 2-arrows. (Actually we will see that the target categories of mappings $\mathsf{put}^{\triangle}_A$ are *pyramids* — constructs slightly more general than triangles.) Below we describe these two families in detail.

4.2.2 Backward propagation of 2-deltas: Put and the 2-structure

Suppose for a view $B = \mathsf{get}(A)$, we have two parallel updates from B, $v_1, v_2\colon B \to B'$, one subsetting the other via $\delta\colon v_1 \Rightarrow v_2$ (see Fig. 6). It means that v_1 creates more new objects in B' (more accurately, more objects in B' are considered new according to delta v_1). Hence, the backward propagation of v_1, update $u_1\colon A \to A'_1$, creates more new objects on the source side than $u_2\colon A \to A'_2$. Hence, in general, model A'_1 will contain more nulls (be less certain) than model A'_2 (see Fig. 1 for a simple example). Hence, there should be a uniquely defined partial completion update $c_\delta\colon A'_1 \to A'_2$ and uniquely defined 2-arrow $\varepsilon^\delta\colon u_1; c^\delta \Rightarrow u_2$. In its current state, the theory of partial completions [3] is complex and under construction, and in the present paper we will work in a more abstract setting and consider c^δ to be just an update.

Arity of $\Leftarrow$-propagation

Given a 2-functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ called a *view*, we define an operation of *backward $\Leftarrow$-propagation* $\mathsf{put}^{\Leftarrow}$ as the following three family of operations/mappings.

a) A family of mappings

$$\mathsf{put}^{\Leftarrow_0}_A\colon \boldsymbol{M}_1(A, *) \leftarrow \boldsymbol{N}_1(B, *)$$

indexed by $\boldsymbol{M}$-objects (as before, $B = \mathsf{get}(A)$). [4]

b) A family of mappings $\mathsf{put}^{\Leftarrow_1}_A$, which assign to each 2-delta $\delta\colon v_1 \Rightarrow v_2$ between parallel view updates $v_1\colon B \to B'$, $v_2\colon B \to B'$, a mediating update

$$c^\delta = \mathsf{put}^{\Leftarrow_1}_A(\delta)\colon A'_2 \leftarrow A'_1$$

as shown in Fig. 6, where $u_i = \mathsf{put}^{\Leftarrow_0}_A v_i$, $i = 1, 2, 12$.

c) A family of mappings $\mathsf{put}^{\Leftarrow_2}_A$, which assign to each 2-delta δ as above, a mediating 2-delta

$$\varepsilon^\delta = \mathsf{put}^{\Leftarrow_2}_A(\delta)\colon u_2 \Leftarrow (u_1; c^\delta)$$

– see Fig. 6 again. Thus, the result of backward propagation is a triangle $u_1 \xrightarrow{(c^\delta, \varepsilon^\delta)} u_2$, and it is easy to see that the three mappings above define a correct graph morphism $\mathsf{put}^{\Leftarrow}_A\colon \boldsymbol{T}A \leftarrow \boldsymbol{T}^{\Leftarrow}B$. In fact, we can specify the construction above by saying that we have a family of graph morphisms.

Definition 1 (**well-behaved** $\mathsf{put}^{\Leftarrow}$)**.** Given a 2-functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ (a *view*), a *backward $\Leftarrow$-propagation* operation is a family of graph moprhisms $\mathsf{put}^{\Leftarrow}_A\colon \boldsymbol{T}_{\boldsymbol{M}}A \leftarrow \boldsymbol{T}^{\Rightarrow}_{\boldsymbol{N}}(A.\mathsf{get})$ indexed by models A from $\boldsymbol{M}$. Below we write B for $A.\mathsf{get}$.

This family is called *well-behaved (wb)*, if it satisfies the following *PutGet law* for any model $A \in \boldsymbol{M}$ and any 2-delta $\delta\colon v_1 \Rightarrow v_2$ considered as a triangle in $\boldsymbol{T}^{\Rightarrow}_{\boldsymbol{N}}B$:

(PutGet)$^{\Leftarrow}_A$ $\qquad (\mathsf{put}^{\Leftarrow}_A\delta).\mathsf{get} = \delta.$

[4] Index 0 means to recall that the operands of the operation are objects/0-cells in the respective triangle categories (although 1-cells in the model space $\boldsymbol{N}$)

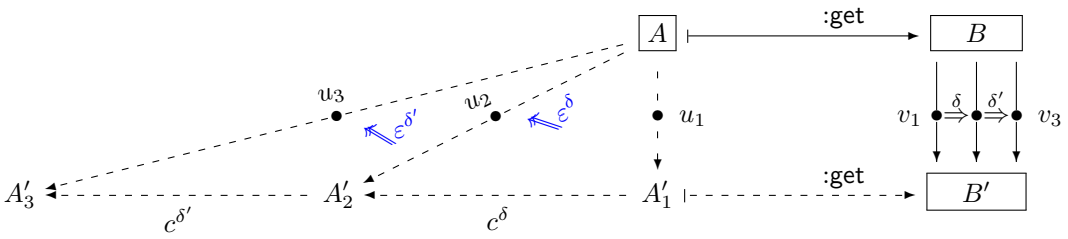


Figure 6: Put and the 2-structure

In detail, the equality above means

$(\text{PutGet})^{\Leftarrow 0}_{A}$ $\quad (\mathsf{put}^{\Leftarrow 0}_{A} v).\mathsf{get} = v$ for any $v \in \boldsymbol{N}_1(B, *)$;

$(\text{PutGet})^{\Leftarrow 1}_{A}$ $\quad (\mathsf{put}^{\Leftarrow 1}_{A} \delta).\mathsf{get} = \mathsf{id}_{B'}$ (where $B' = v_1^{\bullet} = v_2^{\bullet}$);

$(\text{PutGet})^{\Leftarrow 2}_{A}$ $\quad (\mathsf{put}^{\Leftarrow 2}_{A} \delta).\mathsf{get} = \delta$. □

Functoriality of $\mathsf{put}^{\Leftarrow}_{A}$

Our examples show that in practice $\mathsf{put}^{\Leftarrow}_{A}$ has nice functorial properties shown in Fig. 6 (this is, in fact, transitivity of subsetting). Hence, we can require $\mathsf{put}^{\Leftarrow}_{A}$ to be a functor without being too restrictive for practical applications. Moreover, this functor is normally compatible with reindexing.

Definition 2. Operation $\mathsf{put}^{\Leftarrow}$ is called *very well-behaved (vwb)*, if it satisfies the following Stability (identity preservation) and PutPut (2-composition preservation) laws specified below.

For any model A and update $v\colon B \to *$, we require

$(\text{Id})^{\Leftarrow 1}_{A}$ $\quad \mathsf{put}^{\Leftarrow 1}_{A}(\mathsf{id}_v) = \mathsf{id}_{u^{\bullet}}$, where $u = \mathsf{put}^{\Leftarrow 0}_{A} v$ and $u^{\bullet}$ is its target model.

$(\text{Id})^{\Leftarrow 2}_{A}$ $\quad \mathsf{put}^{\Leftarrow 2}_{A}(\mathsf{id}_v) = \mathsf{id}_u$, where $u = \mathsf{put}^{\Leftarrow 0}_{A} v$.

Below we will encode these two equations as the *Stability* law for $\mathsf{put}^{\Leftarrow}$: for all $A \in \boldsymbol{M}_0$,

$(\text{Id})^{\Leftarrow}_{A}$ $\quad \mathsf{put}^{\Leftarrow}_{A}(\mathsf{id}_v) = \mathsf{id}_u$, where $u = \mathsf{put}^{\Leftarrow 0}_{A} v$.

We also require, for any model $A \in \boldsymbol{M}$ and two consecutive 2-deltas $\delta\colon v_1 \Rightarrow v_2$, $\delta'\colon v_2 \Rightarrow v_3$ in $\boldsymbol{T}^{\Rightarrow}_{\boldsymbol{N}} B$ as shown in Fig. 6, the following two equations

$(\text{PutPut})^{\Leftarrow 1}_{A}$ $\quad \mathsf{put}^{\Leftarrow 1}_{A}(\delta; \delta') = \mathsf{put}^{\Leftarrow 1}_{A}\delta \,;\, \mathsf{put}^{\Leftarrow 1}_{A}\delta'$

$(\text{PutPut})^{\Leftarrow 2}_{A}$ $\quad \mathsf{put}^{\Leftarrow 2}_{A}(\delta; \delta') = (\mathsf{put}^{\Leftarrow 2}_{A}\delta \,;\, \mathsf{put}^{\Leftarrow 1}_{A}\delta') \,;\, \mathsf{put}^{\Leftarrow 2}_{A}\delta'$

Below we will encode these two equations as the *PutPut* law for $\mathsf{put}^{\Leftarrow}$: for all $A \in \boldsymbol{M}_0$,

$(\text{PutPut})^{\Leftarrow}_{A}$ $\quad \mathsf{put}^{\Leftarrow}_{A}(\delta; \delta') = \mathsf{put}^{\Leftarrow}_{A}\delta \,;\, \mathsf{put}^{\Leftarrow}_{A}\delta'$

Finally, for any A and update $B \xrightarrow{v} B'$, we require the following diagram to commute:

$$\begin{array}{ccc} \boldsymbol{T}A & \xleftarrow{\mathsf{put}^{\Leftarrow}_{A}} & \boldsymbol{T}^{\Rightarrow}B \\ {\scriptstyle u^*}\uparrow & & \uparrow{\scriptstyle v^*} \\ \boldsymbol{T}A' & \xleftarrow{\mathsf{put}^{\Leftarrow}_{A'}} & \boldsymbol{T}^{\Rightarrow}B' \end{array} \tag{8}$$

□

The data above can be compactly specified as follows.

Lemma 1. *Given a view 2-functor* $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$*, a very well-behaved backward* $\Leftarrow$*-propagation is a (strict) natural 2-transformation* $\mathsf{put}^{\Leftarrow}\colon \boldsymbol{T}_{\boldsymbol{M}} \leftarrow \mathsf{get}; \boldsymbol{T}^{\Rightarrow}_{\boldsymbol{N}}$ *between triangle functors defined above, which satisfies the (PutGet) law.*

4.2.3 Backward propagation of update composition: Toward Lax Putput laws

Operation $\mathsf{put}^{\Leftarrow}$ provides us with propagation of 1-deltas (updates) and 2-deltas, but does not give us anything for fixing the problems of strict Putput. This is the goal of operation $\mathsf{put}^{\triangle}$, which provides any composable pair of view updates, i.e., a commutative triangle on the view side, with a complex (non-commutative) triangle including a comparison construct on the source side (see triangle $AA''A^{\bowtie}$ in Fig. 5).

Arity of $\triangle$-propagation

Given a 2-functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$, we define a *backward propagation* operation $\mathsf{put}^{\triangle}$ as the following three family of mappings.

a) A family of mappings

$$\mathsf{put}^{\triangle 0}_A\colon \boldsymbol{M}_1(A,*) \leftarrow \boldsymbol{N}_1(B,*), \text{ where } B \text{ stands for } \mathsf{get}(A)$$

indexed by $\boldsymbol{M}$-objects, and satisfying the (PutGet) law. (We may assume that this family is borrowed from $\mathsf{put}^{\Leftarrow}_A$ above, or define it here independently and later require that operations $\mathsf{put}^{\triangle 0}_A$ and $\mathsf{put}^{\Leftarrow 0}_A$ coincide for all A.)

b) A family of mappings $\mathsf{put}^{\triangle 1}_A$, which assign to each pair of composable view updates $v_1\colon B \to B_1$, $v_2\colon B_1 \to B_2$ a mediating update

$$m_{12} = \mathsf{put}^{\triangle 1}_A(v_1, v_2)\colon A_{12} \leftarrow A_2$$

as shown in Fig. 7(b), where $u_i = \mathsf{put}_A v_i$, $i = 1, 2, 12$.

c) A family of mappings $\mathsf{put}^{\triangle 2}_A$, which assign to each pair of composable view updates $v_1\colon B \to B_1$, $v_2\colon B_1 \to B_2$, a mediating 2-delta

$$\varepsilon_{12} = \mathsf{put}^{\triangle 2}_A(v_1, v_2)\colon u_{12} \Leftarrow (u_1; u_2); m_{12}$$

– see Fig. 7(b) again, where $b_{12} = u_2; m_{12}$; owing to associativity of arrow composition, 2-arrow ε_{12} can be interpreted as mediating delta $\varepsilon_{12}\colon u_{12} \Leftarrow u_1; b_{12}$ in the back-side triangle AA_1A_{12}.

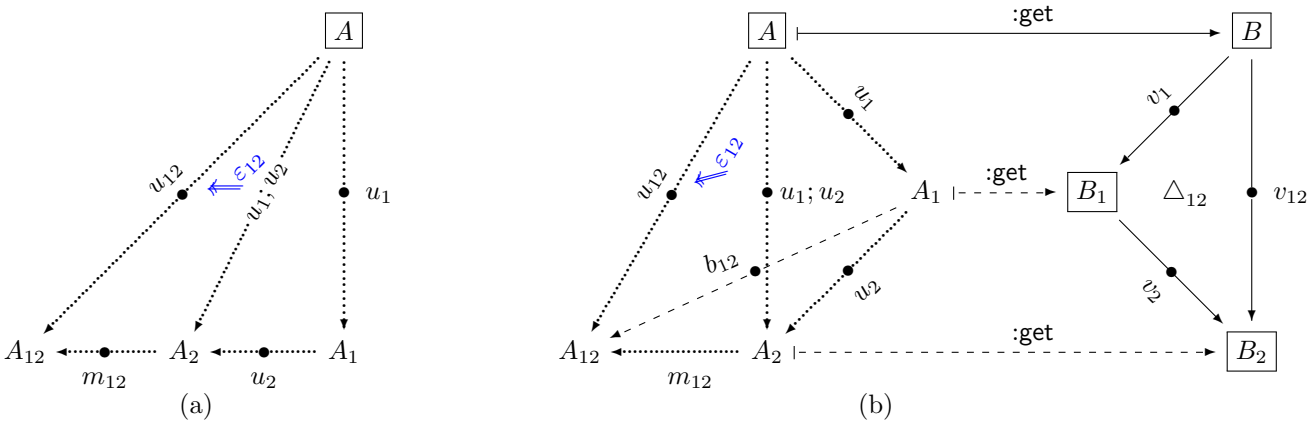


Figure 7: Comparison arrows for Putput

As Fig. 7(b) shows, configuration of arrows produced by $\mathsf{put}^{\triangle}_A$ can be seen as a pyramid. Figure 7(a) presents a planar layout of the same diagram, which shows that a pyramid is just a pair of composable triangles in $\boldsymbol{T}A$: the first is commutative with base u_2, and the second is non-commutative with base m_{12} and delta ε_{12}. The back side triangle of the pyramid (with base b_{12}) is exactly the composition of the two front triangles. The construction makes sense for any 2-category $\boldsymbol{C}$.

Definition 3 (pyramid categories)**.** Let $\boldsymbol{C}$ be a 2-category, $A \in \boldsymbol{C}_0$ and $u, u' \in \boldsymbol{C}_1(A,*)$ are two arrows with the same source. Like triangles, pyramids are special arrows between such arrows, to wit. A *pyramid* $\pi\colon u \Rightarrow u'$ is a triple (b_1, b_2, ε) with $b_1\colon u^{\bullet} \to X$, $b_2\colon X \to u'^{\bullet}$ for some object $X \in \boldsymbol{C}_0$ called the *first* and *second base* of π resp., and $\varepsilon\colon u; b_1; b_2 \Rightarrow u'$ called the *delta* of π.

Given two composable pyramids, $\pi = (b_1, b_2, \varepsilon)\colon u \Rightarrow u'$ and $\pi' = (b'_1, b'_2, \varepsilon')\colon u' \Rightarrow u''$, their *composition* is a pyramid $\pi'' = (b''_1, b''_2, \varepsilon'')\colon u \Rightarrow u''$ with $b''_1 = b_1; b_2; b'_1$, $b''_2 = b'_2$, and $\varepsilon'' = \varepsilon; b'_1; b'_2; \varepsilon'$. For any $u \in \boldsymbol{C}_1(A,*)$, the *identity pyramid* $u \Rightarrow u$ is the triple $(\mathsf{id}_{u^{\bullet}}, \mathsf{id}_{u^{\bullet}}, \mathsf{id}_u)$.[5] □

[5] We have defined composition by taking $X'' = X'$, but there are two other natural ways with $X'' = X$ or $X'' = u'^{\bullet}$. The one we have chosen works better for our further considerations.

It is straightforward to check that composition is associative and identity pyramids are units. Hence, for any 2-category $\boldsymbol{C}$ and any object $A \in \boldsymbol{C}_0$, we have a category $\boldsymbol{P}_{\boldsymbol{C}}A$ of pyramids over A. A functorial arrangement of the construction can be done in parallel with that for triangle categories in Sect. 4.2.1

It's easy to see that the three mappings $\mathsf{put}^{\triangle i}_A$, $i = 0, 1, 2$ above make a correct graph morphism $\mathsf{put}^{\triangle}_A\colon \boldsymbol{P}_{\boldsymbol{M}}A \leftarrow \boldsymbol{T}^{=}_{\boldsymbol{N}}B$ (below we will omit the model spaces subindexes and write $\mathsf{put}^{\triangle}_A\colon \boldsymbol{P}A \leftarrow \boldsymbol{T}^{=}B$). In fact, we can specify the construction of these mappings by saying that we have a family of graph morphisms.

Definition 4 (**well-behaved** $\mathsf{put}^{\triangle}$)**.** Given a view 2-functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$, a *backward $\triangle$-propagation* is a family of graph morphisms $\mathsf{put}^{\triangle}_A\colon \boldsymbol{P}A \leftarrow \boldsymbol{T}^{=}B$ indexed by models A from $\boldsymbol{M}$ (as before, we write B for $A.\mathsf{get}$).

This family is called *well-behaved (wb)*, if it satisfies the following PutGet law for any model $A \in \boldsymbol{M}$ and any commutative triangle $v_1 ; v_2 = v_{12}$ in $\boldsymbol{T}^{=}B$ denoted by $\triangle_{12}$:

$(\mathrm{PutGet})^{\triangle}_A$ $\quad (\mathsf{put}^{\triangle}_A\triangle_{12}).\mathsf{get}_A = \triangle_{12}$ (where get_A is the triangle functor generated by get—see Sect. 4.2.1).

In detail, the equality above means (Fig. 7(b) illustrates the incidence conditions)

$(\mathrm{PutGet})^{\triangle 0}_A$ $\quad (\mathsf{put}^{\triangle 0}_A v).\mathsf{get} = v$ for any $v \in \boldsymbol{N}_1(B, *)$;

$(\mathrm{PutGet})^{\triangle 1}_A$ $\quad (\mathsf{put}^{\triangle 1}_A\triangle_{12}).\mathsf{get} = \mathsf{id}_{B_2}$ (where $B_2 = v_2^{\bullet}$);

$(\mathrm{PutGet})^{\triangle 2}_A$ $\quad (\mathsf{put}^{\triangle 2}_A\triangle_{12}).\mathsf{get} = \mathsf{id}_{v_{12}}$ (where $v_{12} = v_1; v_2$). □

Note that equations $(\mathrm{PutGet})^{\Leftarrow 0}_A$ and $(\mathrm{PutGet})^{\triangle 0}_A$ coincide: they both state the PutGet law for updates (i.e., objects in the triangle categories).

Functoriality of $\mathsf{put}^{\triangle}$

It is clear that operation $\mathsf{put}^{\triangle}$ maps identity triangles ($v_2 = \mathsf{id}_{B_1}$) to identity, but its compatibility with composition is much more intricate than for $\mathsf{put}^{\Leftarrow}$. Some details and discussion could be found in [4] but are omitted here due to space limitations. Briefly, functoriality of $\mathsf{put}^{\triangle}$ turns out closely related to associativity, and provides a sort of associator on the source side, when updates are propagated.

Summary

We can define a *well-behaved lax (asymmetric delta) lens* from a 2-category $\boldsymbol{M}$ to 2-category $\boldsymbol{N}$ as a triple $(\mathsf{get}, \mathsf{put}^{\triangle}, \mathsf{put}^{\Leftarrow})$, in which $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ is a 2-functor, and $\mathsf{put}^{\triangle}$, $\mathsf{put}^{\Leftarrow}$ are families of mappings inverse to get in the sense specified above. The two families are required to be mutually compatible in that mappings $\mathsf{put}^{\triangle 0}_A$ and $\mathsf{put}^{\Leftarrow 0}_A$, which propagate view updates to the source side, are equal for all $A{\in}\boldsymbol{M}_0$. Other components of the two put families are different and play different roles: $\mathsf{put}^{\triangle 1}$ and $\mathsf{put}^{\triangle 2}$ provide mediation arrows to ensure lax Putput, while $\mathsf{put}^{\Leftarrow 1}$ and $\mathsf{put}^{\Leftarrow 2}$ ensure accurate propagation of 2-deltas also subject to a lax discipline. In a proper categorical framework, the two operations can hopefully be integrated into one 2-categorical construct still to be found.

We call a lax lens *very* well-behaved if both $\mathsf{put}^{\triangle}$ and $\mathsf{put}^{\Leftarrow}$ enjoy functoriality wrt. triangle and pyramid categories involved, which is necessary for coherent working of the 2-category machinery. A clean categorical elaboration of $\mathsf{put}^{\triangle}$'s functoriality is also an important future work.

5 Related work and historical remarks: instead of conclusion

Overall, the present paper continues Johnson and Rosebrugh's program (stated very early in several talks, lectures and recently in [13]) of building bx foundations by progressing from poor to richer categories employed for modelling model spaces. In the sequence of Sets (state-based lenses), Posets (Hegner's framework [11]), Cats (delta-lenses), 2-Cats appear as a very natural next step.

More specifically, Putput problems were noticed as early as in the first classical lens papers [9, 14], and continued to be remarked later, e.g., in [15]. In [8], the relational update composition was proposed as a means to mitigate Putput problems, but it was also noticed that it does not solve all problems. The first investigation of Putput based on solid mathematical foundations was undertaken in [12], where (as mentioned in the introduction) Johnson and Rosebrugh found sufficient and necessary conditions for the relational (they say mixed) Putput to hold. Their analysis seems to be the maximum of what could be done for Putput in the strict (i.e., equality-based) setting. That paper can be seen as the first chapter in the long Putput story (with the previous results classified as prehistorical). The present paper owns to paper [12] the categorical approach to Putput, attention to details, and the very genre of a Putput story. Hopefully, it begins its second – lax – chapter.

Acknowledgement. Thanks go to anonymous referees for careful reading the first submitted version of the paper despite numerous typos (for which the author is really sorry), suggestions, and encouragement. I am also grateful to Michael Johnson and Robert Rosebrugh for many stimulating discussions of delta lenses in general, and Putput in particular.

References

[1] A. Anjorin and J. Gibbons, editors. *Proceedings of the 5th International Workshop on Bidirectional Transformations, Bx 2016, co-located with The European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 8, 2016*, volume 1571 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2016.

[2] Z. Diskin. Model Synchronization: Mappings, Tiles, and Categories. In J. M. Fernandes, R. Lämmel, J. Visser, and J. Saraiva, editors, *GTTSE*, volume 6491 of *Lecture Notes in Computer Science*, pages 92–165. Springer, 2009.

[3] Z. Diskin. Asymmetric Delta-Lenses with Uncertainty: Towards a Formal Framework for Flexible BX. Technical Report GSDLab-TR 2016-03-01, University of Waterloo, 2016. http://gsd.uwaterloo.ca/node/660.

[4] Z. Diskin. Compositionality of update propagation: Lax Putput. Technical Report GSDLab-TR 2017-02-28, McMaster University, 2017. http://gsd.uwaterloo.ca/node/691.

[5] Z. Diskin, R. Eramo, A. Pierantonio, and K. Czarnecki. Incorporating uncertainty into bidirectional model transformations and their delta-lens formalization. In Anjorin and Gibbons [1], pages 15–31.

[6] Z. Diskin, H. Gholizadeh, A. Wider, and K. Czarnecki. A three-dimensional taxonomy for bidirectional model synchronization. *Journal of System and Software*, 2015. In Press. Online at doi:10.1016/j.jss.2015.06.003.

[7] Z. Diskin, T. Maibaum, and K. Czarnecki. Intermodeling, queries, and kleisli categories. In *Fundamental Approaches to Software Engineering*, pages 163–177. Springer, 2012.

[8] Z. Diskin, Y. Xiong, and K. Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology*, 10:6: 1–25, 2011.

[9] J. Foster, M. Greenwald, J. Moore, B. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In *POPL*, pages 233–246, 2005.

[10] H. Gholizadeh, Z. Diskin, S. Kokaly, and T. Maibaum. Analysis of source-to-target model transformations in quest. In J. Dingel, S. Kokaly, L. Lucio, R. Salay, and H. Vangheluwe, editors, *Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 28, 2015.*, volume 1500 of *CEUR Workshop Proceedings*, pages 46–55. CEUR-WS.org, 2015.

[11] S. J. Hegner. An order-based theory of updates for closed database views. *Ann. Math. Artif. Intell.*, 40(1-2):63–125, 2004.

[12] M. Johnson and R. D. Rosebrugh. Lens put-put laws: monotonic and mixed. *ECEASST*, 49, 2012.

[13] M. Johnson and R. D. Rosebrugh. Unifying set-based, delta-based and edit-based lenses. In *5th Int. Workshop on Bidirectional Transformations, Bx 2016*, volume 1571 of *CEUR Workshop Proceedings*, 2016.

[14] P. Stevens. Bidirectional model transformations in qvt: semantic issues and open questions. *Software & Systems Modeling*, 9(1):7–20, 2010.

[15] Y. Xiong, H. Song, Z. Hu, and M. Takeichi. Synchronizing concurrent model updates based on bidirectional transformation. *Software and System Modeling*, 12(1):89–104, 2013.