GSDLAB TECHNICAL REPORT

From State- to Delta-based Bidirectional Model Transformations: the Symmetric Case

Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki,
Hartmut Ehrig, Frank Hermann, and Fernando Orejas

GSDLAB–TR 2011–05–03 May 2011

Generative Software
Development Lab

Generative Software Development Laboratory
University of Waterloo
200 University Avenue West, Waterloo, Ontario, Canada N2L 3G1

WWW page: http://gsd.uwaterloo.ca/

From State- to Delta-based Bidirectional Model Transformations: the Symmetric Case

Zinovy Diskin[1], Yingfei Xiong[1], Krzysztof Czarnecki[1], Hartmut Ehrig[2], Frank Hermann[2,3], and Fernando Orejas[4]

[1] Generative Software Development Lab, University of Waterloo, Canada
{zdiskin,yingfei,kczarnec}@gsd.uwaterloo.ca
[2] Institut für Softwaretechnik und Theoretische Informatik,
Technische Universität Berlin, Germany, ehrig@cs.tu-berlin.de
[3] Interdisciplinary Center for Security, Reliability and Trust,
Université du Luxembourg, Frank.Hermann@uni.lu
[4] Departament de Llenguatges i Sistemes Informàtics,
Universitat Politècnica de Catalunya, Barcelona, Spain, orejas@lsi.upc.edu

Abstract. A bidirectional transformation (BX) keeps a pair of interrelated models synchronized. Symmetric BXs are those for which neither model in the pair fully determines the other. We build two algebraic frameworks for symmetric BXs, with one correctly implementing the other, and both being delta-based generalizations of known state-based frameworks. We identify two new algebraic laws—weak undoability and weak invertibility, which capture important semantics of BX and are useful for both state- and delta-based settings. Our approach also provides a flexible tool architecture adaptable to different user's needs.

1 Introduction

Keeping a system of models mutually consistent (model synchronization) is vital for model-driven engineering. In a typical scenario, given a pair of inter-related models, changes in either of them are to be propagated to the other to restore consistency. This setting is often referred to as bidirectional model transformation (BX) [3].

As noted by Stevens [14], despite early availability of several BX tools on the market, they did not gain much user appreciation because of semantic issues. Indeed, to avoid surprises, a user should clearly understand the behavior of synchronization procedures implemented by the tool. To formalize the semantics of BX tools and guide their implementation, algebraic frameworks for BX have been studied intensively [7, 14, 6, 18, 11].

The majority of algebraic BX frameworks (including all those cited above) are *state-based*. Synchronizing operations take the states of models before and after update as input, and produce new states of models as output. This design assumes that model alignment, i.e., discovering relations (*deltas*) between models, is done by update propagating procedures themselves. Hence, two quite different

operations—heuristics-based delta discovery and algebraic delta propagation—are merged, which causes several theoretical and practical problems [2, 5]; we will discuss them in Section 2.2 after considering several basic examples.

To separate delta discovery and propagation, several researchers proposed to build *delta-based* frameworks [4, 2, 5, 10], in which propagation operations use deltas as input and output rather than compute them internally. Such frameworks (a general one [5] and a tree-oriented [2]) have been built for the *asymmetric* BX case, in which one model in the pair is a view of the other and hence does not contain any new information. In practice, however, it is often the case that two models share some information but each of them contains something new not present in the other; following [10], we call this case *symmetric* BX. The symmetric case has been considered in the state-based setting [12, 14, 6, 10], yet a precise delta-based symmetric framework has been an open issue.

In this paper, we fill the gap and develop a delta-based framework for symmetric BX. We build two algebraic structures, *symmetric delta lenses* and *(consistency) maintainers*, which comprise delta-based synchronization operations and laws they must satisfy. Lenses are more abstract and specify an interface of a model synchronization tool; maintainers are closer to implementation and allow the tool to reuse an infrastructure for delta composition. We show that 1) a lens can be built from a maintainer, and 2) the lens's laws are derived from the maintainer's laws so that a desirable lens's behavior is guaranteed when the lens is implemented by a suitable maintainer.

The second major contribution of the paper is the introduction of two new algebraic laws: weak invertibility and weak undoability. A long-standing problem in existing symmetric BX frameworks is that the basic laws (correctness and Hippocraticness [12, 14]) are not enough to ensure reasonable BX behavior, whereas more advanced laws like undoability [14] and invertibility [6] are known to be too strong and exclude many quite practical BXs. Our new laws solve this problem by reshaping strong laws into a weaker form that allows for reasonable symmetric BXs and yet prohibits BXs with unwanted behavior.

The paper is organized as follows. Section 2 analyzes an example and identifies three problems of state-based BXs that motivate our work on delta-based BXs. We present sd-lenses in Section 3 and maintainers in Section 4. Section 5 discusses relation to state-based frameworks, Section 6 discusses related work, and Section 7 concludes the paper.

2 The need for deltas

We begin with an example showing how state-based frameworks work and what their problems are. Then we explain why delta-based frameworks are needed.

2.1 Example

Figure 1 presents two related models A and B. The former specifies a class of Persons with their names and birth years, and the latter specifies Employees

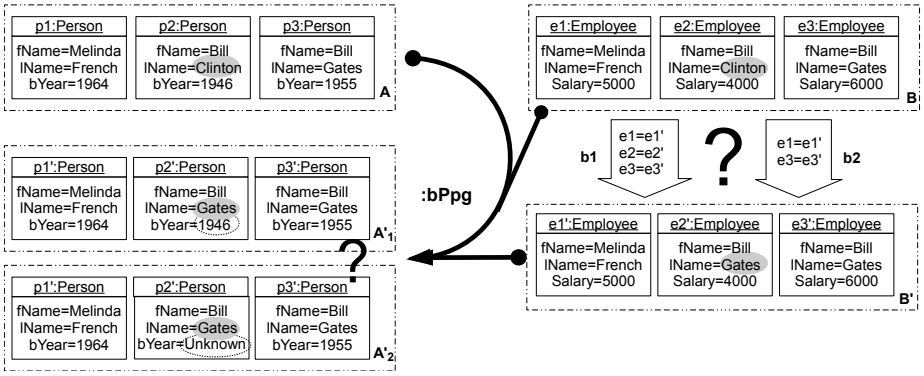


Fig. 1: The need of vertical deltas (updates)

with their names and salaries. Two models are considered consistent if the correspondence between Persons and Employees, inferred from the equality of their full names, is bijective. Initially models A and B are consistent, but then B is modified into B' and we need to propagate the change to the A side.

A suitable state-based BX framework designed for this task is trigonal systems [6]. Changes between the two sides are propagated by two ternary operations: forward propagation fPpg and backward propagation bPpg. When model B changes to B', operation bPpg takes the updated model B' and the original models B, A, and produces an updated model $A' = \mathsf{bPpg}(B', B, A)$. Forward propagation fPpg works similarly: $B' = \mathsf{fPpg}(A', A, B)$.

Figure 1 shows that two reasonable interpretations of the updated model B' are possible. Object $e2'$ may be understood as either a renamed version of $e2$, or a new object inserted into the model while $e2$ is deleted. The difference can be formally captured by specifying sets of pairs $(e, e') \in B \times B'$ with e and e' considered to represent the same object; we call this set $\simeq_v \subset B \times B'$ a *(vertical) sameness* relation. A triple $b = (B, \simeq_v, B')$ is called an *update delta* from B to B' and we write $b\colon B \to B'$. From $\simeq_v$ we can infer which objects were deleted, inserted, or modified. For example, $e2$ is deleted by delta $b2$ because it is not included in $b2$, but it is modified by $b1$ because it is declared to be the same as $e2'$ and the last names in $e2$ and $e2'$ are different.

Now we observe that two different deltas, $b1$ and $b2$, lead to two different synchronization results. To see that, we first define a correspondence between models A and B via full names of objects, i.e., we set a *(horizontal) sameness* relation $\simeq_h \subset A \times B$ between models A and B; in our case, it consists of three pairs (pi, ei), $i = 1, 2, 3$. Propagating delta $b1$ to the A side results in model A'_1: as objects $p2$ and $e2$, $e2$ and $e2'$ are the same, we merely apply modification of $e2$ to $p2$. However, propagation of delta $b2$ leads to model A'_2, which differs from A'_1 in the value of bYear: as object $e2$ is deleted and $e2'$ is inserted, object $p2$ is deleted and A-counterpart of $e2'$ — a new object $p2'$ — is inserted, but its birth date is unknown. Thus, propagation essentially depends on deltas, and

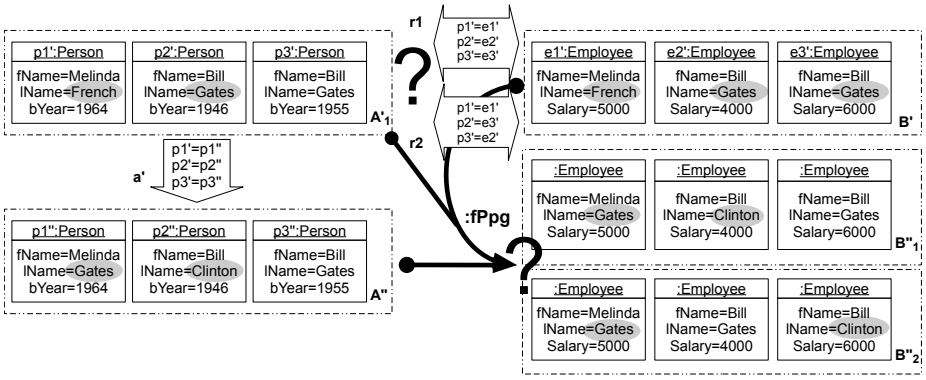
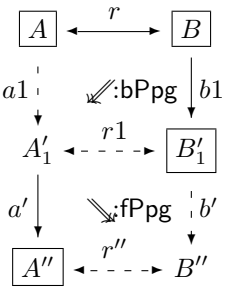


Fig. 2: The need of horizontal delta (correspondence)

propagation operation bPpg has to compute them using some heuristics, and then propagate the change.

To unify terminology and notation, we call a triple $r = (A, \simeq_h, B)$ a *correspondence* or *horizontal* delta from A to B and write $r\colon A \leftrightarrow B$; update deltas are *vertical*. Importantly, the same models A and B may have different correspondence deltas between them. For example, suppose that a user reviews the updated model $A1'$ and discovers that the change is mistaken: it is Melinda French who gets married and changes her last name, but not Bill Clinton. Then the user changes names of objects $p1$ and $p2$ to, respectively, Melinda Gates and Bill Clinton, as shown in Fig. 2 with update delta $a'\colon A'_1 \to A''$. To propagate the update to the B-side, we need to relate models $A1'$ and B' and rename the corresponding Employees. However, because there are two "Bill Gates" in both models, two cases of correspondences, $r1$ and $r2$ in Fig. 2, are possible, which lead to two different results: B''_1 and B''_2. Of course, from the previous propagation we know that the correct delta is $r1$, but since this delta does not explicitly occur in the output of operation bPpg, forward propagation fPpg does not know it and has to infer it from the current states of the models.

2.2 Unweaving delta discovery and propagation



Problems of merging delta discovery into update propagation. First, such a merge, as presented in state-based frameworks, essentially complicates propagation operations and their semantics. Delta discovery is an independent operation with its own laws [1, 15], and is usually far more complex than propagation as such. Weaving delta discovery into update propagation complicates the laws of the latter and makes its behavior less predictable.

Second, it unnecessarily complicates support of update sequences. Indeed, our example can be specified as shown by the inset diagram above (input nodes

are framed and input arrows are solid; output elements are, respectively, non-framed and dashed). It shows that the output horizontal delta $r1$ produced by bPpg must be the input delta for fPpg. However, in a straightforward state-based implementation, operation fPpg computes the delta afresh, which may result in a different delta $r_1' \neq r_1$.

Third, our previous work [5] shows that similar problems appear in sequential composition of BX (think of another BX from B- to C-models) if vertical deltas are replaced by pairs of models, as is done in the state-based frameworks.

A solution to these three problems is to encapsulate delta propagation in a special module, which takes the horizontal and vertical deltas as input, and produces new vertical and horizontal deltas as shown in the inset diagram above; we call such a module a delta-based BX. It has a simple algebraic semantics, prevents erroneous composition of updates and BXs, and allows reusing deltas.

Implementation of deltas. Normally, only small parts of big models are updated, and implementing vertical deltas as sameness relations is very non-economic. A practical solution is to implement them operationally as edit sequences or as overriding deltas [17, 5]. Horizontal deltas can be seen as traceability links, which are maintained by many transformation tools. For either representation, deltas can be abstracted as arrows relating two models.

Managing deltas and tool architecture. Having a separate delta-propagating module provides a flexible tool architecture. For example, the state-based framework can be simulated if deltas are first discovered by a model differencing tool and then passed to the propagation module. If the two models are related by a transformation, horizontal deltas can be inferred from it — this architecture is used in SyncATL [16]. Hybrid interfaces (state-based for one dimension and delta-based for the other) are also possible, e.g., two incremental synchronization tools, based on TGG [8] and QVT [13], take vertical deltas as input and store horizontal deltas internally. An additional advantage of separating delta discovery from propagation is that the user may control the result of differencing and correct it if needed. Finally, if the synchronizer can be tightly coupled with the application, deltas can be obtained by recording the user operations within the applications; in this case, model differencing phase is not needed.

Although the tools mentioned above actually use a separated delta propagation module, they lack a precise specification of both their architecture and semantics of propagation procedures they guarantee. Filling the gap needs a precise definition of delta-based symmetric BX and a formal algebraic theory of delta propagation. Developing both of them is our goal for the rest of the paper.

3 Symmetric delta lenses

We first specify an algebraic structure modeling the very basic properties of update propagation (Section 3.1). Then we enrich the structure with more advanced laws of undoability and invertibility (Section 3.2).

3.1 The basic structure

We begin by defining the space of models and their vertical deltas as a graph with an additional structure representing do-nothing updates and update inversion; this structure makes the graph *reflexive* and *involutive*.

Definition 1 (Model space) A *model space* $\mathbf{A}$ is a graph $(\mathbf{M_A}, \mathbf{\Delta_A}, \$_\mathbf{A})$, whose nodes $A \in \mathbf{M_A}$ are called $\mathbf{A}$-*models*, arrows $a \in \mathbf{\Delta_A}$ are $\mathbf{A}$-model *deltas*, and $\$_\mathbf{A}$ is a quadruple of total unary "bookkeeping" functions $(\square_{\mathbf{A}\_}, \_\square_\mathbf{A}, \mathsf{id}_{\mathbf{A}\_}, \_^{\smile\mathbf{A}})$ (with "_" being the placeholder) providing $\mathbf{A}$ with the structure of reflexive involutive graph explained below.

Functions $\square_{\mathbf{A}}\_, \_\square_\mathbf{A} : \mathbf{\Delta_A} \to \mathbf{M_A}$ provide deltas with their *source* and *target* models resp., and we write $a\colon A \to A'$ if $\square_\mathbf{A} a = A$ and $a\square_\mathbf{A} = A'$. Intuitively, we understand a as a delta resulting from some update to model A, i.e., as a triple $(A, \simeq_v, A')$ like those considered in Section 2.1. By an abuse of terminology, we will often call delta a an update from A to A' (though different sequences of update operations can result in the same delta).

Function $\mathsf{id}_\mathbf{A}\colon \mathbf{A} \to \mathbf{\Delta_A}$ assign to every model A a special *identity* delta $\mathsf{id}_\mathbf{A}A\colon A \to A$ that identically relates A to itself. Such a delta may be thought of as (the result of) an *idle* update to A, which does nothing. To capture this intuition formally, we need to introduce sequential composition of deltas and require $\mathsf{id}_\mathbf{A}$ to be its neutral unit (see [5] for details), but in this paper we do not consider vertical delta composition. However, we will later capture idleness of $\mathsf{id}_\mathbf{A}$-arrows wrt. their composition with horizontal deltas.

Finally, $\_^{\smile\mathbf{A}}$ is an unary operation of *delta inversion*: for $a\colon A \to A'$, arrow $a^{\smile\mathbf{A}}\colon A' \to A$ is the same delta traversed in the opposite direction. For example, the inverse of delta $a = (A, \simeq, A')\colon A \to A'$ in Fig. 2 with $\simeq\ = \{(p1, p1'), (p2, p2'), (p3, p3')\}$ is delta $a^\smile = (A', \simeq^{-1}, A)\colon A' \to A$ with $\simeq^{-1}\ = \{(p1', p1), (p2', p2), (p3', p3)\}$. It can be understood as the delta resulting from undoing update a: changing lNames of $p1'$ and $p2'$ to French and Gates resp.

The following evident laws are required (subscript $\mathbf{A}$ near $\_^\smile$ is omitted):

$$(\mathsf{id}_\mathbf{A}A)^\smile = \mathsf{id}_\mathbf{A}A \text{ for all } A \in \mathbf{M_A} \text{ and } (a^\smile)^\smile = a \text{ for all } a \in \mathbf{\Delta_A},$$

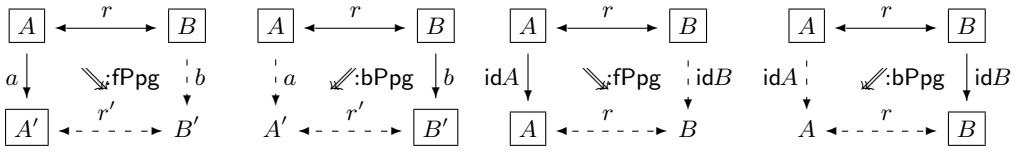
which make operation $\_^\smile$ an *involution* and the graph *involutive*.

Thus, a model space is a reflexive involutive graph.

A model space is also know as a *reflexive involutive graph*, where reflexiveness means that there are identity arrows between nodes, and involution means that arrows have their inverses and the involutive laws hold.

Now we introduce horizontal deltas as arrows between models in two model spaces, and come to the notion of *triple spaces*.

Definition 2 (Triple space) A *triple space* $\mathbf{R}\colon \mathbf{A} \leftrightarrow \mathbf{B}$ or $\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$ consists of a pair of models spaces $(\mathbf{A}, \mathbf{B})$, and a set $\mathbf{R}$ of arrows from $\mathbf{A}$-nodes to $\mathbf{B}$-nodes called *correspondence relations*, or just *corrs*. Formally, $\mathbf{R} = (\mathbf{M_A}, \mathbf{M_B}, \mathbf{\Delta_{AB}}, \$_\mathbf{AB})$ is a graph with $\mathbf{M_A} \cup \mathbf{M_B}$ being the set of nodes, $\mathbf{\Delta_{AB}}$ the set of arrows (corrs), and $\$_\mathbf{AB}$ consists of two functions, $\square_{\mathbf{AB}}\_\colon \mathbf{\Delta_{AB}} \to \mathbf{M_A}$ and $\_\square_{\mathbf{AB}}\colon \mathbf{\Delta_{AB}} \to \mathbf{M_B}$,



(a) fPpg (b) bPpg (c) (IdPpg) law

Fig. 3: Stable sd-lens: operations (a,b) and the law (c)

providing corrs with their *source* and *target* models. For $r \in \mathbf{\Delta_{AB}}$, we write $r\colon A \leftrightarrow B$ if $\square_{\mathbf{AB}} r = A$ and $r\square_{\mathbf{AB}} = B$.

To ease terminology, we will use term 'delta' generically for both updates (*vertical* deltas) and correspondences (*horizontal* deltas). We will also write bookkeeping functions, i.e., components of $\$_{\mathbf{A}}$, $\$_{\mathbf{B}}$, and $\$_{\mathbf{AB}}$ without subscripts.

Remark 1 (on bidirectional arrows). Use of bidirectional arrows does not mean that spaces **A** and **B** play the same role: we still distinguish **A** as the *left* and **B** as the *right* spaces of the triple space. Corrs can be thought of as bidirectional UML-associations between models. In a more refined formal setting, we could introduce corrs going from **B**- to **A**-models, and an involutive operation of inverting a corr. Then a bidirectional corr can be understood as a pair of mutually inverse directed corrs.

Now we define operations modeling update propagation.

Definition 3 (sd-lenses) A *symmetric delta lens (sd-lens)* over a triple space $\mathbf{A} \xleftrightarrow{\mathbf{R}} \mathbf{B}$ is a pair of *forward* and *backward propagation* operations (note that backward propagation arrow goes from right to left)
$\mathsf{fPpg}\colon \mathbf{\Delta_A} \,{}^{\square}\!\times\, \mathbf{\Delta_{AB}} \to \mathbf{\Delta_B} \times_{\square} \mathbf{\Delta_{AB}}$ and $\mathsf{bPpg}\colon \mathbf{\Delta_A} \,{}_{\square}\!\times\, \mathbf{\Delta_{AB}} \leftarrow \mathbf{\Delta_B} \times^{\square} \mathbf{\Delta_{AB}}$
of arities shown in Fig. 3(a,b): input nodes are framed, input arrows are solid, and the output elements are non-framed and dashed. Figure 4 shows an example: operation fPpg takes deltas a and r and produces deltas b and r'.

Symbol ${}^{\square}\!\times$ in the formulas above denotes the subset of the respective Cartesian product consisting of all pairs of arrows with the same source: $\mathbf{\Delta_A} \,{}^{\square}\!\times\, \mathbf{\Delta_{AB}} = \{(a,r) \in \mathbf{\Delta_A} \times \mathbf{\Delta_{AB}}\colon\ \square_{\mathbf{A}} a = \square_{\mathbf{AB}} r\}$, and respectively $\mathbf{\Delta_B} \times_{\square} \mathbf{\Delta_{AB}} = \{(b,r) \in \mathbf{\Delta_B} \times \mathbf{\Delta_{AB}}\colon\ b\square_{\mathbf{B}} = r\square_{\mathbf{AB}}\}$ is the subset of pairs with the same target. Similarly, the meaning of symbols $\times^{\square}$ and ${}_{\square}\!\times$ is defined by diagram Fig. 3(b). We must also require right correspondence of the input and output pairs: for fPpg, if $(b,r') = \mathsf{fPpg}(a,r)$, then $\square b = r\square$ and $\square r' = a\square$, and for bPpg, if $(a,r') = \mathsf{bPpg}(b,r)$, then $\square a = \square r$ and $r'\square = b\square$. We call these and similar equations specifying relationships between arrows *incidence conditions*.

Note that the arity diagrams unambiguously specify all required incidence conditions, and their explicit string-based formulation as above can be omitted. In fact, operations like fPpg and bPpg act upon arrow diagrams, and can

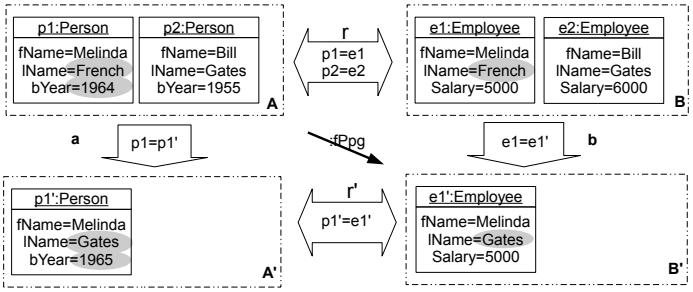


Fig. 4: Example of update propagation

be accurately formalized in terms of *diagram algebra* [4], which allows one to avoid bulky formulation of incidence conditions. Below we will use the arity diagram of an operation as a part of the definition rather than an auxiliary illustration. The reader wishing to translate such definitions into more habitual formulas, can easily do it as explained above. To guide this translation, we will also provide formula-based definitions using polymorphic symbol $\boxtimes$ denoting the subset of the respective Cartesian product defined by the arity diagram; the meaning of this symbol thus depends on the arity diagram. For example, arities above are described as follows: $\mathsf{fPpg}\colon \mathbf{\Delta_A} \boxtimes \mathbf{\Delta_{AB}} \to \mathbf{\Delta_B} \boxtimes \mathbf{\Delta_{AB}}$, $\mathsf{bPpg}\colon \mathbf{\Delta_A} \boxtimes \mathbf{\Delta_{AB}} \leftarrow \mathbf{\Delta_B} \boxtimes \mathbf{\Delta_{AB}}$, plus equations specifying incidence of the output and input arrows. Below we will omit incidence conditions of the latter type: they will be evident from the arity diagrams.

The small double arrows in the middle labeled by :fPpg, :bPpg indicate that the squares are *application instances* of the operations (other instances are are formed by other arguments). In the same manner we could write also $a{:}\mathbf{\Delta_A}$, $r{:}\mathbf{\Delta_{AB}}$ *etc*, but we omit these to avoid too heavy notation.

It is convenient to use also the following notation: for the situation in Fig. 3(a), we write $a.\mathsf{fPpg}(r)$ for b and $r.\mathsf{fPpg}(a)$ for r', and similarly for bPpg. To resolve ambiguity, we always use a, b to denote deltas in $\mathbf{A}, \mathbf{B}$, and r to denote correspondences.

A natural requirement for sd-lenses is that if the input delta changes nothing, the output delta should also change nothing. Formally, we call an sd-lens *stable* if the following law holds for any corr $r\colon A \to B$ (see Fig. 3c):
(IdPpg) $\quad \mathsf{fPpg}(\mathsf{id}A,\mathrm{r})=(\mathsf{id}B,\mathrm{r})$ and $\mathsf{bPpg}(\mathsf{id}B,\mathrm{r})=(\mathsf{id}A,\mathrm{r})$.

The rest of the paper assumes this law holds by default unless the otherwise is explicitly specified.

We write an sd-lens over a triple space $\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$ as a double bidirectional arrow $\boldsymbol{\lambda}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ meaning that the second arrow refers to a pair of operations $(\mathsf{fPpg}, \mathsf{bPpg})$ constituting the lens.

3.2 Invertibility and undoability

A basic requirement for *bidirectional* model synchronization is compatibility of propagation operations between themselves. Given a corr $r\colon A \leftrightarrow B$, an update $a\colon A \to A'$ is propagated into update $b = a.\mathsf{fPpg}(r)$, which can be propagated back to update $a' = b.\mathsf{bPpg}(r)$. For an ideal situation of *strong invertibility*, we should require $a' = a$. Unfortunately, it does not hold in general because **A**-specific part of the information is lost in passing from a to b, and cannot be restored. For example, in Fig. 4 **A**-objects have birth years, which are absent on the **B**-side and hence are lost in a'. However, we could still require invertibility for data shared between A and B. In our example, name changes are shared and will be restored in a'; hence, $a \neq a'$ but $a'.\mathsf{fPpg} = a.\mathsf{fPpg}$. We thus come to the notion of *weak invertibility* of update propagation; it is formalized as follows.

Definition 4 (update equivalence) Given an sd-lens $\boldsymbol{\lambda}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ and a corr $r\colon A \leftrightarrow B$, two updates of model A, $a_1\colon A \to A'_1$ and $a_2\colon A \to A'_2$, are called *r-equivalent* if $a_1.\mathsf{fPpg}(r) = a_2.\mathsf{fPpg}(r)$; we then write $a_1 \sim_r a_2$. Similarly, we introduce *r-equivalence* $b_1 \sim_r b_2$ on **B**-side. (It is easy to see that both relations are indeed equivalence relations.)

Definition 5 (invertible lenses) Operations fPpg and bPpg are *(weakly) invertible* if equations below hold for any $r\colon A \leftrightarrow B$ and all $a\colon A \to A'$, $b\colon B \to B'$:
(fbInv) $\quad a.\mathsf{fPpg}(r).\mathsf{bPpg}(r) \sim_r a$.
(bfInv) $\quad b.\mathsf{bPpg}(r).\mathsf{fPpg}(r) \sim_r b$.

We will call an sd-lens satisfying the laws *invertible*. We show in [9] that invertible sd-lenses can be implemented with triple-graph grammars.

Another important requirement for a reasonable BX is *undoability* discussed by Stevens [14] in the state-based setting. In an ideal situation of *strong undoability*, if update a is first propagated as b and then is cancelled by delta $a^{\smile}\colon A' \to A$, we require a reasonable BX to produce delta $b^{\smile}\colon B' \to B$ to cancel the change on the other side. Unfortunately, it does not hold in general because some information about B may be lost in B' and cannot be restored. For example, Fig. 5 continues the story of Fig. 4 and shows an update $a^{\smile}$ canceling a. According to corr r', a corresponding new object $e2$ (Bill Gates in **B**) should be inserted into model B' and return it back to B. However, since Bill's Salary was lost in B', the propagation of $a^{\smile}$ along r' can only set his Salary to $\mathsf{Unknown}$ thus resulting in a new object $e2''$ and a new model B''. It is a vertical-delta analog of the phenomenon we have just discussed for horizontal deltas, and the strong condition should be again relaxed by considering updates up to their equivalence.

Definition 6 (undoable lenses) An sd-lens is called *(weakly) undoable* if the following *forward-undo* and *backward-undo* laws hold:
(fUndo) $\quad$ Let $(b, r') = \mathsf{fPpg}(a, r)$. Then $a^{\smile}.\mathsf{fPpg}(r') \sim_{r'} b^{\smile}$.
(bUndo) $\quad$ Let $(a, r') = \mathsf{bPpg}(b, r)$. Then $b^{\smile}.\mathsf{bPpg}(r') \sim_{r'} a^{\smile}$.

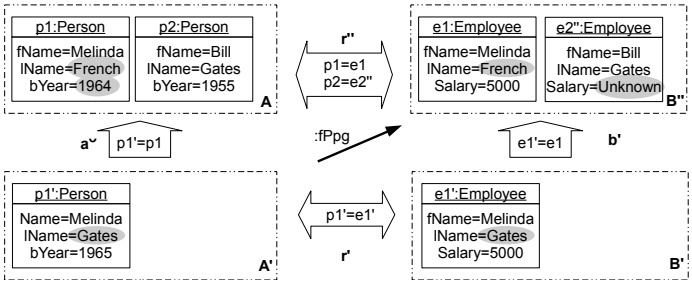


Fig. 5: Undoing update a from Fig. 4

In Appendix, we show that an sd-lens may be (i) invertible but not undoable, (ii) undoable but not invertible, or (iii) invertible and undoable. It means that the two notions are independent and consistent.

To unify terminology, we will call an invertible/undoable lens *horizontally/*resp. *vertically well-behaved (Wb)*. A lens is *well-behaved* if it is both horizontally and vertically Wb. We will also refer to the laws as *horizontal/vertical round-tripping*.

4 Consistency maintenance and alignment

We have seen that a well-behaved sd-lens exhibits a truly BX-behavior. An advantage of the framework is its simplicity yet applicability to practical scenarios. However, simplicity of the sd-lens framework comes for a price.

First, an update propagation in sd-lenses actually consists of two steps, and their coupling prevents the reuse of operations in the implementation. Consider Fig. 6 that shows the case of propagation in Fig. 4 in more details. The first step is to align models A' and B and compute a new (diagonal) correspondence delta $d\colon A' \leftrightarrow B$ based on the original delta r and update $a\colon A \to A'$. We call this operation *forward (re-)alignment* and denote it as fAln. Note that re-alignment is nothing but composition of two deltas (a simple computation), and should not be confused with delta discovery (requiring heuristics). With this reservation, we will call re-alignment just alignment.

The new correspondence d reveals an inconsistency: objects $p1'$ and $e1$ are declared to be the same yet their lName attributes are different. Hence, in the second step consistency must be restored by updating object $e1$ to $e1'$, and thus we produce an update delta $b\colon B \to B'$ and consistent correspondence delta $r'\colon A' \leftrightarrow B'$ from delta d. We call this operation *forward (consistency) restoration*, fRst. Since different restoration operations can be built on top of the same alignment framework, we could reuse alignment operations. However, their reuse cannot be realized within the sd-lens interface, since (re-)alignment operations are woven into update propagation in sd-lenses.

The second problem of the sd-lens interface is related to an important BX requirement — Hippocraticness law of Meertens/Stevens [12, 14]. When model A is updated to A', it may happen that the new diagonal delta d is still consistent

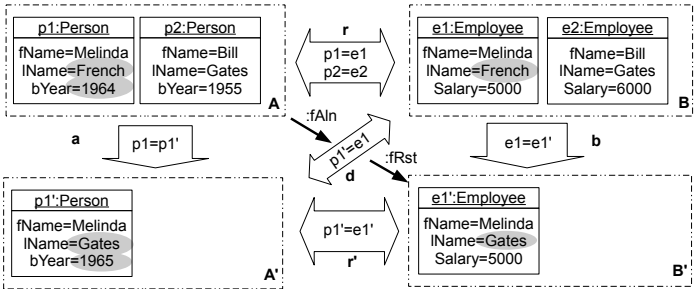


Fig. 6: Two steps in update propagation

and then nothing should be done on the **B**-side. However, since in sd-lenses we have no access to diagonal deltas, we cannot formulate the requirement above.

We call a pair of forward and backward alignment operations an *alignment framework* to stress its basic supporting role for restoration operations built on top of it. We call a pair of forward and backward restoration operations a *maintainer*. Below in this section we formalize the two notions and show that well-behaved maintainers correctly implement well-behaved sd-lenses.

4.1 Alignment taken seriously

We define the notion of alignment framework as a triple space enriched with re-alignment operations.

Definition 7 (Alignment framework) An *alignment framework* over a triple space $\mathbf{R}\colon \mathbf{A} \leftrightarrow \mathbf{B}$ is a couple of operations

$$\mathsf{fAln}\colon \boldsymbol{\Delta}_{\mathbf{A}} \boxtimes \boldsymbol{\Delta}_{\mathbf{AB}} \to \boldsymbol{\Delta}_{\mathbf{AB}} \qquad \text{and} \qquad \mathsf{bAln}\colon \boldsymbol{\Delta}_{\mathbf{AB}} \leftarrow \boldsymbol{\Delta}_{\mathbf{B}} \boxtimes \boldsymbol{\Delta}_{\mathbf{AB}}$$

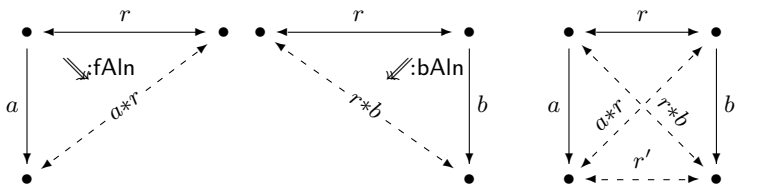
called *forward* and *backward alignment* resp., where symbols $\boxtimes$ denote subsets of the respective Cartesian products consisting of all incident arrows as specified by Fig. 7(a,b) (see p.8). We will also write $a * r$ for $\mathsf{fAln}(a, r)$ and $r * b$ for $\mathsf{bAln}(b, r)$.

Note that computation of the initial corr $r\colon A \leftrightarrow B$ is beyond the framework: the initial alignment (model matching) may need complex heuristic and other context dependent information. Hence, operations above are actually *re*-alignment operations, but we will often call them alignments to ease terminology.

There are two laws. Identity updates do not actually need re-alignment:
(IdAln) $\quad \mathsf{id}A * r = r = r * \mathsf{id}B$
for any corr $r\colon A \to B$.

The result of applying a sequence of interleaving forward and backward alignments does not depend on the order of application as shown in Fig. 7(c):
(AlnAln) $\quad (a * r) * b = a * (r * b)$
for any $a \in \boldsymbol{\Delta}_{\mathbf{A}}, r \in \boldsymbol{\Delta}_{\mathbf{AB}}, b \in \boldsymbol{\Delta}_{\mathbf{B}}$.[5]

[5] We could also directly define alignment as arrow composition (with pre-involution if necessary); then AlnAln law is nothing but associativity.



(a) fAln (b) bAln (c) fAln;bAln = bAln;fAln

Fig. 7: Alignment operations and their laws

We will call diagrams like shown in Fig. 7(a,b,c) *commutative* if the arrow at the respective operation's output is indeed equal to that one computed by the operation. For example, diagram (c) is commutative if $r' = a * r * b$.

Note that in general $r \neq a^{\smile} * (a * r)$ and $r \neq (r * b) * b^{\smile}$ because some information can be lost in delta composition.[6]

We will write an alignment framework as an arrow $\boldsymbol{\alpha}\colon \mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$ meaning that $\boldsymbol{\alpha}$ comprises a triple space $\mathbf{R}$ (one bidirectional arrow) and two operations fAln, bAln over it (the other one).

4.2 Consistency maintainers: Hippocratic update propagation

Definition 8 (maintainers) A *(consistency) maintainer* over an alignment framework $\boldsymbol{\alpha}\colon \mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$ comprises (i) a subclass $\mathbf{K} \subset \boldsymbol{\Delta}_{\mathbf{AB}}$ of *consistent* corrs and (ii) a couple of *consistency restoration* operations

$$\mathsf{fRst}\colon \boldsymbol{\Delta}_{\mathbf{AB}} \to \boldsymbol{\Delta}_{\mathbf{B}} \boxtimes \boldsymbol{\Delta}_{\mathbf{AB}} \quad \text{and} \quad \mathsf{bRst}\colon \boldsymbol{\Delta}_{\mathbf{A}} \boxtimes \boldsymbol{\Delta}_{\mathbf{AB}} \leftarrow \boldsymbol{\Delta}_{\mathbf{AB}}$$

of arities shown in Fig. 8 (a,b): output nodes and arrows are shown blank and dashed resp.

If $(b, r') = \mathsf{fRst}(r)$, we will also write $r|$ for b and $r_{\_}$ for r'; similarly, if $(a, r') = \mathsf{bRst}(b)$, we write $|r$ and $_r$ for a and r'. In composed formulas, bars and underscores always have the highest priority.

A maintainer is called *correct* if its output corrs are always consistent, and are compositions of the original corr with output updates:

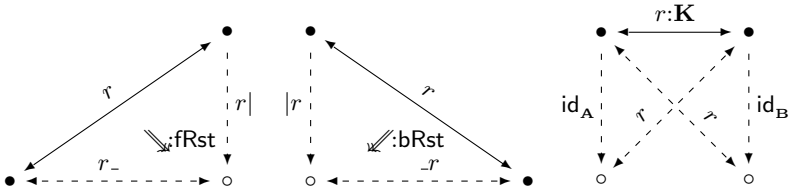
(Corr) $\quad r * r| = r_{\_} \in \mathbf{K} \quad$ and $\quad |r * r = {}_r \in \mathbf{K}$

A maintainer is called *Hippocratic* (we borrow Stevens' term [14]) if it does nothing for an originally consistent corr as shown in Fig. 8(c):

(Hipp) $\quad$ If $r\colon A \to B \in \mathbf{K}$, then $|r = \mathsf{id}A$, $r| = \mathsf{id}B$ and ${}_r = r = r_{\_}$.

We write a maintainer as an arrow $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K} \subset \mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ comprising pairs of operations (fAln,bAln) and (fRst,bRst) over the triple space $\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$.

[6] In a more refined setting with composition of vertical deltas, neither $a; a^{\smile}$ nor $a^{\smile}; a$ are identities.



(a) fRst (b) bRst (c) Hippocraticness

Fig. 8: Consistency restoration operations (a,b) and their laws

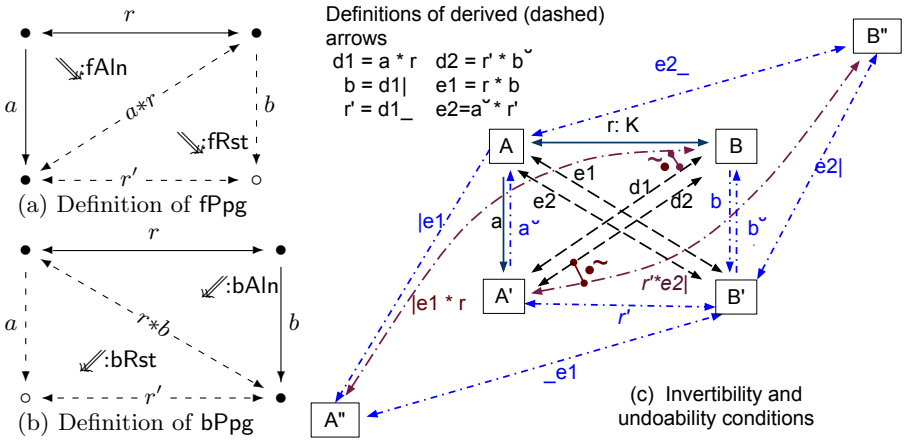


Fig. 9: From maintainers to lenses

4.3 From maintainers to lenses: invertibility and undoability

Maintainers are designed to implement lenses: update propagation operations can be defined via alignment and restoration operations as shown in Fig. 9(a,b).

Definition 9 (from maintainers to lenses) Given a correct maintainer $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K}\subset\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$, we define a lens $\ulcorner\boldsymbol{\mu}\urcorner\colon \mathbf{A} \overset{\mathbf{K}}{\Longleftrightarrow} \mathbf{B}$ by setting

$\mathsf{fPpg}(a, r) \overset{\text{def}}{=} (d|, d\_)$ with $d = a * r$, and $\mathsf{bPpg}(b, r) \overset{\text{def}}{=} (|e, \_e)$ with $e = r * b$.

It is easy to see that lens $\ulcorner\boldsymbol{\mu}\urcorner$ is stable as soon as $\boldsymbol{\mu}$ is Hippocratic. That is, a correct and Hippocratic maintainer implements a stable lens.

Now we want to state conditions for $\boldsymbol{\mu}$ ensuring that the lens $\ulcorner\boldsymbol{\mu}\urcorner$ is well-behaved. Since the notion of update equivalence is crucial here, we first reformulate it as *corr equivalence* in terms of restoration operations.

Definition 10 (corr equivalence) Two corrs with the same target, $r_i\colon A_i \leftrightarrow B$, $i = 1, 2$ are called *forward equivalent* if $r_1| = r_2|$; we write $r_1 \sim_{\bullet} r_2$. Dually, two corrs with the same source $r_i\colon A \leftrightarrow B_i$ are *backward equivalent*, $r_1 \overset{\sim}{\bullet} r_2$, if $|r_1 = |r_2$.

Lemma 1. *Given corr* $r\colon A \leftrightarrow B$ *and updates* $a_i\colon A \to A'_i$,$b_i\colon B \to B'_i$, $i = 1, 2$,

(1) $\quad a_1{*}r \sim_\bullet a_2{*}r$ *(in* $\boldsymbol{\mu}$*)* *iff* $a_1 \sim_r a_2$ *(in* $\ulcorner\boldsymbol{\mu}\urcorner$*)*

(2) $\quad r{*}b_1 \mathrel{\bullet\!\!\sim} r{*}b_2$ *(in* $\boldsymbol{\mu}$*)* *iff* $b_1 \sim_r b_2$ *(in* $\ulcorner\boldsymbol{\mu}\urcorner$*)*

Proof. Immediate from Definition 9 $\square$.

The next step is to substitute operations defined in Definition 9 into Definitions 5 and 6 of invertibility and undoability. The result is the following definition and accompanying diagram Fig. 9(c).

Definition 11 (well-behaved maintainer) (a) A correct maintainer is called *invertible* or *horizontally well-behaved (hWb)* if the following two dual conditions hold for any $r\colon A \leftrightarrow B \in \mathbf{K}$ (see diagram Fig. 9(c) for the first of them):

(fbInv_m) For any $a\colon A \to A'$, let $d1 = a{*}r$, $e1 = r * d1|$. Then $|e1 * r \sim_\bullet d1$

(bfInv_m) For any $b\colon B \to B'$, let $d1 = r{*}b$, $e1 = |d1 * r$. Then $r * e1| \mathrel{\bullet\!\!\sim} d1$

(b) A correct maintainer is called *undoable* or *vertically well-behaved (vWb)* if the following two dual conditions hold for any $r\colon A \leftrightarrow B \in \mathbf{K}$:

(fUndo_m) For any $a\colon A \to A'$, let $d1 = a{*}r$, $b = d1|$, $r' = d1\_$, $d2 = r' * b^\smile$, and $e2 = a^\smile * r'$. Then $d2 \mathrel{\bullet\!\!\sim} r' * e2|$

(bUndo_m) For any $b\colon B \to B'$, let $d1 = r{*}b$, $a = |d1$, $r' = \_d1$, $d2 = a^\smile * r'$, and $e2 = r' * b^\smile$. Then $d2 \sim_\bullet |e2 * r'$

(again see diagram Fig. 9(c) for the first of the conditions).

Details clarifying the meaning of formulas can be found in the long version. The notion of invertible maintainer is implicit in [9], where alignment and restoration operations are realized by TGG-means.

(c) A correct maintainer is called *well-behaved (Wb)* if it is well-behaved both horizontally and vertically.

Lemma 1 and Fig. 9 make proof of the following theorem straightforward.

Theorem 1. *Let* $\boldsymbol{\mu}\colon \mathbf{A} \overset{\mathbf{K}\subset\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ *be a correct maintainer and* $\ulcorner\boldsymbol{\mu}\urcorner\colon \mathbf{A} \overset{\mathbf{K}}{\Longleftrightarrow} \mathbf{B}$ *is the sd-lens derived from it. Then the following holds*

(i) $\ulcorner\boldsymbol{\mu}\urcorner$ *is stable iff* $\boldsymbol{\mu}$ *is Hippocratic.*

(ii) $\ulcorner\boldsymbol{\mu}\urcorner$ *is invertible iff* $\boldsymbol{\mu}$ *is invertible.*

(iii) $\ulcorner\boldsymbol{\mu}\urcorner$ *is undoable iff* $\boldsymbol{\mu}$ *is undoable.*

Proof. (i) is evident. For proving (ii), we first note that equation fbInv_m implies $a.\mathsf{fPpg}(r).\mathsf{bPpg}(r) \sim_r a$ as is easily seen from triangle $AB'A''$ (updates are equivalent because of lemma 1), that is, fbInv_m provides lens' fbInv. The bf-part is proved in the same way. For (iii), we note that fUndo_m provides equality $a^\smile.\mathsf{fPpg}(r') \sim_{r'} b^\smile$: consider triangle $AB'B''$ and apply the lemma again, that is, fUndo_m implies lens' undoability fUndo. The b-part is proved symmetrically. $\square$.

Corollary 1. *A correct maintainer* $\boldsymbol{\mu}$ *implements a well-behaved sd-lens* $\ulcorner\boldsymbol{\mu}\urcorner$ *iff* $\boldsymbol{\mu}$ *is itself well-behaved.*

The theorem shows that heavy definitions of maintainers' laws can be hidden under the hood of the sd-lens framework. The latter thus demonstrates a reasonable trade-off between concreteness and abstraction: it is abstract enough to free the user from the (re-)alignment concerns, yet provides enough flexibility by explicitly including deltas.

5 From delta- to state-based model sync

We will show that state-based frameworks appear as special simple cases of delta-based ones, if model deltas are degenerated into pairs of models. The following notion is central.

Definition 12 (Simple spaces) A graph $\mathbf{A} = (\mathbf{M_A}, \mathbf{\Delta_A}, \$_\mathbf{A})$ is called *simple* if for any pair of nodes (A, A') there is exactly one arrow $a\colon A \to A'$. Hence, we can identify arrows with pairs of nodes and define $\mathbf{\Delta_A} = \mathbf{M_A}\times\mathbf{M_A}$ with $\Box(A, A') \stackrel{\text{def}}{=} A$, and $(A, A')\Box \stackrel{\text{def}}{=} A'$.

Any simple graph is automatically reflexive, $\mathsf{id}A \stackrel{\text{def}}{=} (A, A)$, and involutive, $(A, A')^\smile \stackrel{\text{def}}{=} (A', A)$

Similarly, a correspondence graph $\mathbf{R} = (\mathbf{M_A}, \mathbf{M_B}, \mathbf{\Delta_{AB}}, \$_\mathbf{AB})$ is called simple if there is exactly one arrow $r\colon A \leftrightarrow B$ for any pairs of nodes $A \in \mathbf{M_A}$ and $B \in \mathbf{M_B}$.

A triple space is *simple* if all three graphs are simple; we then write $\mathbf{A} \stackrel{\mathbf{R}}{\leftarrow\!\bullet\!\rightarrow} \mathbf{B}$.

Lemma 2. *A binary relation $\mathbf{R} \subset \mathbf{A} \times \mathbf{B}$ over sets $\mathbf{A}$,$\mathbf{B}$, determines a unique simple triple space with $\mathbf{M_A} = \mathbf{A}$, $\mathbf{M_B} = \mathbf{B}$, $\mathbf{\Delta_A} = \mathbf{A}\times\mathbf{A}$, $\mathbf{\Delta_B} = \mathbf{B}\times\mathbf{B}$, $\mathbf{\Delta_{AB}} = \mathbf{R}$, and evident bookkeeping functions.*

Now we consecutively consider three classes of state-based symmetric Bx: state-based symmetric lenses (a new construct defined here), state-based maintainers (Meertens and Stevens), and state-based complement-based symmetric lenses (Hofmann *et al*), define their well-behavedness, and specify relations with respective delta-constructs over triple spaces.

5.1 From delta- to state-based symmetric lenses

Definition 13 (ss-lenses) A *state-based symmetric lens (ss-lens)* $\lambda\colon \mathbf{A} \stackrel{\mathbf{R}}{\leftarrow\!\bullet\!\rightarrow} \mathbf{B}$ consists of two non-empty sets $\mathbf{A}, \mathbf{B}$, a *consistency relation* $\mathbf{R} \subset \mathbf{A}\times\mathbf{B}$, and two binary operations: $[\_, \_\rangle\colon \mathbf{A}\times\mathbf{R} \to \mathbf{B}$ and $\langle\_, \_]\colon \mathbf{A} \leftarrow \mathbf{B}\times\mathbf{R}$.

To ease notation, we will write AB for a pair $(A, B) \in \mathbf{R}$, $B' = A'.[AB\rangle$ if $B' = [A', AB\rangle$, and $A' = B'.\langle AB]$ if $A' = \langle B', AB]$.

An ss-lens is called *stable* if for any $AB \in \mathbf{R}$, the following state-based counterpart of IdPpg holds:

$(\mathsf{IdPpg}^\bullet)$ $\qquad A.[AB\rangle = B$ and $B.\langle AB] = A.$

An ss-lens is called *correct* if it provides consistency of its output, that is, for all $AB \in \mathbf{R}$, $A' \in \mathbf{A}$, $B' \in \mathbf{B}$, $(A', A'.[AB\rangle) \in \mathbf{R}$ and $(B'.\langle AB], B') \in \mathbf{R}$.

Definition 14 ((Propagation) equivalence) Given an ss-lens $\lambda\colon \mathbf{A} \overset{\mathbf{R}}{\leftarrow\!\bullet\!\rightarrow} \mathbf{B}$ and pair $AB \in \mathbf{R}$, models A_1, A_2 are called *AB-equivalent* if $A_1.[AB\rangle = A_2.[AB\rangle$; we write $A_1 \sim_{AB} A_2$. We define $B_1 \sim_{AB} B_2$ in the same (symmetric) way.

Definition 15 (well-behaved ss-lenses) A stable correct ss-lens is called *(weakly) invertible/(weakly) undoable* if it satisfies the following state-based counterparts of (weak) invertibility/undoability laws:

($\mathsf{Inv}^\bullet$) $A.[AB\rangle.\langle AB] \sim_{AB} A$ and $B.\langle AB].[AB\rangle \sim_{AB} B$.

($\mathsf{fUndo}^\bullet$) Let $B' = A'.[AB\rangle$. Then $B \sim_{A'B'} A.[A'B'\rangle$

($\mathsf{bUndo}^\bullet$) Let $A' = B'.\langle AB]$. Then $A \sim_{A'B'} B.\langle A'B']$

An ss-lens is called *well-behaved (wb)* if it correct, stable, undoable and invertible.

Remark 2. Ss-lenses are close to trigonal systems [6] with totally defined operations. However, invertibility and undoability formulated in [6] are strong.

Theorem 2. *Well-behaved sd-lenses over simple triple spaces and well-behaved ss-lenses are equivalent constructs.*

Proof. The passage from simple well-behaved sd- to well-behaved ss-lenses is straightforward. The converse passage is also easy: given an ss-lens $\lambda\colon \mathbf{A} \overset{\mathbf{R}}{\leftarrow\!\bullet\!\rightarrow} \mathbf{B}$, we build a triple space by lemma 2, and define $\mathsf{fPpg}(AA', AB) \overset{\text{def}}{=} (BB', A'B')$ where $B' = A'.[AB\rangle$; operation bPpg is defined symmetrically. Checking the laws is straightforward. Finally, the two passages are evidently mutually inverse.

Thus, the state-based lens models of BX are nothing but state-based reducts of sd-lens models.

5.2 From delta- to state-based maintainers

In a nutshell, the notion of well-behaved state-based maintainers can be derived from delta-based maintainers in the same way as well-behaved ss-lenses were derived from sd-lenses (Lemma 3 shows how to manage alignment). Then a maintainer-counterpart of theorem 2 can be proved. Here is the details.

Lemma 3. *A simple triple space with $\mathbf{R} = \mathbf{A} \times \mathbf{B}$ becomes an alignment framework if we define $AA' * AB = A'B$ and $AB * BB' = AB'$.*

Proof. The laws of alignment frameworks, IdAln and AlnAln, are evident. □

Definition 16 (Stevens [14]) A *state-based maintainer (s-maintainer)* $\mu\colon \mathbf{A} \overset{\mathbf{K}}{\leftarrow\!\bullet\!\rightarrow} \mathbf{B}$ consists of two sets (model spaces) $\mathbf{A}, \mathbf{B}$, a binary *consistency* relation $\mathbf{K} \subset \mathbf{A} \times \mathbf{B}$ and two binary *restoration* operations $[\_, \_\rangle\colon \mathbf{A} \times \mathbf{B} \to \mathbf{B}$ and $\langle \_, \_]\colon \mathbf{A} \leftarrow \mathbf{A} \times \mathbf{B}$.

S-maintainer is called

– *correct* if for any pair $(A, B) \in \mathbf{A} \times \mathbf{B}$, $(A, [AB\rangle) \in \mathbf{K}$ and $(\langle AB], B) \in \mathbf{K}$;

– *Hippocratic*, if $[AB\rangle = B$ and $A = \langle AB]$ as soon as $(A, B) \in \mathbf{K}$.

– *undoable*, if for any consistent pair $(A, B) \in \mathbf{K}$ and all $A' \in \mathbf{A}$, $B' \in \mathbf{B}$,

$$[A, [A'B\rangle\rangle = B \text{ and } A = \langle\langle AB'], B]$$

As discussed in [14], undoability rarely holds in practice because of the loss of information during updates. To weaken the condition, we need the corresponding notion of model equivalence.

Definition 17 ((Restoration) equivalence) Given an s-maintainer, models $A_1, A_2 \in \mathbf{A}$ are called *equivalent wrt. model* $B \in \mathbf{B}$, if $[A_1 B\rangle = [A_2 B\rangle$; we write $A_1 \sim_B A_2$. Similarly, models B_1, B_2 are *equivalent wrt. model* $A \in \mathbf{A}$, $B_1 \sim_A B_2$, if $\langle AB_1] = \langle AB_2]$.

Definition 18 (undoability and invertibility) A correct s-maintainer is called

(i) *(weakly) undoable* or *vertically Wb* if for any consistent pair $(A, B) \in \mathbf{K}$ and all $A' \in \mathbf{A}$, $B' \in \mathbf{B}$, the following holds:

$(\mathsf{Undo}_m^{\bullet})$ $\qquad [A[A'B\rangle\rangle \sim_{A'} B$ and $A \sim_{B'} \langle\langle AB']B]$

(ii) *(weakly) invertible* or *horizontally Wb* if for any consistent pair $(A, B) \in \mathbf{K}$ and all $A' \in \mathbf{A}$, $B' \in \mathbf{B}$, the following holds:

$(\mathsf{Inv}_m^{\bullet})$ $\qquad \langle A[A'B\rangle] \sim_B A'$ and $B' \sim_A [\langle AB']B\rangle$

(iii) A correct s-maintainer is *well-behaved* if it is Wb both vertically and horizontally.

The following result is analogous to Theorem 2

Theorem 3. *Well-behaved maintainers over simple triple spaces and well-behaved s-maintainers are equivalent constructs.*

Now Theorem 1 together with Theorems 2, 3 provides the following new state-based result.

Corollary 4 *Let* $\mu\colon \mathbf{A} \overset{\mathbf{K}}{\leftrightarrow\!\!\bullet\!\!\Rightarrow} \mathbf{B}$ *be a correct s-maintainer and* $\ulcorner\mu\urcorner\colon \mathbf{A} \overset{\mathbf{K}}{\leftrightarrow\!\!\bullet\!\!\Rightarrow} \mathbf{B}$ *is the correct ss-lens derived from it. Then the following holds*

(i) μ *is Hippocratic iff* $\ulcorner\mu\urcorner$ *is stable.*
(ii) μ *is invertible iff* $\ulcorner\mu\urcorner$ *is invertible.*
(iii) μ *is undoable iff* $\ulcorner\mu\urcorner$ *is undoable.*

5.3 Relation to symmetric lenses by Hofmann *et al* [10].

We first reformulate the notion of symmetric lens in our terms and notation.

Definition 19 (state-based symmetric lenses with complement) A *state-based symmetric lens with complement* or *ssc-lens* in short, consists of three sets $\mathbf{A}, \mathbf{B}, \mathbf{C}$ and two operations $\mathsf{putr}\colon \mathbf{A} \times \mathbf{C} \to \mathbf{B} \times \mathbf{C}$ and $\mathsf{putl}\colon \mathbf{A} \times \mathbf{C} \leftarrow \mathbf{B} \times \mathbf{C}$.

We denote an ssc-lens by an arrow $\lambda\colon \mathbf{A} \overset{\mathbf{C}}{\leftrightarrow\!\!\bullet\!\!\Rightarrow} \mathbf{B}$.

An ssc-lens is called *stable* if the following laws hold:

$(PutRL)$ $\mathsf{putl}(B, C') = (A, C')$ if $\mathsf{putr}(A, C) = (B, C')$
$(PutLR)$ $\mathsf{putr}(A, C') = (B, C')$ if $\mathsf{putl}(B, C) = (A, C')$

It is straightforward to check that a stable ssc-lens is a well-behaved symmetric lens defined in [10] with the only difference: the latter have an element *missing* $\in C$, which is omitted in ssc-lenses. Thus, what is called round-tripping laws PutRL, PutLR in [10] are ours IdPpg laws, while our round-tripping laws (invertibility and undoability) are not considered in [10].

To set relationships between sd-lenses and ssc-lenses, we need the following notion.

Definition 20 (semi-simple triple spaces) A triple space $(\mathbf{A}, \mathbf{R}, \mathbf{B})$ is called *semi-simple* if graphs $\mathbf{A}, \mathbf{B}$ are simple (but nothing special is assumed about $\mathbf{R}$).

Theorem 5. *An sd-lens* $\boldsymbol{\lambda}\colon \mathbf{A} \overset{\mathbf{R}}{\Longleftrightarrow} \mathbf{B}$ *over a semi-simple triple space gives rise to an ssc-lens* $\boldsymbol{\lambda}^{\bullet}\colon \mathbf{A} \overset{\mathbf{R}}{\Leftarrow\!\bullet\!\Rightarrow} \mathbf{B}$. *The latter is stable as soon as lens* $\boldsymbol{\lambda}$ *is such.*

Proof. First of all we define complement $\mathbf{C} = \mathbf{R}$. Then, given $r \in \mathbf{C} = \mathbf{R}$ and $A \in \mathbf{A}$, $B \in \mathbf{B}$, we define $\mathsf{putr}(A, r) = \mathsf{fPpg}(a, r)$ with $a = (\Box r, A)$, and $\mathsf{putl}(B, r) = \mathsf{bPpg}(b, r)$ with $b = (r\Box, B)$. Then sd-lens stability (i.e., IdPpg law) becomes stability in the sense of ssc-lenses (well-behavedness in terms of [10]).

Remark 3. To introduce the notion of *missing* element into the sd-lens framework, we need to add so called *initial objects* to our model spaces (this constructs is well-known in category theory), we leave it for future work.

6 Related Work

In the previous section we have already seen the relation between our framework and state-based frameworks. In this section we discuss other two classes of related work.

Mathematical foundations for building delta-based frameworks (called *tile algebra*) are described in [4]. Diagonal synchronizers specified there are basically sd-lenses that distinguish between consistent and inconsistent corrs at the input of propagation operations; in addition, they are equipped with alignment operations called rematching. However, neither update inversion, nor the round-tripping laws are considered in [4].

7 Conclusion

A delta-based symmetric BX is a synchronization module that does nothing but propagating vertical deltas over horizontal ones; how these deltas are computed and passed to the module is a separate concern. This design provides a flexible architecture and fixes compositional problems of the state-based frameworks. In the paper we built two algebraic frameworks for symmetric delta-based BXs: more abstract sd-lenses that screen simple but tedious re-alignment computations from the user, and closer to implementation maintainers. We found new— weaker—versions of important invertibility and undoability laws, which

do constrain synchronization behavior, and yet do not exclude many practically interesting BXs incompatible with the strong laws considered previously. Our main result shows that an sd-lens can be implemented by a suitable maintainer, and the former is weakly invertible and undoable iff the latter is such.

The framework still lacks lens and maintainer combinators for specifying complex BX in a compositional way. A well-designed set of combinators would make our frameworks practically applicable to the design of BX languages. We leave it for future work.

Acknowledgment. We are grateful to Michał Antkiewicz, Leo Passos and Arif Wider for discussion and fruitful comments. Financial support was provided by the Ontario Research Fund and NSERC.

References

1. Alanen, M., Porres, I.: Difference and union of models. In: UML. pp. 2–17 (2003)
2. Barbosa, D.M.J., Cretin, J., Foster, N., Greenberg, M., Pierce, B.C.: Matching lenses: alignment and view update. In: ICFP. pp. 193–204 (2010)
3. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional transformations: A cross-discipline perspective. In: ICMT (2009)
4. Diskin, Z.: Model synchronization: mappings, tile algebra, and categories. In: Post-proc. GTTSE 2009. pp. 92–165 (2011)
5. Diskin, Z., Xiong, Y., Czarnecki, K.: From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. Journal of Object technology 10, 6:1–25 (2011)
6. Diskin, Z.: Algebraic models for bidirectional model synchronization. In: MoDELS. pp. 21–36 (2008)
7. Foster, J.N., Greenwald, M., Moore, J., Pierce, B., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3) (2007)
8. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and System Modeling 8(1), 21–43 (2009)
9. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of Model Synchronization Based on TGG. In: MODELS (2011)
10. Hofmann, M., Pierce, B.C., Wagner, D.: Symmetric lenses. In: POPL (2011)
11. Hu, Z., Mu, S.C., Takeichi, M.: A programmable editor for developing structured documents based on bidirectional transformations. Higher-Order and Symbolic Computation 21(1-2), 89–118 (2008)
12. Meertens, L.: Designing constraint maintainers for user interaction (1998), Available from http://www.kestrel.edu/home/people/meertens/
13. Song, H., Huang, G., Chauvel, F., Zhang, W., Sun, Y., Mei, H.: Instant and incremental QVT transformation for runtime models. In: MODELS (2011)
14. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Software and System Modeling 9(1), 7–20 (2010)
15. Xing, Z., Stroulia, E.: UMLDiff: an algorithm for object-oriented design differencing. In: ASE. pp. 54–65 (2005)
16. Xiong, Y., Liu, D., Hu, Z., Zhao, H., Takeichi, M., Mei, H.: Towards automatic model synchronization from model transformations. In: ASE. pp. 164–173 (2007)

17. Xiong, Y., Hu, Z., Zhao, H., Song, H., Takeichi, M., Mei, H.: Supporting automatic model inconsistency fixing. In: ESEC/SIGSOFT FSE. pp. 315–324 (2009)
18. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. Software and Systems Modeling. To appear

Appendix

Since weak invertibility and weak undoability both enforce some forms of symmetry, we may have a concern: are they indeed two different laws? Does one law imply the other? If they are indeed different, do we have lenses satisfying both laws? Specifically, we need to answer the following three questions: (1) whether there exits an undoable sd-lens that is not invertible, (2) whether there exists an invertible sd-lens that is not undoable, and (3) whether there exists a undoable and invertible sd-lens. In this section we show that the answers to the three questions are all "yes" and show three examples that fit into the three cases, respectively.

The three examples are built on the same triple space $\mathbf{A} \overset{\mathbf{R}}{\longleftrightarrow} \mathbf{B}$, which is defined as follows.

$$\begin{aligned}
&\mathbf{M_A} = \{A\}\\
&\mathbf{\Delta_A} = \{a\colon A \to A,\ a^{\smile}\colon A \to A,\ \mathsf{id}A\colon A \to A\}\\
&\mathbf{M_B} = \{B\}\\
&\mathbf{\Delta_B} = \{b\colon B \to B,\ b^{\smile}\colon B \to B,\ \mathsf{id}B\colon B \to B\}\\
&\mathbf{\Delta_{AB}} = \{r\colon A \to B\}
\end{aligned}$$

In the domain of each side, there is only one model and three arrows: the identify arrow and two mutually inverse arrows. Between $\mathbf{M_A}$ and $\mathbf{M_B}$ there is only one arrow r.

The first sd-lens $(\mathsf{fPpg}_1,\ \mathsf{bPpg}_1)$ is undoable but not invertible. Its two operations are defined below.

$$\begin{aligned}
&\mathsf{fPpg}_1(\mathsf{id}A,\ r) = (\mathsf{id}B,\ r) && \mathsf{bPpg}_1(\mathsf{id}B,\ r) = (\mathsf{id}A,\ r)\\
&\mathsf{fPpg}_1(a,\ r) = (b^{\smile},\ r) && \mathsf{bPpg}_1(b,\ r) = (a,\ r)\\
&\mathsf{fPpg}_1(a^{\smile},\ r) = (b,\ r) && \mathsf{bPpg}_1(b^{\smile},\ r) = (a^{\smile},\ r)
\end{aligned}$$

It is easy to check $(\mathsf{fPpg}_1,\ \mathsf{bPpg}_1)$ is stable and undoable. However, it is not invertible because $a.\mathsf{fPpg}_1(r).\mathsf{bPpg}_1(r) = a^{\smile}$ but a and $a^{\smile}$ are not r-equivalent as $a.\mathsf{fPpg}_1(r) = b^{\smile}$ and $a^{\smile}.\mathsf{fPpg}_1(r) = b$.

The second sd-lens $(\mathsf{fPpg}_2,\ \mathsf{bPpg}_2)$ is invertible but not undoable. Its two operations are defined below.

$$\begin{aligned}
&\mathsf{fPpg}_2(\mathsf{id}A,\ r) = (\mathsf{id}B,\ r) && \mathsf{bPpg}_2(\mathsf{id}B,\ r) = (\mathsf{id}A,\ r)\\
&\mathsf{fPpg}_2(a,\ r) = (b,\ r) && \mathsf{bPpg}_2(b,\ r) = (a,\ r)\\
&\mathsf{fPpg}_2(a^{\smile},\ r) = (\mathsf{id}B,\ r) && \mathsf{bPpg}_2(b^{\smile},\ r) = (a,\ r)
\end{aligned}$$

It is easy to check this lens is stable. By examining the definition of weak invertibility on each vertical arrow, we can see that this lens is invertible. However, it

is not undoable because $a.\mathsf{fPpg}_2(r) = b$ and $a^{\smile}.\mathsf{fPpg}_2(r) = \mathsf{id}B$ but b and $\mathsf{id}B$ are not r-equivalent as $b.\mathsf{bPpg}_2(r) = a$ and $\mathsf{id}B.\mathsf{bPpg}_2(r) = \mathsf{id}A$.

The third sd-lens $(\mathsf{fPpg}_3,\ \mathsf{bPpg}_3)$ is both undoable and invertible. Its two operations are defined below.

$$\begin{array}{ll}
\mathsf{fPpg}_3(\mathsf{id}A,\ r) = (\mathsf{id}B,\ r) & \mathsf{bPpg}_3(\mathsf{id}B,\ r) = (\mathsf{id}A,\ r) \\
\mathsf{fPpg}_3(a,\ r) = (b,\ r) & \mathsf{bPpg}_3(b,\ r) = (a,\ r) \\
\mathsf{fPpg}_3(a^{\smile},\ r) = (b^{\smile},\ r) & \mathsf{bPpg}_3(b^{\smile},\ r) = (a^{\smile},\ r)
\end{array}$$

By examining the two operations on every vertical arrow, we can see that the lens is stable, undoable and invertible.