A Generic Scheme and Properties of Bidirectional Transformations

Hugo Pacheco[1], Nuno Macedo[1], Alcino Cunha[1], and Janis Voigtländer[2]

[1] HASLab / INESC TEC & Universidade do Minho, Braga, Portugal
[2] Institute for Computer Science, University of Bonn, Germany

Abstract. The recent rise of interest in bidirectional transformations (BXs) has led to the development of many BX frameworks, originating in diverse computer science disciplines. From a user perspective, these frameworks vary significantly in both interface and predictability of the underlying bidirectionalization technique. In this paper we start by presenting a generic BX scheme that can be instantiated to different concrete interfaces, by plugging-in the desired notion of update and traceability. Based on that scheme, we then present several desirable generic properties that may characterize a BX framework, and show how they can be instantiated to concrete interfaces. This generic presentation is useful when exploring the BX design space: it might help developers when designing new frameworks and end-users when comparing existing ones. We support the latter claim, by applying it in a comparative survey of popular existing BX frameworks.

1 Introduction

Bidirectional transformations (BXs) are a "mechanism for maintaining the consistency of two (or more) related sources of information" [11]. The challenge of writing BXs has been long known in the database community since the seminal studies on view-update translation by Bancilhon and Spyratos [3] and Dayal and Bernstein [13]. More recently, the pioneering work of Foster et al. on combinatorial languages for BXs [21] has recast a lot of attention towards this challenge, and given the impulse to the birth of a new research field on BXs, uniting researchers from diverse computer science communities including programming languages, model-driven engineering and databases. In the last ten years, this burgeoning interest in BXs has led to the proposal of a vast number of approaches [11,43], inspired by different visions of the problem and motivated by different contexts where the need for bidirectionality arises.

In face of the multitude and diversity of existing approaches, each tool has been tailored to answer the challenges of its particular bidirectional scenarios, and has evolved to support different formal properties and specification styles that best suit its needs. Therefore, many of the fundamental problems of the field are not yet well established, mostly due to the non-existence of a universal classifying system and to the lack of common theoretical grounds between many of the approaches. This makes it hard to compare the various solutions, to understand precisely their advantages and limitations and provide effective criteria for assessing progress in the field.

To move forward, this pressing unification need has been gaining voice in the BX subcommunities, and such a maturation effort has been slowly undergoing through a series of seminars and workshops, displayed in publications such as [11,31,26,47]. Unfortunately, despite such significant effort, the community is still far from reaching a consensus on the terminology and properties that are desirable and satisfiable by BX tools. For example, some BX programming languages satisfy specific properties that are hard to correlate with the properties satisfied by other languages or whose practical implications on the expressiveness and behavior of the corresponding BXs are hard to understand. Moreover, although some properties are already well understood in the databases or programming languages communities, their meaning and applicability in the MDE community remains unclear. On the other side, the specification style and deployment level of most BX programming languages are still not adequate for tackling real-world scenarios currently supported by model-driven BX approaches.

In this paper, we propose a generic BX scheme in which concrete interfaces can be instantiated by choosing the desired representation of the update and traceability information. On top of that scheme, we propose a set of generic semantic properties that embody the desirable bidirectional behavior of the transformations. In addition, we walk through a number of BX frameworks and instantiate them in our scheme by incrementally exploring its design space, and the expected properties naturally emerge from such exercise. As a first attempt to validate our generic BX scheme, we present a comparative survey of up to 40 existing BX approaches emerging from diverse BX subcommunities, classified according to their interface and semantic properties. Besides providing an insightful high-level picture of the state-of-art of the field of BXs, it raises interesting questions about the key design features for classifying BXs.

Section 2 presents our generic scheme and explores the instantiations of some popular frameworks (namely mappings, lenses, maintainers, trigonal systems, edit lenses and symmetric delta-lenses). Section 3 presents the generic bidirectional properties which are then instantiated for the frameworks enumerated above, while Section 4 presents a first effort to survey existing BX techniques under the proposed axes. Sections 5 and 6 discuss related work on other classification efforts and set forth the path towards a more complete survey on the design space of BXs.

2 Scheme

We begin by clarifying what we mean by BX. The goal of a BX between A and B is to enforce consistency between values of types A and B. The term "type" should be understood as a broad placeholder for type, schema or metamodel, with "value" denoting value, instance, or model, accordingly. Consistency recovery is achieved by means of two transformations to and from whose purpose is,

Explicit	E	There is an explicitly declared consistency relation.
Transformation	T	The consistency relation is one of the transformations.
Implicit	I	The consistency relation is implicit.

Table 1: Consistency Relation.

respectively, to propagate A updates into consistent B updates and vice-versa. Some BX frameworks derive the two transformations from an explicitly declared *consistency relation* $\mathsf{R} \subseteq A \times B$ between both types. Often, conversely, the consistency relation is actually expressed in terms of the underlying transformations. In these cases, usually one of the transformations must be specified by the user, the opposite one being derived from it. In some rare cases the consistency relation is an implicit notion of the system. Table 1 summarizes these options.

This definition precludes some frameworks sometimes said to also be BX. That is the case of frameworks with general synchronization procedures that recover consistency between values that were updated concurrently. Notice that our goal is not to give a definitive definition of what is BX (thus rejecting such frameworks as not being BX), but just to clarify and limit the scope of this paper.

We depict the two application scenarios of a BX between A and B as follows:

$$\begin{tikzcd}
A \arrow[d] \arrow[dr, phantom, "\xRightarrow{\mathsf{to}}"] & B \arrow[l, dotted] \arrow[d] & A \arrow[r, dotted] \arrow[d] \arrow[dr, phantom, "\xLeftarrow{\mathsf{from}}"] & B \arrow[d] \\
A \arrow[r, dotted] & B & A & B \arrow[l, dotted]
\end{tikzcd}$$

In the left scenario, we first have $a : A$ and $b : B$ that are somehow consistent. The objective of to is to transform an update on a into an update on b such that consistency is restored. Updates are depicted using solid arrows (although they are not necessarily functions, as clarified in Section 2.1). The transformation to may also receive some extra information concerning the system state prior to the update on a. Namely, it might have access to some trace information testifying how a and b were consistent. Such *traceability* is depicted using a dotted arrow, and when not trivially derived from the updated values must also be returned by to. The right scenario is dual.

In general, transformations to and from can be typed as follows:

$$\begin{aligned}
\mathsf{to} &: \overrightarrow{\mathsf{U}}(A) \times \overleftarrow{\mathsf{T}}(A,B) \to \overleftarrow{\mathsf{U}}(B) \times \overrightarrow{\mathsf{T}}(A,B) \\
\mathsf{from} &: \overleftarrow{\mathsf{U}}(B) \times \overrightarrow{\mathsf{T}}(A,B) \to \overrightarrow{\mathsf{U}}(A) \times \overleftarrow{\mathsf{T}}(A,B)
\end{aligned}$$

Here, $\overrightarrow{\mathsf{U}}$ is a parameterized type constructor that denotes the type of A updates that to propagates, and $\overrightarrow{\mathsf{T}}$ a type constructor that denotes the type of the traceability to should produce (to be received by from). Dually, we have $\overleftarrow{\mathsf{U}}$ and $\overleftarrow{\mathsf{T}}$ for from. As we will see in the next sections, the interface of existing (and potential) BX frameworks can be obtained by giving concrete definitions for these type constructors. We should also clarify at this point that by transformation

Symmetric	S	The update propagation nature is the same in both directions. We have equal definitions for $\overrightarrow{\mathsf{U}}$ and $\overleftarrow{\mathsf{U}}$, for $\overrightarrow{\mathsf{T}}$ and $\overleftarrow{\mathsf{T}}$, and similar laws for both to and from.
Asymmetric	A	The update propagation nature is different in both directions. This may lead to different definitions for $\overrightarrow{\mathsf{U}}$ and $\overleftarrow{\mathsf{U}}$, for $\overrightarrow{\mathsf{T}}$ and $\overleftarrow{\mathsf{T}}$, and different laws for to and from.

Table 2: Symmetry.

we mean a *partial function*. Frameworks differ on the degree of totality of both transformations, which will be characterized by a specific property (Section 3).

This formal characterization encompasses both *symmetric* and *asymmetric* frameworks (see Table 2). Asymmetric frameworks are often biased towards transformation scenarios where one of the types is "larger" and contains more information than the other, whereas symmetric frameworks are more balanced and tend to consider that both types contain roughly the same information or that each may contain information not present in the other.

For each instance of the arrows in the above diagrams it may be useful to reason about its source and target values. We will use (overloaded) operators δ and ρ to denote them. Namely, for each update constructor $\overrightarrow{\mathsf{U}}$ (and dually for $\overleftarrow{\mathsf{U}}$), we have $\delta : \overrightarrow{\mathsf{U}}(A) \to A$ and $\rho : \overrightarrow{\mathsf{U}}(A) \to A$ that, given an update, denote the value in its pre- and post-state, respectively. Similarly for traceabilities, $\delta : \overrightarrow{\mathsf{T}}(A, B) \to A$ and $\rho : \overrightarrow{\mathsf{T}}(A, B) \to B$ denote the source and target values related by an instance of $\overrightarrow{\mathsf{T}}(A, B)$, respectively. $\overleftarrow{\mathsf{T}}$ traceabilities are seen in the other direction, so $\delta : \overleftarrow{\mathsf{T}}(A, B) \to B$ and $\rho : \overleftarrow{\mathsf{T}}(A, B) \to A$. We will denote updates and traceabilities by bold characters ($\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{r}, \boldsymbol{s}, ...$) in contrast to non-bold characters for the values ($a, b, ...$) returned by these operators. Although it is assumed that every arrow has a pre- and post-state value, that does not mean that they are retrieved directly from the update representation (that information may not even be present in the constructor). When a transformation $\mathsf{to}\ (\boldsymbol{a}, \boldsymbol{s}) = (\boldsymbol{b}, \boldsymbol{r})$ occurs, updates and traceabilities must agree on the respective sources and targets. This can be captured by the following properties, coined *incidence conditions* in [17].

$$\delta\boldsymbol{a} = \rho\boldsymbol{s} \qquad \delta\boldsymbol{b} = \delta\boldsymbol{s} \qquad \rho\boldsymbol{a} = \delta\boldsymbol{r} \qquad \rho\boldsymbol{b} = \rho\boldsymbol{r} \tag{1}$$

The laws for from are dual. These properties enable the retrieval of states even if not explicitly present in the constructor. For instance, an update $\boldsymbol{a}$ may not have information about its pre-state, but it could for instance be retrieved from the traceability $\boldsymbol{s}$, since $\delta\boldsymbol{a} = \rho\boldsymbol{s}$. We also assume that the input traceability truly testifies the consistency relation, i.e., $(\rho\boldsymbol{s}, \delta\boldsymbol{s}) \in \mathsf{R}$, which will be denoted by $\boldsymbol{s} \in \mathsf{R}$. Due to the incidence conditions, $\rho\boldsymbol{s}$ and $\delta\boldsymbol{s}$ can instead be accessed by $\delta\boldsymbol{a}$ and $\delta\boldsymbol{b}$, respectively, whenever they are not present in traceability.

Post-state $\overrightarrow{\mathsf{U}}(A) = A$	S	An update is represented only by the post-state.
Both states $\overrightarrow{\mathsf{U}}(A) = A \times A$	$\mathbb{S}$	The update is represented by the pre- and the post-state.
Delta $\overrightarrow{\mathsf{U}}(A) = A \times A \times \mathsf{D}(A, A)$	D	Beside the pre- and the post-state, an update representation also comes with a *sameness relation* stating which components of both are conceptually the same.
Edit $\overrightarrow{\mathsf{U}}(A) = \mathsf{O}(A)^\star$	E	An update is represented by the sequence of edit operations that was performed.
State + Edit $\overrightarrow{\mathsf{U}}(A) = A \times \mathsf{O}(A)^\star$	$\mathbb{E}$	An update is represented by the pre-state and the sequence of edit operations that was performed.
Function $\overrightarrow{\mathsf{U}}(A) = A \to A$	F	An update is represented by a semantic value that models it as an endo-function.

Table 3: Update.

2.1 Update representation

One of the main axes distinguishing existing frameworks is *update representation*, i.e., what are the concrete definitions of $\overrightarrow{\mathsf{U}}$ and $\overleftarrow{\mathsf{U}}$ in the above generic scheme. Some possible definitions are presented in Table 3.

Frameworks that fall within the first two categories are usually known as *state-based*, since only the value in the post- (and sometimes also the pre-) state of an update is considered. When some knowledge of the exact changes that were (or have to be) performed in an update is represented, we have an *operation-based* framework. A possible way to represent such changes is via a *sameness relation* tracing back components of the post-state to corresponding components in the pre-state. In Table 3 we encapsulate the concrete definition of such a sameness relation in the parameterized type constructor D, since it depends on the domains involved in the transformation[3]. Notice that a sameness relation carries more information than the conjunction of pre- and post-state alone: for example, if a component is deleted and a new one inserted with the same content, these components would be unconnected in the sameness relation, but indistinguishable otherwise. A sequence of edit operations is another possible representation for updates. Again, since the set of edit operations supported by each type varies, we abstract away its concrete definition in the parameterized type constructor O. Another alternative is to store the pre-state along with the edit-sequence, in which case the post-state could also be calculated by applying the edit-sequence to the pre-state. When updates are performed programmatically, one might only have access to the executable that performed them instead of a syntactic representation. This information might still be exploited by frameworks implemented on top of semantic BX techniques.

Although the extra knowledge in operation-based frameworks can lead to BXs that satisfy more precise properties (see discussions in [17,30]), they usually de-

[3] $\mathsf{D}(A, B)$ should not be confused with the set of all binary relations between A and B, which is denoted by $\mathcal{P}(A \times B)$.

mand a tight coupling with applications, so that they can track the changes that characterize an update. Transforming updates onto updates also makes BXs more natural with incrementality [24] (rather than recomputing a new model when the correlated model changes, only a small "delta" is propagated). State-based frameworks, on the other hand, are more flexible and support more usage scenarios, like integration with off-the-shelf applications that have not been designed with bidirectionality in mind, and are moreover less sensitive to "noise" in the updates. The distinction between state- and operation-based approaches is not always obvious. Hybrid approaches may build a state-based system with a richer operation-based core. As they discard all update information, some model differencing procedure is required to infer new hypothetical update operations. Similarly, an incremental system can have a simple state-based core, but keep track of operations merely as an optimization, to exploit the locality of updates.

2.2 Traceability representation

Likewise to update representation, Table 4 presents possible definitions for *traceability representation*, i.e., the concrete definitions of $\overrightarrow{\mathsf{T}}$ and $\overleftarrow{\mathsf{T}}$ in the above generic scheme. In the first category we have frameworks without any trace information. This is a very limiting scenario, since only the to be translated update itself is known when attempting to recover consistency. Not even information about the current state of the opposite domain is known: in a state-based framework this means that only "fresh" values must be produced, ruling out any sort of incremental updating. In the second category traceability amounts precisely to the value of the opposite domain. This is the case, for example, of frameworks that tackle the view-update problem, and that require (specifically, from requires) access to the (previous) value of the source (A) to fetch information not recoverable from the view (B). Traceability can also be represented by means of a *complement*: $\mathsf{C}(A, B)$ is a parameterized type constructor that encapsulates the complement of a type A with respect to another type B. Essentially, elements of C will contain some of the components of one (or both) value(s) not present in the other. Finally, we can also use a sameness relation to trace the execution of a previous update translation, pinpointing the exact pairs of components in the source and target that testify the consistency between them.

Sometimes the returned traceability is redundant and can be computed from the remaining available information. A possible instantiation where this occurs is when $\overrightarrow{\mathsf{U}} = \mathrm{S}$ and $\overrightarrow{\mathsf{T}} = \mathrm{S}$. Since $\mathsf{to}\ (\boldsymbol{a}, \boldsymbol{s}) = (\boldsymbol{b}, \boldsymbol{r})$, the $\boldsymbol{r} : \overrightarrow{\mathsf{T}}(A, B)$ will actually be equivalent to $\delta\boldsymbol{r} : a$, which from (1) is equivalent to $\rho\boldsymbol{a} : A$, the post-state of the source update that is precisely one of the inputs of to. In such cases, the transformations will be typed just as follows:

$$\mathsf{to} : \overrightarrow{\mathsf{U}}(A) \times \overleftarrow{\mathsf{T}}(A, B) \to \overleftarrow{\mathsf{U}}(B) \qquad \mathsf{from} : \overleftarrow{\mathsf{U}}(B) \times \overrightarrow{\mathsf{T}}(A, B) \to \overrightarrow{\mathsf{U}}(A) \tag{2}$$

None $\overrightarrow{\mathsf{T}}(A,B) = 1$	N	No trace information is represented.
State $\overrightarrow{\mathsf{T}}(A,B) = A$	S	Only the source state is represented.
Complement $\overrightarrow{\mathsf{T}}(A,B) = \mathsf{C}(A,B)$	C	Some complement of the source and/or target values is represented.
Delta $\overrightarrow{\mathsf{T}}(A,B) = A \times B \times \mathsf{D}(A,B)$	D	Beside both consistent states, a sameness relation between them is represented.

Table 4: Traceability.

2.3 Exploring the design space

By instantiating these two axes we get different flavors of BX frameworks. As we will present next, some instantiations correspond to well-known existing frameworks, but others have not been instantiated yet, and an interesting question is whether they might be useful or not.

Not surprisingly, one of the most popular schemes is that of bidirectional *mappings*, where no traceability information is represented and updates are typically represented by only the post-state. Formally, we have $\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathrm{S}$ and $\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{N}$, leading to the scheme $\mathsf{to} : A \to B$ and $\mathsf{from} : B \to A$. In these frameworks, the consistency relation typically corresponds to one of the transformations and thus is omitted. A symmetric instantiation of this category is that of bijective languages, whose transformations establish a bijection between subsets of A and B. These subsets contain essentially the same information but just present it differently. Such languages promote the interoperability between different formats and are easy to reason about because bijectivity is preserved by composition and inversion. Less restrictive asymmetric mapping frameworks encompass transformations that are only reversible in a particular direction, for example when B refines A.

The most popular asymmetric scheme, *lenses* [21], was proposed as a solution to the classical view-update problem from database theory [3]. Updates are still represented using just the post-state, but from requires traceability information to deal with missing information, namely the original source value of type A. More precisely, we have $\overrightarrow{\mathsf{T}} = \mathrm{S}$ and $\overleftarrow{\mathsf{T}} = \mathrm{N}$. Since the returned $\overleftarrow{\mathsf{T}}(A,B)$ is equal to the input of to, we end up with the simplified scheme (2), resulting in the interface $\mathsf{to} : A \to B$ and $\mathsf{from} : B \times A \to A$ (known, respectively, as *get* and *put* in the lens framework). The consistency relation is assumed to be $(a,b) \in \mathsf{R}$ iff $\mathsf{to}\ a = b$. In order to guarantee stronger properties or support particular transformation scenarios, some lens-like approaches are operation-based , or use an additional sameness relation, providing a traceability between views and sources.

The asymmetric treatment of lenses only works well for (essentially) surjective (information decreasing) transformations, since to cannot access B details without counterpart in A. For more general transformations without a dominant flow of information, where each of the source and target models may contain information not present in the other, we end up with symmetric schemes. Among those, *maintainers* [37] are among the most popular, where $\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathrm{S}$ and $\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{S}$, leading to the scheme $\mathsf{to} : A \times B \rightarrow B$ and $\mathsf{from} : B \times A \rightarrow A$, where updates are propagated given knowledge of the pre-state of the respective opposite transformations. Likewise to lenses, since the output traceability can trivially be derived from input updates, it is not returned by the transformations. Also, unlike the previous frameworks, there is now an explicitly declared consistency relation R, from which to and from are somehow inferred.

Trigonal systems were proposed in [14] to avoid the recalculation of the whole state values when updates are incremental. As a generalization of maintainers, besides having knowledge of the target pre-state, information about the source pre-state is also present. Concretely, we now have $\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathbb{S}$ and $\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{S}$. The existence of an explicit consistency relation R is also assumed. Like the previous schemes, the content captured by $\overrightarrow{\mathsf{T}}$ is trivially derived from the input information, so it is not returned by to. However, since the value of the pre-state of $\overleftarrow{\mathsf{U}}$ can also be directly retrieved from $\overleftarrow{\mathsf{T}}$, it is also omitted from the output. Putting it all together, we have the scheme $\mathsf{to} : (A \times A) \times B \rightarrow B$ and $\mathsf{from} : (B \times B) \times A \rightarrow A$.

Symmetric lenses [28] assume $\overleftarrow{\mathsf{U}} = \overleftarrow{\mathsf{T}} = \mathrm{S}$ and represent the traceability as a complement ($\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{C}$). Thus, we have $\mathsf{to} : A \times \mathsf{C}(A, B) \rightarrow B \times \mathsf{C}(A, B)$ (and vice-versa), and the complement $\mathsf{C}(\mathrm{S}, T)$ stores both the information of A not present in B (passed as input to from) and vice-versa. *Edit lenses* [29] are an operation-based formulation of symmetric lenses, with edit-sequences as updates ($\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathrm{E}$). We have $\mathsf{to} : \mathsf{O}(A)^\star \times \mathsf{C}(A, B) \rightarrow \mathsf{O}(B)^\star \times \mathsf{C}(A, B)$ and the opposite for from. This time, the transformations do not process states and the complement $\mathsf{C}(A, B)$ stores only some extra information about A and B that is used to disambiguate updates, but not sufficient to restore the original states.

Among symmetric schemes, *symmetric delta-lenses* [16] are one of the most general. Here $\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathrm{D}$ and $\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{D}$: besides the pre-state values of A and B, we have a sameness relation between them in the traceability, and likewise for the update itself. We have $\mathsf{to} : (A \times A \times \mathsf{D}(A, A)) \times (A \times B \times \mathsf{D}(A, B)) \rightarrow (B \times B \times \mathsf{D}(B, B)) \times (A \times B \times \mathsf{D}(A, B))$ and the opposite for from. An explicit consistency relation R between updates is also present.

3 Properties

The interface of a framework gives some hints about its expressivity but says little about the actual behavior of the transformations. Such behavior is usually specified by high-level algebraic properties that enforce some predictability on the system (namely concerning bidirectionality). Table 5 identifies several generic properties that, independently of the framework, might be desirable from an end-user perspective. We only present the properties from the perspective of from (i.e., propagating updates from the B side to the A side). The dual properties

$$\frac{}{\mathsf{from}\ (\mathsf{id}_B, \boldsymbol{r}) \sqsubseteq (\mathsf{id}_A, \boldsymbol{r}^\circ)} \qquad \text{(from-Stability)}$$

$$\frac{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) = (\boldsymbol{a}, \boldsymbol{s})}{\mathsf{to}\ (\boldsymbol{a}, \boldsymbol{r}^\circ) \sqsubseteq (\boldsymbol{b}, \boldsymbol{s}^\circ)} \qquad \text{(from-Invertibility)}$$

$$\frac{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) = (\boldsymbol{a}, \boldsymbol{s})}{\mathsf{from}\ (\boldsymbol{b}^\circ, \boldsymbol{s}^\circ) \sqsubseteq (\boldsymbol{a}^\circ, \boldsymbol{r}^\circ)} \qquad \text{(from-Undoability)}$$

$$\frac{\mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{r}) = (\boldsymbol{a}_1, \boldsymbol{s}_1) \quad \mathsf{from}\ (\boldsymbol{b}_2, {\boldsymbol{s}_1}^\circ) = (\boldsymbol{a}_2, \boldsymbol{s}_2)}{\mathsf{from}\ (\boldsymbol{b}_2 \circ \boldsymbol{b}_1, \boldsymbol{r}) \sqsubseteq (\boldsymbol{a}_2 \circ \boldsymbol{a}_1, \boldsymbol{s}_2)} \qquad \text{(from-History-ignorance)}$$

$$\frac{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) = (\boldsymbol{a}, \boldsymbol{s})}{\boldsymbol{s} \in \mathsf{R}} \qquad \text{(from-Correctness)}$$

$$\frac{\boldsymbol{b} \circ \boldsymbol{r} \in \mathsf{R}}{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) \sqsubseteq (\mathsf{id}_A, (\boldsymbol{b} \circ \boldsymbol{r})^\circ)} \qquad \text{(from-Hippocraticness)}$$

$$\frac{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) = (\boldsymbol{a}, \boldsymbol{s}) \quad \boldsymbol{s}^\circ \circ \boldsymbol{a} = {\boldsymbol{s}_1}^\circ \circ \boldsymbol{a}_1 \quad \boldsymbol{s}_1 \in \mathsf{R}}{\boldsymbol{a} \leqslant \boldsymbol{a}_1} \qquad \text{(from-Least-update)}$$

$$\frac{}{\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) \downarrow} \qquad \text{(from-Totality)}$$

Table 5: Properties.

can also be specified for to. All free variables are implicitly universally quantified. This formalization is to some extent a textual version of the graphical *tile algebra* [15], previously used to formalize some of the laws presented here. Since the transformation can be partial, the properties are only required to hold when they yield a result. Given a transformation f, $f\ x \downarrow$ holds when f is defined on x, and $f\ x \sqsubseteq y$ holds if $f\ x \downarrow \Rightarrow f\ x = y$.

Stability imposes that null updates must be translated to null updates, in the sense that if a B is not modified, then no change shall be performed on the consistent A. We represent a null update on A by a constant id_A, where $\delta\mathsf{id}_A = \rho\mathsf{id}_A$. *Invertibility* states that it shall be possible to revert the application of a transformation by applying the opposite transformation. This law implies that updates on the B side are translated faithfully to A, otherwise they could not be inverted. In asymmetric frameworks $\overrightarrow{\mathsf{T}}(A, B)$ might be different from $\overleftarrow{\mathsf{T}}(A, B)$. In those cases, we assume that traceability $\boldsymbol{r} : \overrightarrow{\mathsf{T}}(A, B)$ can be reversed as $\boldsymbol{r}^\circ : \overleftarrow{\mathsf{T}}(A, B)$, where $\delta\boldsymbol{s}^\circ = \rho\boldsymbol{s}$ and $\rho\boldsymbol{s}^\circ = \delta\boldsymbol{s}$. *Undoability* ensures that an update translation can be undone by re-applying the same transformation with an inverse update. Likewise to traceability, given an update $\boldsymbol{a} : \overrightarrow{\mathsf{U}}(A)$ its inverse will be denoted by $\boldsymbol{a}^\circ : \overleftarrow{\mathsf{U}}(A)$, where $\delta\boldsymbol{a}^\circ = \rho\boldsymbol{a}$ and $\rho\boldsymbol{a}^\circ = \delta\boldsymbol{a}$. Also, two updates $\boldsymbol{a}_1 : \overrightarrow{\mathsf{U}}(A)$ and $\boldsymbol{a}_2 : \overrightarrow{\mathsf{U}}(A)$ can be sequentially composed as $\boldsymbol{a}_2 \circ \boldsymbol{a}_1 : \overrightarrow{\mathsf{U}}(A)$. *History-ignorance* states that update translation does not depend on the past history. In practice, this means that two consecutive update translations can be performed at once on the composed update.

The following three properties involve the consistency relation. *Correctness* simply states that a transformation restores consistency. *Hippocraticness* is a stronger version of stability (although not always desirable [14]), stating that an update that does not break the consistency should be ignored. In the generic formulation we assume that a traceability $\boldsymbol{r} : \overrightarrow{\mathsf{T}}(A, B)$ can be composed with an update $\boldsymbol{b} : \overrightarrow{\mathsf{U}}(B)$ to yield a traceability $\boldsymbol{b} \circ \boldsymbol{r} : \overrightarrow{\mathsf{T}}(A, B)$ relating the original source A with the updated B. Due to the incidence conditions, checking the consistency of $\boldsymbol{b} \circ \boldsymbol{r}$ is equivalent to checking the consistency of these values. In schemes where the consistency relation is one of the transformations, correctness and hippocraticness degenerate into invertibility and stability, respectively. The *least-update* property can be seen as an additional *quality property* entailing that the returned update must be the smallest among all R-consistent ones that could have been returned. To compare updates, we assume the existence of a total preorder $\leqslant$ on $\overrightarrow{\mathsf{U}}(A)$. When $\mathsf{from}\ (\boldsymbol{b}, \boldsymbol{r}) = (\boldsymbol{a}, \boldsymbol{s})$ then $\boldsymbol{a}$ must be smaller than every update $\boldsymbol{a}_1$ that could lead to a value consistent with the post-state of $\boldsymbol{b}$ (testified by a consistent traceability $\boldsymbol{s}_1$). Assuming that, for an already consistent state, the null target update is the unique minimal update, least-update subsumes hippocraticness.

So far, we presented the properties modulo undefinedness of the unidirectional transformations. This is because totality requirements are by themselves an important feature in the design of a BX framework. In practice, it is often convenient to acknowledge that the type system might not be expressive enough to capture all constraints induced by the transformations, and to allow the source and target types to be larger than the actual domains of the transformations, leading to partially defined transformations. While this might be a “show stopper” for batch applications that are expected to always produce results, it is usually acceptable for interactive applications: an editor does not need to handle every update and can signal an error to the user disallowing a specific modification. Partiality is not adequate for security applications [22] though, since users might extract information about the hidden data from the cases for which the transformations fail. As such, to allow a finer-grain comparison of frameworks, we choose to factor out *totality* as an orthogonal property: it holds for a transformation if it is defined for every possible combination of update and traceability.

Sometimes, weaker versions of the above laws may be satisfied instead. For example, we can have weaker versions of *invertibility* where the final state is equal to the original one modulo another update translation. This particular *weak invertibility* is a kind of *convergence* law (or *bi-idempotence* in [30]), since it entails that update translation eventually converges into stable states. Other *weak* variants of the laws occur when value comparison ignores details that are inessential for an application scenario, like ordering, whitespaces or structure sharing. For operation-based frameworks, these may also mean that round-tripping does not preserve the full update, but only its post-state. An interesting weaker version of totality is *safety* [40] (also known as *domain correctness* in [14]), which entails that a transformation is defined at least for the range of the opposite one, independently of its pre-state. Taking into account the consistency relation, safety can also be stated as follows: a transformation must be defined for every source value that has at least one consistent target. Weaker versions of *correctness* include allowing the creation of inconsistent states if no consistent ones exists.

3.1 Revisiting the design space

To instantiate the generic properties for a concrete framework, one must first devise how to express null, inversion and composition of updates and traceability. In some frameworks some of these might not be expressible, meaning that some properties may not be applicable. For example, *mappings* have no traceability and updates are represented just by the post-state, hence there is no way to reason about pre-states. This implies that neither null updates, nor update inversion can be defined, and from-Stability, from-Undoability, and from-Hippocractiness are not expressible. from-Invertibility is just $\mathsf{to}\ (\mathsf{from}\ b) \sqsubseteq b$, ensuring from to be left-invertible (injective). Without any knowledge of the pre-state, from-History-ignorance holds trivially. Assuming the consistency relation $(a, b) \in \mathsf{R} \equiv (b = \mathsf{to}\ a)$, from-Correctness degenerates to from-Invertibility, and from-Least-update amounts to checking that if $\mathsf{from}\ b = a$, then a is smaller than every a_1 leading to the same b, that is $\mathsf{from}\ b = a \wedge \mathsf{to}\ a = \mathsf{to}\ a_1 = b \Rightarrow a \leqslant a_1$.

Unlike mappings, in *lenses* there is some traceability information when applying from that allows us to reason about pre-states. In particular, when from is applied to traceability a (the original source value), due to the consistency relation $(a, b) \in \mathsf{R} \equiv (b = \mathsf{to}\ a)$, we know that the pre-state of the input update is $\mathsf{to}\ a$. Hence, a null update has the same post-state, and from-Stability can be instantiated as $\mathsf{from}\ (\mathsf{to}\ a, a) \sqsubseteq a$ (known in this framework as GETPUT). Instantiation of from-Invertibility is more straightforward — just ignore unused traceability — yielding $\mathsf{from}\ (b_1, \_) = a_1 \Rightarrow \mathsf{to}\ a_1 \sqsubseteq b_1$ (known as PUTGET). To instantiate from-Undoability, we follow the same approach as in from-Stability, resulting in $\mathsf{from}\ (b_1, a) = a_1 \Rightarrow \mathsf{from}\ (\mathsf{to}\ a, a_1) \sqsubseteq a$. Since the composition of two state-based updates $b \circ a$ is just b, instantiation of from-History-Ignorance is $\mathsf{from}\ (b_1, a) = a_1 \wedge \mathsf{from}\ (b_2, a_1) = a_2 \Rightarrow \mathsf{from}\ (b_2, a) \sqsubseteq a_2$ (known as PUTPUT). Due to the consistency relation being to, from-Correctness and from-Hippocraticness degenerate into from-Invertibility and from-Stability, respectively. Likewise to mappings, from-Least-update is formulated as $\mathsf{from}\ (b_1, a) = a_1 \wedge \mathsf{to}\ a_2 = b_1 \Rightarrow a_1 \leqslant a_2$.

Like in the previous frameworks, the value of the pre-state of the input update is not explicitly represented in *maintainers*. However, unlike lenses, the declared consistency relation of a maintainer may not be functional (deterministic), and a value of A is not uniquely related to another value of B. This means that it is impossible to identify a null update given only the original source value present in the traceability, and from-Stability cannot be formulated. In the case of from-Invertibility, to revert a transformation $\mathsf{from}\ (b_1, a) = a_1$ we need to invert the traceability a, and recover the original consistent b. As discussed above, in general this is not possible, but we can generalize this property assuming that the transformation can be reverted for any consistent b, that is $(a, b) \in \mathsf{R} \wedge \mathsf{from}\ (b_1, a) = a_1 \Rightarrow \mathsf{to}\ (a_1, b) \sqsubseteq b_1$. Following a similar approach, from-Undoability can be formulated as $(a, b) \in \mathsf{R} \wedge \mathsf{from}\ (b_1, a) \sqsubseteq$

$a_1 \Rightarrow \mathsf{from}\ (b, a_1) \sqsubseteq a$. Since the from interface is similar to lenses, from-History-ignorance is exactly the same. Due to the explicit consistency relation, from-Correctness is directly formulated as $\mathsf{from}\ (b, \_) = a \Rightarrow (a, b) \in \mathsf{R}$ and from-Hippocraticness as $(a, b) \in \mathsf{R} \Rightarrow \mathsf{from}\ (b, a) \sqsubseteq a$. Lastly, from-Least-update is again similar to that of lenses but with an explicit consistency relation, i.e., $\mathsf{from}\ (b, a) = a_1 \wedge (a_2, b) \in \mathsf{R} \Rightarrow a_1 \leqslant a_2$.

In *trigonal systems*, updates are represented by both pre- and post-state value. As such, null updates and composition and inversion of updates can be directly defined as $\mathsf{id}_B = (b, b)$, $(b_1, b_2) \circ (b_2, b_3) = (b_1, b_3)$ and $(b_1, b_2)^\circ = (b_2, b_1)$, and the instantiation of properties becomes rather straightforward. Namely, from-stability is $\mathsf{from}\ ((b, b), a) \sqsubseteq a$, from-Invertibility is $\mathsf{from}\ ((b, b_1), a) = a_1 \Rightarrow \mathsf{to}\ ((a, a_1), b) \sqsubseteq b_1$, from-Undoability is $\mathsf{from}\ ((b, b_1), a) = a_1 \Rightarrow \mathsf{from}\ ((b_1, b), a_1) \sqsubseteq a$, and from-History-ignorance is $\mathsf{from}\ ((b, b_1), a) = a_1 \wedge \mathsf{from}\ ((b_1, b_2), a_1) = a_2 \Rightarrow \mathsf{from}\ ((b, b_2), a) \sqsubseteq a_2$. Likewise to maintainers, due to the explicit consistency relation, the remaining instantiations are also immediate: from-Correctness is $\mathsf{from}\ ((\_, b_1), \_) = a_1 \Rightarrow (a_1, b_1) \in \mathsf{R}$, from-Hippocracticness is $(a, b) \in \mathsf{R} \Rightarrow \mathsf{from}\ ((\_, b_1), a) \sqsubseteq a$ and from-Least-update is $\mathsf{from}\ ((\_, b_1), a) = a_1 \wedge (a_2, b_1) \in \mathsf{R} \Rightarrow (a, a_1) \leqslant (a, a_2)$. Likewise, the instantiation of the properties in the framework of *symmetric delta-lenses* is straightforward, since both the pre- and post-state values of updates are represented and have an explicit consistency relation.

In *edit lenses*, updates are represented by sequences of edit operations: the null update is the empty sequence $[\,]$, composing updates $\boldsymbol{a}$ and $\boldsymbol{b}$ amounts to concatenation $\boldsymbol{a} +\!\!+ \boldsymbol{b}$, and assuming each edit operation to be undoable, an update $\boldsymbol{a} = [a_1, .., a_n]$ could be inverted, for example, as $\boldsymbol{a}^\circ = [{a_n}^\circ, .., {a_1}^\circ]$. Traceability information (the complement) is "symmetric" in the sense that $\boldsymbol{c}^\circ = \boldsymbol{c}$. Equipped with these definitions, some properties can be directly instantiated: from-Stability is $\mathsf{from}\ ([\,], \boldsymbol{c}) \sqsubseteq ([\,], \boldsymbol{c})$, from-Invertibility is $\mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) = (\boldsymbol{a}_1, \boldsymbol{c}_1) \Rightarrow \mathsf{to}\ (\boldsymbol{a}_1, \boldsymbol{c}) \sqsubseteq (\boldsymbol{b}_1, \boldsymbol{c}_1)$, from-Undoability is $\mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) = (\boldsymbol{a}_1, \boldsymbol{c}_1) \Rightarrow \mathsf{from}\ ({\boldsymbol{b}_1}^\circ, \boldsymbol{c}_1) \sqsubseteq ({\boldsymbol{a}_1}^\circ, \boldsymbol{c})$, and from-History-ignorance is $\mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) = (\boldsymbol{a}_1, \boldsymbol{c}_1) \wedge \mathsf{from}\ (\boldsymbol{b}_2, \boldsymbol{c}_1) = (\boldsymbol{a}_2, \boldsymbol{c}_2) \Rightarrow \mathsf{from}\ (\boldsymbol{b}_1 +\!\!+ \boldsymbol{b}_2, \boldsymbol{c}) \sqsubseteq (\boldsymbol{a}_1 +\!\!+ \boldsymbol{a}_2, \boldsymbol{c}_2)$. In from-Correctness the existence of the initial source-target pair (a, b) that gave origin to complement $\boldsymbol{c}$ is assumed, over which the resulting edit-sequences are applied, resulting in $(a, b) \in \mathsf{R} \wedge \mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) = (\boldsymbol{a}_1, \boldsymbol{c}_1) \Rightarrow (\boldsymbol{a}_1\ a, \boldsymbol{b}_1\ b) \in \mathsf{R}$, where $(\boldsymbol{a}\ a)$ denotes the application of the edit-sequence $\boldsymbol{a}$ to the value a. The information about the system state is external to the transformations, as updates are only represented by edit-sequences. from-Hippocracticness applies to transformations over the complement of states that were already consistent, represented as $(a, b) \in \mathsf{R} \Rightarrow \mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) \sqsubseteq ([\,], \boldsymbol{c})$. Lastly, from-Least-update is formulated as $(a, b) \in \mathsf{R} \wedge \mathsf{from}\ (\boldsymbol{b}_1, \boldsymbol{c}) = (\boldsymbol{a}_1, \boldsymbol{c}_1) \wedge (\boldsymbol{a}_2\ a, \boldsymbol{b}_1\ b) \in \mathsf{R} \Rightarrow \boldsymbol{a}_1 \leqslant \boldsymbol{a}_2$. Note that the pre-order compares only edit-sequences.

Each framework has a notion of what is a *well-behaved* transformation, i.e., the minimum properties it must satisfy to be considered reasonable. Typically, a transformation is *well-behaved* if it is at least stable and correct, i.e., preserves null updates and recovers consistency. In mappings to-Correctness degenerates into to-Invertibility, meaning that to is injective and B is a *refinement* of A

(it contains more information). In a *well-behaved* symmetric mapping, both to-Invertibility and from-Invertibility hold and the BX is an *isomorphism*, such that to and from are bijections (when restricted to the respective domains). In asymmetric lenses, from-Invertibility together with from-Stability are the typical laws required for a lens to be well-behaved. The law from-Invertibility implies that to is surjective (again when restricted to the respective domain) and B is an *abstraction* of A (also called a view), meaning that the target contains less information than the source. If a framework also satisfies history ignorance, then it is usually considered *very well-behaved*. Since stability and history ignorance entail undoability, very well-behaved frameworks are also undoable.

Some approaches do not enforce any totality requirements . However, this can easily be abused: a (partial) BX can be trivially well-behaved if both transformations are always undefined. For asymmetric frameworks, one transformation generally dominates the data flow and has stronger totality requirements than the other, and thus approaches usually assume to to be total and from either partial or safe. For symmetric frameworks, there is not generally a dominant data flow (for example, not every Java feature can be represented with a relational database schema, and vice-versa), and both transformations may be plausibly partial. For *total* BXs, the types capture the exact domains over which the transformation is defined and guaranteed to behave well, ensuring that update translation cannot fail at run time.

4 A survey of existing BX frameworks

Based on the proposed generic scheme and properties, we attempted a comprehensive survey of existing BX tools and frameworks. Table 6 presents the results of this first effort[4]. Regarding the scheme, the different axes were classified according to Tables 1, 2, 3, and 4. Regarding the semantic properties, for every property of Table 5, we use right arrows to denote that the property holds for to and a left arrow for from. A normal arrow denotes that a property is satisfied by all well-behaved BXs in a given approach, while a dashed arrow signals that well-behaved BXs only satisfy a weaker version. The absence of an arrow means that well-behaved BXs do not satisfy a particular property, or that such property is not explicitly stated by the authors or implied by other properties. This also means that laws implied by others are not depicted. For instance, as we have seen, correctness and hippocraticness in lenses degenerates into invertibility and stability, due to the consistency relation $b \mathsf{R} a \equiv b = \mathsf{to}\ a$. For totality, the absence of an arrow means that the transformation is partial, and a dashed arrows means that the transformation is safe.

[4] A more detailed classification of BX approaches related to this paper can be found in [40, Chapter 3]. Although the mentioned scheme regards only 3 specific frameworks (mappings, lenses and maintainers), this complementary work also proposes a taxonomy for the particular deployment features of BX frameworks and a textual justification for each of the entries in Table 6.

Some particular entries in Table 6 do not represent concrete BX tools but proposals of BX frameworks. Such entries are signaled with an asterisk $*$: marked properties indicate the criteria for well-behaved BXs in such frameworks. The duplicate entry of [37] is due to the fact that two different approaches are presented in it.

Table 6 is not intended to be complete, but rather to provide a high-level picture of existing BX tools. It must be read with some caution, though, since it does not capture specific intricacies of particular frameworks that are not representable in our generic scheme, and since some of them are omissive or ambiguous regarding particular features, which leads to some subjectivity in the classification. To alleviate this we intend to publish the current survey online, and engage the authors in the classification of their own tools and frameworks. With completeness in mind, we also appreciate any suggestion of additional classification axes and BX frameworks to include in the survey, in order to reach a detailed global picture of the state of the art of the field.

5 Related work

Acknowledging the heterogeneity of the field, [11] surveys the related literature on BXs, grouping existing work by subcommunities, identifying some of the grand challenges of the field and providing a modest discussion on the terminology, key concepts and semantic properties used across the represented communities. A more focused picture on bidirectional model transformations is given in [45], with emphasis on tool support and inherent open challenges. Acknowledging the growing effort in the BX community towards unification, [47] compares five particular BX tools in terms of their strengths and weaknesses for particular scenarios. For such comparison, it proposes a simple taxonomy to analyze the behavior of BX tools from a model-driven perspective, but that does not consider particular BX properties.

A detailed feature model for the classification of model transformation approaches is proposed in [12], together with a survey of a vast number of existing approaches. Nevertheless, this classification is focused on design features rather than semantic properties, and does not pay particular detail to bidirectionality. A detailed scheme for classifying a wide spectrum of bidirectional model synchronization axiomatizations and features is provided in [1], illustrating particular instantiations with examples of existing systems. Still, only design properties are considered, and not the semantic properties of each instance. In contrast, [14] proposes a classifying system in which some state-based BX frameworks can be compared and analyzed in terms of their semantic laws. Despite most laws proposed there for lenses, maintainers and trigonal systems matching our own instantiations (see Section 3), invertibility and undoability do not. In fact, the laws for maintainers assume some surjectivity constraints on the transformations, while those for trigonal systems do not. This is an indication that no consistent method for the instantiation of the laws was followed. A general framework for building delta-based model synchronization frameworks is proposed in [15], which takes the shape $\overrightarrow{\mathsf{U}} = \overleftarrow{\mathsf{U}} = \mathrm{D}$ and $\overrightarrow{\mathsf{T}} = \overleftarrow{\mathsf{T}} = \mathrm{D}$ and considers generic laws similar to those from Table 5, except for invertibility and least-update. However, it is not explored how existing frameworks can be instantiated under this general framework, and how the choices made on the scheme affect the properties. We follow a bottom-up characterization of the laws from state-based mappings to symmetric delta lenses, and also provide an extensive comparison of existing BX tools.

6 Conclusions and future work

In this paper, we have presented a generic scheme and several generic properties of BX frameworks. The generic scheme can be instantiated along two main axes (update and traceability representation), and the presented properties cover the most for bidirectional laws proposed in the literature. We have also shown how this generic presentation can be instantiated to obtain and compare most of the existing concrete BX frameworks, such as lenses or maintainers. We have applied this comparative study not only to such broad framework categories, but to a large set of concrete BX tools and techniques proposed in the literature. In the future we intend to extend this survey effort with more entries and classification axes (covering, for example, deployment features, such as data domain or bidirectionalization technique) to achieve a detailed global picture of the state of the art of the field.

One of the key roles of properties is to ensure some degree of predictability concerning the behavior of BXs. As seen in our survey, most frameworks only guarantee stability, correctness and invertibility, but unfortunately these properties still leave a lot of room for unpredictable and sometimes unreasonable behavior, defeating their goal as a means for comparing the effectiveness of two BX frameworks. In the long term, this problem should be solved with the study of new properties that better characterize more refined behavior, like minimization of update translation. Meanwhile, in the continuation of [11], we intend to address the problem by developing a suite of paradigmatic examples for each application domain, so that the user can compare the behavior of different frameworks within that domain. Moreover, citing [47]:

> A more ambitious goal would be a truly unified theoretical foundation to BX. [...] Such a unification would not be trivial to accomplish, since it will require a huge collaborative effort, involving researchers from distinct communities, countries and scientific cultures. However, such a unification would be desirable, both for better addressing existing bidirectional scenarios and for tackling largely unexplored, yet important scenarios.

We believe our work presents a solid base towards such unified theoretical foundation. In its current form, we think it might help developers when designing new frameworks and end-users when comparing existing ones. Finally, we intend to improve the accuracy and completeness of our survey by enrolling the BX community: the taxonomy and current results will be published online, and authors will be invited to discuss and extend it by classifying their own frameworks.

Acknowledgements

This work is funded by ERDF - European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by national funds through the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-020532. The first author of the paper is also sponsored by FCT grant SFRH/BD/69585/2010.

References

1. Antkiewicz, M., Czarnecki, K.: Design space of heterogeneous synchronization. In: GTTSE 2007, LNCS, vol. 5235, pp. 3–46. Springer (2008)
2. Atanassow, F., Jeuring, J.: Customizing an XML-Haskell data binding with type isomorphism inference in Generic Haskell. Sci. Comput. Program. 65, 72–107 (2007)
3. Bancilhon, F., Spyratos, N.: Update semantics of relational views. ACM T. Database Syst. 6(4), 557–575 (1981)
4. Barbosa, D., Cretin, J., Foster, J., Greenberg, M., Pierce, B.: Matching lenses: alignment and view update. In: ICFP. pp. 193–204. ACM (2010)
5. Berdaguer, P., Cunha, A., Pacheco, H., Visser, J.: Coupled schema transformation and data: Conversion for XML and SQL. In: PADL, LNCS, vol. 4354, pp. 290–304. Springer (2007)
6. Bohannon, A., Foster, J., Pierce, B., Pilkiewicz, A., Schmitt, A.: Boomerang: resourceful lenses for string data. In: POPL. pp. 407–419. ACM (2008)
7. Bohannon, A., Pierce, B., Vaughan, J.: Relational lenses: a language for updatable views. In: PODS. pp. 338–347. ACM (2006)
8. Brabrand, C., Møller, A., Schwartzbach, M.: Dual syntax for XML languages. Information Systems 33, 385–406 (2008)
9. Cicchetti, A., di Ruscio, D., Eramo, R., Pierantonio, A.: JTL: A bidirectional and change propagating transformation language. In: SLE. pp. 183–202. No. 6563 in LNCS, Springer (2011)
10. Cunha, J., Fernandes, J.P., Mendes., J., Pacheco, H., Saraiva., J.: Bidirectional transformation of model-driven spreadsheets. In: ICMT. pp. 105–120. No. 7307 in LNCS, Springer (2012)
11. Czarnecki, K., Foster, J., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.: Bidirectional transformations: A cross-discipline perspective. In: ICMT, LNCS, vol. 5563, pp. 260–283. Springer (2009)
12. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Systems Journal 45(3), 621–645 (2006)
13. Dayal, U., Bernstein, P.: On the correct translation of update operations on relational views. ACM T. Database Syst. 7, 381–416 (1982)
14. Diskin, Z.: Algebraic models for bidirectional model synchronization. In: MoDELS, LNCS, vol. 5301, pp. 21–36. Springer (2008)
15. Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: GTTSE 2009, LNCS, vol. 6491, pp. 92–165. Springer (2011)
16. Diskin, Z., Xiong, Y., Czarnecki, K.: From state- to delta-based bidirectional model transformations: the asymmetric case. J. of Object Technol. 10, 6:1–25 (2011)

17. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: The symmetric case. In: MoDELS, LNCS, vol. 6981, pp. 304–318. Springer (2011)
18. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information preserving bidirectional model transformations. In: FASE, LNCS, vol. 4422, pp. 72–86. Springer (2007)
19. Ennals, R., Gay, D.: Multi-language synchronization. In: ESOP, LNCS, vol. 4421, pp. 475–489. Springer (2007)
20. Fegaras, L.: Propagating updates through XML views using lineage tracing. In: ICDE. pp. 309–320 (2010)
21. Foster, J., Greenwald, M., Moore, J., Pierce, B., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM T. Progr. Lang. Sys. 29(3), 17 (2007)
22. Foster, J., Pierce, B., Zdancewic, S.: Updatable security views. In: CSF-22. pp. 60–74. IEEE Computer Society (2009)
23. Foster, J., Pilkiewicz, A., Pierce, B.: Quotient lenses. In: ICFP. pp. 383–396. ACM (2008)
24. Giese, H., Wagner, R.: Incremental model synchronization with triple graph grammars. In: MoDELS, LNCS, vol. 4199, pp. 543–557. Springer (2006)
25. Hermann, F., Ehrig, H., Orejas, F., Czarnecki, K., Diskin, Z., Xiong, Y.: Correctness of model synchronization based on triple graph grammars. In: MoDELS, LNCS, vol. 6981, pp. 668–682. Springer (2011)
26. Hermann, F., Voigtländer, J.: First International Workshop on Bidirectional Transformations (BX 2012): Preface. ECEASST 49 (2012)
27. Hidaka, S., Hu, Z., Inaba, K., Kato, H., Matsuda, K., Nakano, K.: Bidirectionalizing graph transformations. In: ICFP. pp. 205–216. ACM (2010)
28. Hofmann, M., Pierce, B., Wagner, D.: Symmetric lenses. In: POPL. pp. 371–384. ACM (2011)
29. Hofmann, M., Pierce, B., Wagner, D.: Edit lenses. In: POPL. pp. 495–508. ACM (2012)
30. 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)
31. Hu, Z., Schürr, A., Stevens, P., Terwilliger, J.F.: Dagstuhl Seminar on Bidirectional Transformations (BX). SIGMOD Record 40(1), 35–39 (2011)
32. Kawanaka, S., Hosoya, H.: biXid: a bidirectional transformation language for XML. In: ICFP. pp. 201–214. ACM (2006)
33. Kennedy, A.: Pickler combinators. J. of Funct. Program. 14, 727–739 (2004)
34. Liu, D., Hu, Z., Takeichi, M.: Bidirectional interpretation of XQuery. In: PEPM. pp. 21–30. ACM (2007)
35. Macedo, N., Cunha, A.: Implementing QVT-R bidirectional model transformations using Alloy. In: FASE (2013), to appear
36. Matsuda, K., Hu, Z., Nakano, K., Hamana, M., Takeichi, M.: Bidirectionalization transformation based on automatic derivation of view complement functions. In: ICFP. pp. 47–58. ACM (2007)
37. Meertens, L.: Designing constraint maintainers for user interaction (1998), manuscript available at `http://www.kestrel.edu/home/people/meertens`
38. Melnik, S., Adya, A., Bernstein, P.: Compiling mappings to bridge applications and databases. In: SIGMOD. pp. 461–472. ACM (2007)
39. Mu, S.C., Hu, Z., Takeichi, M.: An algebraic approach to bi-directional updating. In: APLAS, LNCS, vol. 3302, pp. 2–20. Springer (2004)

40. Pacheco, H.: Bidirectional Data Transformation by Calculation. Ph.D. thesis, University of Minho (July 2012)
41. Pacheco, H., Cunha, A.: Generic Point-free Lenses. In: MPC. LNCS, vol. 6120, pp. 331–352. Springer (2010)
42. Pacheco, H., Cunha, A., Hu, Z.: Delta lenses over inductive types. ECEASST 49 (2012)
43. Schürr, A., Klar, F.: 15 years of triple graph grammars. In: ICGT, LNCS, vol. 5214, pp. 411–425. Springer (2008)
44. Stevens, P.: Bidirectional Model Transformations in QVT: Semantic Issues and Open Questions. In: MoDELS, LNCS, vol. 4735, pp. 1–15. Springer (2007)
45. Stevens, P.: A landscape of bidirectional model transformations. In: GTTSE 2007, LNCS, vol. 5235, pp. 408–424. Springer (2008)
46. Takeichi, M.: Configuring bidirectional programs with functions. In: IFL (2009)
47. Terwilliger, J., Cleve, A., Curino, C.: How clean is your sandbox? In: ICMT. LNCS, vol. 7307, pp. 1–23. Springer (2012)
48. Terwilliger, J., Delcambre, L., Logan, J.: Querying through a user interface. Data Knowl. Eng. 63(3), 774–794 (2007)
49. Voigtländer, J.: Bidirectionalization for free! In: POPL. pp. 165–176. ACM (2009)
50. Voigtländer, J., Hu, Z., Matsuda, K., Wang, M.: Combining syntactic and semantic bidirectionalization. In: ICFP. pp. 181–192. ACM (2010)
51. Wadler, P.: Views: a way for pattern matching to cohabit with data abstraction. In: POPL. pp. 307–313. ACM (1987)
52. Wang, M., Gibbons, J., Matsuda, K., Hu, Z.: Gradual refinement: blending pattern matching with data abstraction. In: MPC. pp. 397–425. Springer (2010)
53. Wang, M., Gibbons, J., Wu, N.: Incremental updates for efficient bidirectional transformations. In: ICFP. pp. 392–403. ACM (2011)

Feature / Approach	Scheme				Properties								
	Symmetry	$\overrightarrow{\subset}\ \overleftarrow{\subset}$	$\overleftarrow{\top}\ \overrightarrow{\top}$	R	Stable	Invertible	Convergent	Undoable	History Ignorant	Correct	Hippocratic	Least-update	Total
Brabrand et al. (2008) [8]	S	S	N	E		⇢⇠				⇄			⇄
Kawanaka and Hosoya (2006) [32]	S	S	N	E						⇄			⇄
Ehrig et al. (2007) [18]	S	S	D	E		⇄				⇄			⇄
Wadler (1987) [51]	S	S	N	T		⇄							
Atanassow and Jeuring (2007) [2]	S	S	N	I		⇄							
Kennedy (2004) [33]	A	S	N	T		→⇠							
Terwilliger et al. (2007) [48]	A	S E	N	T		→							→⇠
Cunha et al. (2012) [10]	A	E	N	E	⇄	→			⇄	⇄	←		⇄
Mu et al. (2004) [39]	A	S E	N	T		→	⇠						
Berdaguer et al. (2007) [5]	A	S	N	T		→							→⇠
Wang et al. (2010) [52]	A	S	N	T		←							⇄
Foster et al. (2007) [21]	A	S	N S	T	⇄	←							⇄
Bohannon et al. (2006) [7]	A	S	N S	T	⇄	←							⇄
Bohannon et al. (2008) [6]	A	S	N S	T	→⇠	←							⇄
Foster et al. (2008) [23]	A	S	N S	T	⇢⇠	⇠							⇄
Barbosa et al. (2010) [4]	A	S D	N D	T	⇄	←							⇄
Hu et al. (2008) [30]	A	S E	N S	T			⇢⇠						→
Liu et al. (2007) [34]	A	S E	N S	T	⇄	←							→
Hidaka et al. (2010) [27]	A	S E	N S	T	←	⇠							→
Takeichi (2009) [46]	A	S	N S	I	←								→
Matsuda et al. (2007) [36]	A	S	N C	T	⇄	←			←				→
Voigtländer (2009) [49]	A	S	N S	T	⇄	←			←				→
Voigtländer et al. (2010) [50]	A	S	N S	T	⇄	←							→
Fegaras (2010) [20]	A	S E	N D	T	←	⇠							→
Melnik et al. (2007) [38]	A	S	N S	E	⇄	←							⇢←
Diskin et al. (2011) [16]*	A	D	N S	T	⇄	←			⇄				→⇠
Wang et al. (2011) [53]*	A	S F	N S	T	⇄	←		←					→
Pacheco and Cunha (2010) [41]	A	S	N S	T	⇄	←							⇄
Pacheco and Cunha (2012) [42]	A	D	N S	T	⇄	←							⇄
Meertens (1998) [37]	S	S	S	E						⇄	⇄		⇄
Meertens (1998) [37]	S	E	S	E						⇄		⇄	⇄
Stevens (2007) [44]*	S	S	S	E				⇄		⇄	⇄		⇄
Macedo and Cunha (2013) [35]	S	S	S	E						⇄		⇄	⇢⇠
Hofmann et al. (2011) [28]	S	S	C	I	⇄								⇄
Hofmann et al. (2012) [29]	S	E	C	I						⇄			⇢⇠
Diskin et al. (2011) [17]*	S	D	D	T			⇄	⇢⇠		⇄			
Hermann et al. (2011) [25]	S	D	D	E			⇄			⇄			⇢⇠
Cicchetti et al. (2011) [9]	S	S	S	E	⇄					⇢⇠			⇄
Ennals and Gay (2007) [19]	S	S	S	I	⇢⇠					⇄			⇄

Table 6: Comparison of existing BX approaches.