Incorporating Uncertainty into Bidirectional Model Transformations and their Delta-Lens Formalization

Zinovy Diskin[1,2] Romina Eramo[3] Alfonso Pierantonio[3] Krzysztof Czarnecki[2]

[1]NECSIS,
McMaster University, Canada
diskinz@mcmaster.ca

[2]Generative Software Development Lab,
University of Waterloo, Canada
{zdiskin|kczarnec}@gsd.uwaterloo.ca

[3]Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
Università degli Studi dell'Aquila, Italy
{name.surname}@univaq.it

Abstract

In Model-Driven Engineering, bidirectional transformations are key to managing consistency and synchronization of related models. Delta-lenses are a flexible algebraic framework designed for specifying delta-based synchronization operations. Since model consistency is usually not a one-to-one correspondence, the synchronization process is inherently ambiguous, and consistency restoration can be achieved in many different ways. This can be seen as an uncertainty reducing process: the unknown uncertainty at design-time is translated into known uncertainty at run-time by generating multiple choices. However, many current tools only focus on a specific strategy (an update policy) to select only one amongst many possible alternatives, providing developers with little control over how models are synchronized. In this paper, we propose to extend the delta-lenses framework to cover incomplete transformations producing a multitude of possible solutions to consistency restoration. This multitude is managed in an intentional manner via models with built-in uncertainty.

1 Introduction

A basic assumption underlying the lens framework to bidirectional transformations (bx) is that an update policy ensuring the uniqueness of backward propagation is known to the transformation writer at design time. Each operation of forward propagation is supplied with a corresponding backward operation based on some update policy, which is a priori known to the transformation writer so that the transformation language becomes bidirectional. This idea was emphasized by the lens creators by calling the approach *linguistic* [FGM+07]. This implies that backward propagation can be modeled by a single-valued algebraic operation, and thus semantics of bx can be specified in ordinary algebraic terms as a pair of single-valued operations subject to a set of ordinary equations (laws). On this base, the entire algebraic machinery of lenses, including delta-lenses, is built.

However, this major assumption has the following major drawback. At the time of transformation writing, the policy is often not known or uncertain, but postponing the transformation design until the policy will be known is also not a good solution. To choose the policy, the user should normally have the possibility to observe

In: A. Anjorin, J. Gibbons (eds.): Proceedings of the Fifth International Workshop on Bidirectional Transformations (Bx 2016), Eindhoven, The Netherlands, April 8, 2016, published at http://ceur-ws.org

possible variations, compare them, and discuss with the stakeholders involved, so that choosing a right policy is typically a process depending on many parameters and contexts. Requiring the transformation writer to select a single update policy at the design time could be a too strong imposition, and tools based on the current lens framework gives the user little (if at all) control over the inherent uncertainty of update propagation.

A straightforward approach to enriching lenses with an uncertainty mechanism would be to consider multi-valued transformations modeled by multi-valued operations. However, multi-valued operations essentially complicate the algebraic framework, and are not easy to manage technically. Our idea is different, and allows us to manage uncertainty within the usual algebra of single-valued operations. We extend model spaces with *incomplete* (uncertain) models, each of which determines a set of its full completions, which are ordinary complete (fully-specified, certain) models. This allows us to model a multi-valued underspecified transformation by a single-valued one, which results in an incomplete/uncertain model encoding the required set of ordinary complete models. Thus, what is essentially changed is the notion of model spaces over which delta lenses operate rather than the very notion of a delta-lens (but, of course, the latter also needs a suitable adaptation).

Structure of the paper. The next section presents an example to motivate the importance of modeling uncertainty in bx. In Sect. 3, we discuss the example in terms of delta lenses and model completions. Section 4 presents a formal model along with its discussion. Related work is discussed in Sect. 5, and Sect. 6 draws conclusions and future plans.

2 Background and Motivation

A major problem that one needs to deal with in bx is that the consistency relation between model spaces is often of many-to-many or one-to-many types, so that restoring consistency is an inherently ambiguous process [Ste09, ZPH14]. Consistency violated by an update on one side, can be restored by updating the other side in multiple ways, which makes at least one direction of update propagation a multi-valued operation. For example, the budget of a complex project can be uniquely computed, but if the budget is changed, there are usually many ways to change the project to adjust it to the new budget. To make the update propagation a single-valued operation, the transformation designer should make certain assumptions (often referred to as an *update propagation policy*) ensuring uniqueness of consistency restoration in a reasonable way. Non-triviality of this problem is well known in the database literature for a long time under the name of the *view update problem* [BS81]. Indeed, the budget is a view of a project, and adjusting the project to a new budget is nothing but propagation of the view update back to the source. Below is a simple scenario illustrating the problem.

2.1 Managing budget cuts is easy...

Helen is a manager of the Human Resources (HR) department. She tracks and analyses employee scheduling and performance, and negotiates employee contracts. The budgetary resources for projects are managed by Fred, an officer of the Finance and Accounting (FA) department. HR and FA work together on the budgeting process and need an automated synchronizing system (a bx-engine), which was developed by an IT-department star Ian.

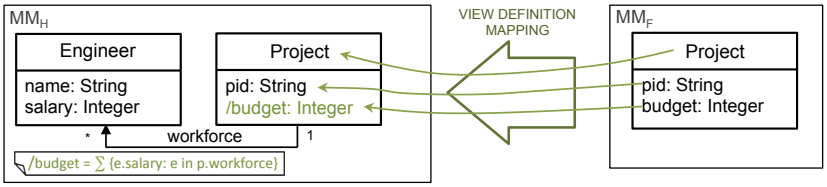

Figure 1: Metamodels and the view definition mapping.

Figure 1 depicts two metamodels developed by Ian to model Helen's and Fred's views of the project. The metamodel MM_H (in the left half of the figure) allows specifying `Project` objects with their `pid` (project identifier). Each project is associated with a number of `Engineer`s (with theirs `name` and `salary`) via association `workforce`. The metamodel MM_F (in the right half of the figure) represents Fred's view of the projects, in which only budgets are of interest. The forward transformation (*get-the-view*) is defined as follows: given a project `p`, its budget is computed by taking the sum of salaries of the engineers: $\texttt{p.budget} = \Sigma\ \{\texttt{e.salary} : \texttt{e} \in \texttt{p.workforce}\}$, and this derived attribute`/budget` is taken to be the attribute budget in MM_F. The view definition mapping defining this view consists of three links shown in Fig. 1 by curved lines, while the entire mapping is shown as a block arrow encompassing the links.

One of the company's most promising projects is *Perpetuum Mobile Engine* (PME) identified by `pid=PME`, and Fred was able to allocate a decent budget of 200 units monthly for PME. For this budget, Helen organized the

workforce of the project as shown by model A in the top left of Fig. 2: she hired two senior engineers, `Ben` and `Bill`, with equal salaries 100. Query `/budget` defined in metamodel MM_H, can be executed for model A and results in the `/budget`(A)=200.

Engineers `Ben` and `Bill` have been working hard but with a limited success, and when the company has undertaken administrative cost reductions, Fred is forced to cut the `PME` budget to `150` (as depicted in the model B'). To adapt the project to the reduced budget and restore consistency between the HR and the FA models, the system designed by Ian prescribed to cut salaries of `Ben` and `Bill` and make them each equal to 75.

Ian is a great fan of delta-lenses, and when he was tasked to implement a bx system, he implemented a delta-lenses framework, in which propagation operations use deltas as input and output rather than compute them internally. Deltas consist of links between model elements. Horizontal links specify correspondences between the elements of models to be synchronized, e.g., horizontal delta Δ_{AB} in Fig. 2 consists of three links: $L0$ relating objects, and $L01, L02$ relating the respective attributes. Note, these links are instances of the respective view definition links (i.e., associations) in Fig. 1. (Formally, instance links are just pairs of elements written with arrows.) The link $L02$ in Δ_{AB} maps attribute `budget` of model B to the *derived* attribute `/budget` in model A, and as values of the two attributes are equal, we consider models to be consistent. Vertical links specify traceability from the original to the updated model. For example, the vertical delta $\Delta_{BB'}$ in Fig. 2 consists of three links, which say that the project $q2$ with its attributes in model B' is the same project with the same *semantics* of attributes as in model B. However, as the *values* of the attribute `budget` in models B and B' are different, we conclude that the attribute `budget` is updated in B'.

As models A and B' are inconsistent (budgets are different), the delta lens invokes operation of backward update propagation called *put-update-back* or just `put`). This operation (shown in Fig. 2 by a chevron `put`, takes two deltas, Δ_{AB} and $\Delta_{BB'}$, together with their source and target models, and produces an update (vertical delta) $\Delta_{AA'}$ (only links between OIDs are shown to save space, the attribute links can be restored from them) together with an updated correspondence. Given models and deltas are shaded, while the derived model and two deltas are blank (and blue with a color display).

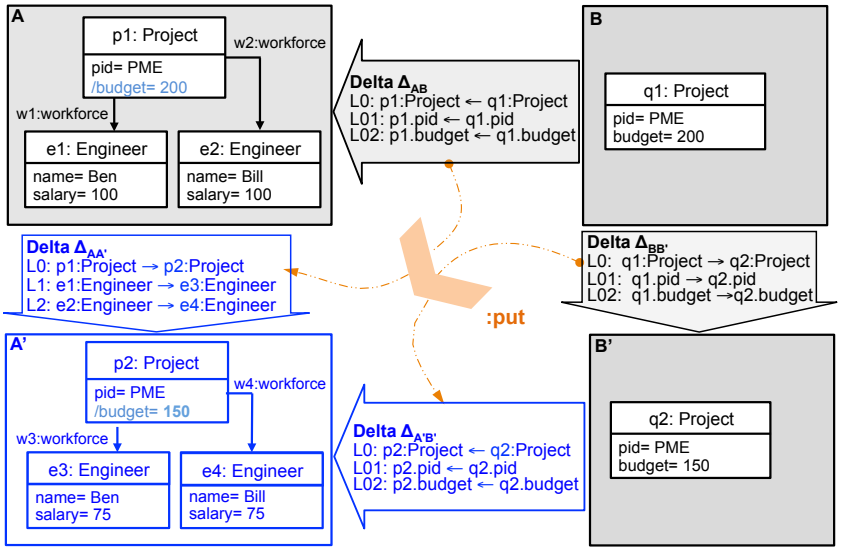

Figure 2: An example of a single-valued delta len

As there are many ways to restore consistency, to make `put` single-valued (certain, deterministic), some update policy ensuring uniqueness is needed. Ian has based his lens design on a simple and reasonable policy: a budget cut is to be propagated to cuts in the salaries of all engineers *proportionally* to the values of salaries. Since `Ben` and `Bill` have equal salaries, the cut in 50 units results in 25 cut for each of them.

2.2 ..but not as easy as it may seem.

Unfortunately, Helen was not happy with the automatic update provided by the delta lens. She told Ian she was not sure that cuts should be equal, rather, they should depend on some background information about the engineers and their contribution to the PME project, with which she still needed to familiarize herself. She also mentioned that she is disappointed with the result of synchronization via delta lenses that Ian so much praised.

After a few days of thinking, Ian found how to make the lens framework more flexible. Now `put` would produce an *incomplete* (or *uncertain*) model A'_1, in which the attributes `salary` of the engineers are uncertain but must be within a predefined range, e.g. `[65..85]`, as shown in the left part of the Fig.3. Constraint C_1 ensures that although the user can vary the salaries, their sum must be not greater than 150. (Moreover, Helen could even modify the lower and upper bounds of the ranges if needed.) Incomplete model A'_1 has multiple *final completions*, i.e., models with fixed values for the attribute `salary`, e.g., 70 for `Ben`, and 80 for `Bill`. In-between, there are

many completions that are *not-final*, e.g., [65..70] for `Ben` and [85..80] for `Bill`. Thus, the lens framework should include the notions of uncertain models and their completions, which seem to be not very difficult to manage.

Being very satisfied with the solution, Ian came to HR again, but found another problem. After studying the employee profiles, Helen found that as both Ben and Bill are senior and experienced engineers, reduction of their salary is not a good solution at all. A Helen's new idea was to keep one of the engineers with the same or even increased salary, and hire a recent graduate as his assistant for a smaller salary. Again, to keep this approach flexible, salaries should not be be fixed but rather kept within some predefined ranges. When Ian asked which of the engineers, `Ben` or `Bill`, Helen intends to keep, she answered that she would decide this later.

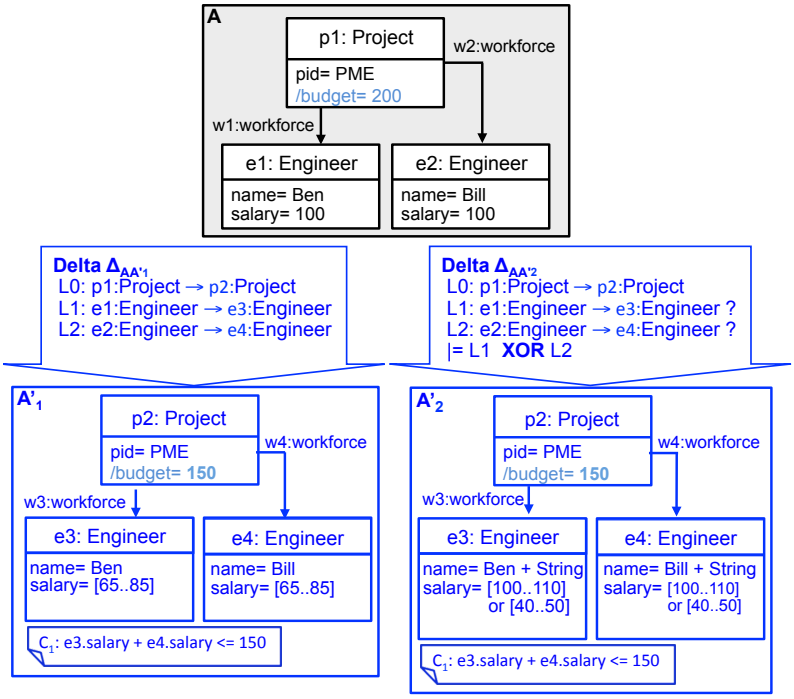

Figure 3: The incomplete models A'_1 and A'_2

After some thinking, Ian implemented a new lens satisfying Helen's new approach: this lens should produce an *incomplete* update $\Delta_{AA'_2}$ resulting in an incomplete model A'_2 as shown in the right part of Fig.3 (where + denotes the disjoint union of a singleton set and the set of strings, and union of integer ranges is denoted by "or" to avoid confusion with addition of numbers). Delta $\Delta_{AA'_2}$ is composed of three object links, two of which are optional. It means that engineer $e3$ could be `Ben`, if link $L1$ is present in the delta, or be a new engineer with unknown name, if link $L1$ is removed from the delta; similarly for the link $L2$. Moreover, the XOR constraint specified in the logical part of the delta labeled with $\models$, says that one and only one link must be present, i.e., one of the engineers `e3,e4` must be a previous engineer and one must be a new hire. In addition, we mark each link with symbol ? showing its optional existence.[1] Two other constraints on possible completions are assumed specified but not shown in the figure: if link $L1$ is present, then `e3.salary` is within the range [100..110], and if it is absent, then the salary range is [40..50]; similarly for link $L2$ and the salary range for `e4`.

However, at the moment Ian finished debugging his new bx system, Helen informed him that she talked to her boss Hilary, who said that the possibility of leaving in the project only one engineer is not excluded. Moreover, Hilary told Helen that the board of directors is thinking about cancelling the PME project, but is not certain yet, and needs to analyse some additional data to take a decision. At this point Ian became entirely convinced that incorporating uncertainty into the lens framework was an urgent task of vital importance for lenses.

3 Bx's Underspecification via Delta-Lenses with Uncertainty

A straightforward approach to enrich lenses with an uncertainty mechanism would be to consider multi-valued transformations modeled by multi-valued operations. However, multi-valued operations essentially complicate the algebraic framework, and are not easy to manage technically. Ian's (and our) idea is different: we extend model spaces with uncertain, or *incomplete* models (which can be seen as incomplete instances of the respective metamodel as described in [BDA+13, RE15]). Each such an incomplete model determines a set of its full completions, which are ordinary fully-specified models. This allows us to model a multi-valued underspecified transformation by a single-valued operation, which results in an incomplete model encoding the required set of

[1] In the presence of the XOR-formula above, ? labels are redundant yet suggestive. Moreover, they become very handy if we want to declare optional existence of both links without constraints so that there are four possible configurations: $\{L1, L2\}$, $\{L1\}$, $\{L2\}$, and $\emptyset$. Then we simply mark optionality of each of the links without further constraints.

ordinary complete models.

The example below illustrates the main ingredients of the approach. Figure 4 depicts Ian's system within the delta-lenses framework with built-in uncertainty, which we call *delta-lenses with uncertainty*. The incomplete update $\Delta_{AA'}\colon A \leftarrow A'$ represents a set of update policies discussed with Helen; in fact, it encodes two updates from Fig. 3, but canceling the entire PME project is not considered. That is, updates $\Delta_{AA'_1}\colon A \leftarrow A'_1$ and $\Delta_{AA'_1}\colon A \leftarrow A'_2$ from Fig. 3 can be seen as *non-final completions* of update $\Delta_{AA'}\colon A \leftarrow A'$: they are more complete than it, but not all the choices are fixed.

In more detail, the process of update completion is described in Fig. 5. The schema of the process is described in the upper part above the dashed line. The part below the line presents an instantiation of the schema: nodes are models, vertical and inclined arrows are update deltas, and horizontal arrows are *completion* deltas (i.e., *C-Delta*). The scheme consists of three cojoined triangles, $\triangle AA'A'_1$, $\triangle AA'_1A'_2$ and $\triangle AA'_2A'_3$, whose tiling is distorted in the lower part of the figure to fit in the page. Horizontal arrows (bases of the triangles) are model completions; more accurately, a completion is a triple consisting of a less certain model, more certain model, and the c-delta relating them.

Each of the triangles describes a completion of its right-hand side update to its left-hand side update, where an update is understood as a triple consisting of the original model, the updated model, and the delta relating them. The right-most and the left-most triangles are commutative: in each of them, the left delta is a composition of the right delta with the base c-delta. The middle triangle is not commutative: the composed link $(e2{\rightarrow}e4)_{\in\Delta_{AA'_1}}; (e4{\rightarrow}e4)_{\in C\Delta_{A'_1A'_2}}$ is not included into $\Delta_{AA'_2}$. We can understand this as first composing two deltas $\Delta_{AA'_1}; C\Delta_{A'_1A'_2}$, and then transforming the result by the removal of the link specified above.

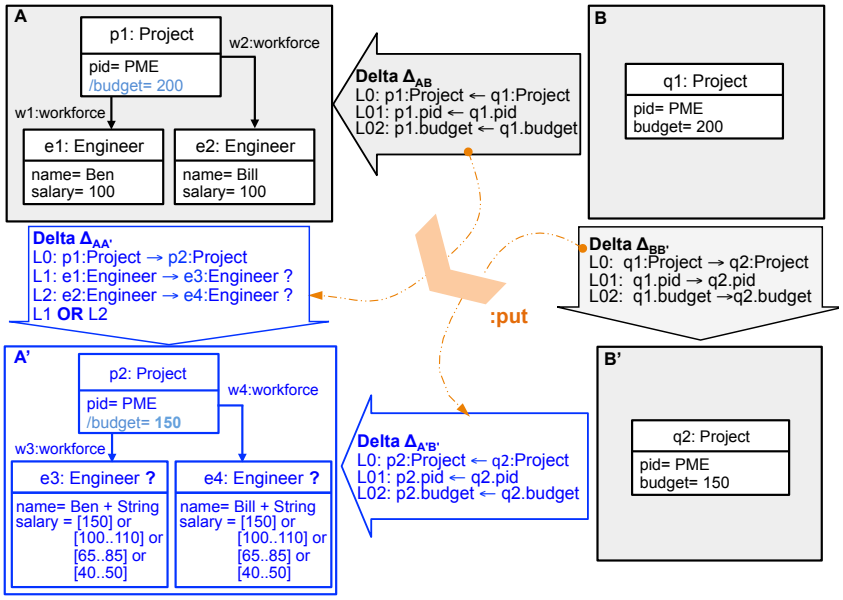

Figure 4: A round-trip scenario with uncertainty.

Models, in general, consist of two parts: a set of objects and links (called an object diagram in UML), and an uncertainty mechanism; the latter, in its turn, may have three components: (a) ranges of possible values for some attributes (e.g., salary in model A'), (b) optionality of some objects (e.g., in model A', optionality of objects $e3$, $e4$ is shown by ? marks), and (c) logical formulas that constrain possible completions. E.g., formula $e3 \vee e4$ says that a completion in which both engineers are removed is prohibited (although the existence of each of the engineers is optional). Update deltas, in general, also consist of two parts: a set of links that specifies a mapping (relation), and a logical formula (marked with $\models$) that constrains possible completions. For example, formula $L1 \vee L2$ in $\Delta_{AA'}$ says that the delta can be completed by keeping one or both links. In addition, the update specification includes logical formulas relating uncertainty of both the delta and the updated model. E.g., for incomplete update $\Delta_{AA'}\colon A \leftarrow A'$, if in a possible completion, link $L1$ is kept but both link $L2$ and object $e4$ are deleted, then the salary of object $e3$ in this completion must be 150, while if both links are kept, then salaries of e3 and e4 are to be within the range [65..85]. These inter-delta-model constraints again show that an update is a triple of objects rather than just its delta.

The completion specified by triangle $AA'A'_1$ is the following one. The updated model must contain two engineers (with commutativity of the triangle and removal of the salary value 150), but the option of keeping both previous engineers is now excluded due to the constraint $L1$ xor $L2$ in delta $\Delta_{AA'_1}$, which is stronger than constraint $L1 \vee L2$. All this makes update $\Delta_{AA'_1}$ more certain, or more complete, than update $\Delta_{AA'}$. However, it is not fully certain in that which of the engineers is to be kept is still not decided The next completion triangle decides to keep `Ben` and hire a recent graduate, but their exact salaries are still not certain. Finally, the third

triangle fixes exact salaries and thus results in a (fully) complete update $\Delta_{AA'_3}$: $A \leftarrow A'_3$ targeting at a (fully) complete model A'_3.

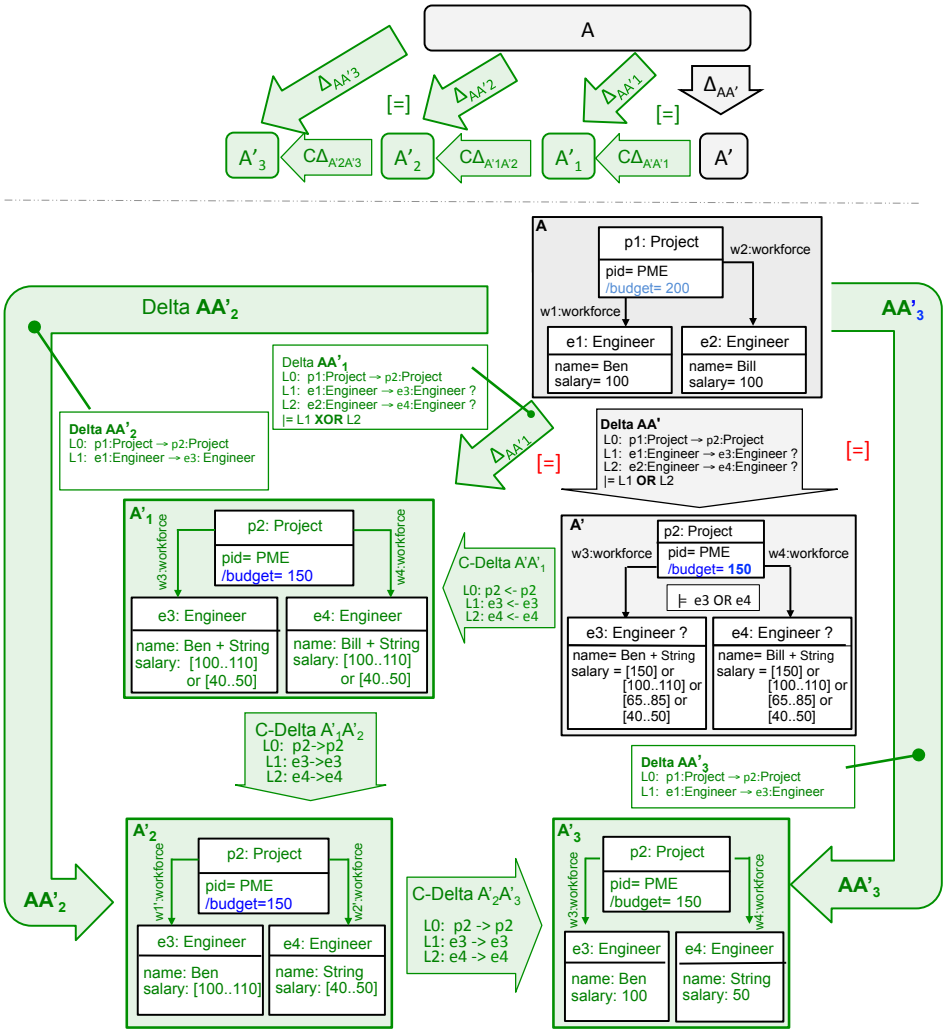

Figure 5: A multi-valued model and its completions within the delta-lenses framework

4 Formal framework: Incorporating uncertainty into asymmetric delta-lenses

The first issue in building incomplete bx is a suitable enrichment of the notion of a model space. However, building a formal model of incomplete models and their completions is challenging —as challenging as many other problems of dealing with uncertainty. These problems are a classical subject of database and AI research, whose foundations are still debated. Specifically, in a recent series of seminal papers [Lib14b, Lib14a, Lib15], Leonid Libkin showed a major deficiency of a widely accepted common approach to uncertainty (the so called *certain answers*), and proposed a simple elegant framework to fix the deficiency. In a nutshell, Libkin proposed to view incomplete models (databases) as both semantic objects (i.e., as *models* in the parlance of the classical model theory) and syntactic knowledge (i.e., *theories*) about the objects, and built a corresponding formal framework.

The two-fold view of uncertain models is smoothly integrated in our approach to uncertainty originated in [BDA+13]. In a nutshell, we formally model the process of models completion by arrows in a suitable category, so that progressing from an uncertain model A to its particular full completion can be seen as an arrow chain $\mathsf{c} = (A^{\mathsf{c}}_0 \to A^{\mathsf{c}}_1 \to A^{\mathsf{c}}_2 \to ... \to A^{\mathsf{c}}_{\dashv})$ with $A^{\mathsf{c}}_0 = A$ and $A^{\mathsf{c}}_{\dashv}$ being a fully certain model. Each model A_k in this chain besides the last one ($k = 0, 1, ...$) can be a branching point of the completion process, which can go into another direction ($A_k = A^{\mathsf{c}'}_0 \to A^{\mathsf{c}'}_1 \to ... \to A^{\mathsf{c}'}_{\dashv}$) terminating at another fully certain model $A^{\mathsf{c}'}_{\dashv}$, and so on. In general, even small finite uncertain models can have an infinite number of completions exactly like small finite theories can have an infinite number of models. For any two-arrow fragment of a completion chain, $A_{n-1} \to A_n \to A_{n+1}$,

model A_n can be seen as both a semantic model for theory A_{n-1} and a theory of model A_{n+1}. Fully certain models are thus theories of themselves. In this section, we develop the arrow view of completion into a formal framework of model spaces with uncertainty, and show how the notion of a delta lens could be adapted to work over such spaces and respect uncertainty.

Our plan is as follows. Section 4.1 presents our categorical notation, sometimes not quite standard. Special attention is paid to an elementary construction of arrow triangles, which is heavily used later. In Sect. 4.2, we formalize the completion process sketched above with the notion of *u-space* ('u' stands for uncertainty). A logic for u-spaces is introduced in Sect. 4.3. However important are uncertain models and their completions, our main objects of interest in BX are updates, and they can also be uncertain as we have seen in our running example. Hence, in Sect. 4.4 we introduce *uu-spaces*, which formalize both uncertain model and uncertain update completions (hence, 'uu' in their name). Update completions are formalized as special arrow triangles, hence the attention to the construct in Sect. 4.1.3. Section 4.5 is the culmination. We introduce two constructs to relax the rigidity of ordinary single-valued update policies. The first one, *multi-valued update policies*, do it in a direct semantic way with the notion of a multi-valued operation. The second one, *uu-lenses*, do it indirectly (and we may say syntactically) by encoding a multitude of models by a suitable uncertain model. Our main results specify a pair of adjoint functors between the categories of multi-valued policies and uu-lenses built over the same given view functor get, so that the two notions are, in a sense, equivalent. Thus, uu-lenses can be seen as a sound and complete representation of multi-valued update policies.

Several words about formalities as such. As is traditional for the lens framework, the mathematical structures are very general, and specify basic relationships and operations over models and updates without saying what models and updates really are. This abstraction allows one to instantiate lenses with rather different notions of a model, e.g., models can be attributed typed graphs, relational databases, semi-structured data/XML documents, or models of a FOL theory, enriched with an uncertainty specification mechanism. In stating the axioms (laws) our structures should satisfy, we were trying to respect the following requirements. First, we wanted to exclude what we feel are wrong examples, and allow for (what we feel are) the right ones. Checking the axioms for concrete instantiations is a work in progress, some results can be found in [Dis16]. Second, the laws should allow us to prove our results in a sufficiently direct way. We did not strive to present an economical system of axioms—it is possible that some of them can be inferred from the others, nor we strived to find the most economical presentation. Some unintended omisisons are, unfortunately, also possible: the formal section of the paper was significantly extended after submission, and so far the proofs were not peer reviewed. Overall, as both checking our formal requirements against major instantiations and completing the proofs are a work in progress, our formal defintions are somewhat experimental and subject to change. We consdier the paper as a working paper for a working workshop. The accompanying technical report [Dis16] is intended to provide a maintained version of the formal framework for some time onwards.

4.1 Categorical preliminaries and notation

To simplify technicalities and avoid size problems, we will assume all our categories to be small, and use terms set and class interchangeably.

4.1.1 Objects and arrows. If $\boldsymbol{C}$ is a category, $\boldsymbol{C}^\bullet$ denotes its class of objects, and for $A, B{\in}\boldsymbol{C}^\bullet$, $\boldsymbol{C}(A,B)$ is the set of morphisms/arrows from A to B. Also, $\boldsymbol{C}(A,*)$ is $\bigcup_{B\in\boldsymbol{C}^\bullet}\boldsymbol{C}(A,B)$, and dually, $\boldsymbol{C}(*,B)$ is $\bigcup_{A\in\boldsymbol{C}^\bullet}\boldsymbol{C}(A,B)$, Let then $\boldsymbol{C}$ be the class of all arrows denoted by the same letter as the entire category (i.e., writing $u{\in}\boldsymbol{C}$ means that u is an arrow, while objects are elements of $\boldsymbol{C}^\bullet$). If $u\colon A\to B$, we write ${}^\bullet u$ for A and $u^\bullet$ for B. We write $u\colon A\to *$ and $u\colon *\to B$ for elements of $\boldsymbol{C}(A,*)$ and $\boldsymbol{C}(*,B)$ resp. Sequential composition of $u\colon A\to B$, $v\colon B\to C$ is denoted by $u;v$, and the identity of object A is id_A. If prop is a property of $\boldsymbol{C}$-arrows, the class $\{u{\in}\boldsymbol{C}\colon u\models\mathsf{prop}\}$ is denoted by $\boldsymbol{C}_{\mathsf{prop}}$, which gives us three subcategories $\boldsymbol{C}_{\mathsf{iso}},\boldsymbol{C}_{\mathsf{epi}},\boldsymbol{C}_{\mathsf{mono}}$. Isomorphic objects are denoted by $A\cong B$, and let $A^{\cong}$ denote A's *iso-closure* $\{B{\in}\boldsymbol{C}^\bullet\colon B\cong A\}$. If $\boldsymbol{C}'$ is a subcategory of $\boldsymbol{C}$ with the same class of objects, i.e., $\boldsymbol{C}'\subset\boldsymbol{C}$ but $\boldsymbol{C}'^\bullet=\boldsymbol{C}^\bullet$, we write $\boldsymbol{C}'\subseteq_\bullet\boldsymbol{C}$ and say $\boldsymbol{C}'$ is an *object-full* subcategory of $\boldsymbol{C}$.

Given a functor (indexed category) $f\colon\boldsymbol{C}\to\boldsymbol{Cat}$ and an object $A{\in}\boldsymbol{C}^\bullet$, we write $f^\bullet A$ for $(fA)^\bullet$. Thus, fA is a category while $f^\bullet A$ is its class of objects. For two parallel functors $f',f\colon\boldsymbol{C}\to\boldsymbol{Cat}$, we say f' is an *object-full subfunctor* of f and write $f'\subseteq_\bullet f$ if $f'A\subseteq_\bullet fA$ for all $A{\in}\boldsymbol{C}^\bullet$ and, for any $u\colon A\leftarrow B$ in $\boldsymbol{C}$, reindexing $f'u\colon f'A\to f'B$ is the restriction of reindexing $fu\colon fA\to fB$ to $f'A$.

4.1.2 Families, spans and products. Given a set X, a family of its elements is denoted by expression $\mathbf{x}=\{\!\!\{x_i{\in}X\colon i\in I\}\!\!\}$ with double brackets, while the set of elements participating in the family is denoted by expression $\mathbf{x}^{\#}=\{x_i{\in}X\colon i\in I\}$ with single brackets; we will refer to this set as to the *member set* of the family.

Even if the family at hand is injective, i.e., $x_i \neq x_j$ for $i \neq j$, and hence $\mathbf{x}$ is a bijection $\mathbf{x}\colon I \to \mathbf{x}^{\#}$, we still denote the family with double brackets. Any subset $Y \subset X$ of the carrier set gives rise to a family $\{\!\{x_y{\in}X\colon y \in Y\}\!\}$ with $x_y = y$ for all $y{\in}Y$.

A *span* in a category $\boldsymbol{C}$ is a family of arrows $\mathbf{u} = \{\!\{u_i\colon H \to * \in \boldsymbol{M}\colon i \in I_{\mathbf{u}}\}\!\}$ with a common source called the *head* of the span and denoted by ${}^{\bullet}\mathbf{u}$; the arrows are called *legs*, and their targets are *feet* of the spans. The family of feet is denoted by $\mathbf{u}^{\bullet} = \{\!\{u_i{}^{\bullet}\colon i{\in}I_{\mathbf{u}}\}\!\}$ (note that it may be non-injective family if even the family of legs is injective). Given a set J and a function $f\colon J \to I_{\mathbf{u}}$, *reindexing* of $\mathbf{u}$ along f is the span $\{\!\{u_{fj}\colon j{\in}J\}\!\}$ denoted by $f^*\mathbf{u}$.

Given a family of $\boldsymbol{C}$-objects $\mathbf{A} = \{\!\{A_i{\in}\boldsymbol{C}^{\bullet}\colon\ i \in I\}\!\}$, its (chosen) product is a span $\Pi\mathbf{A} = \{\!\{\Pi_i\mathbf{A}{\in}\boldsymbol{C}({}^{\bullet}\Pi\mathbf{A}, A_i)\colon i \in I\}\!\}$, whose head is denoted by ${}^{\bullet}\Pi\mathbf{A}$, and legs $\Pi_i\mathbf{A}$ are called *projections* (note that the product span uses the same indexing set I). We will follow a common practice and denote the head of the product span and the entire span by the same letter Π, but sometimes we will use ${}^{\bullet}\Pi$ for the head if we want to emphasize that we are talking about an object. Products are called *idempotent*, if for any family of isomorphic objects $\mathbf{A} = \{\!\{A_i{\in}\boldsymbol{C}^{\bullet}\colon i \in I\}\!\}$ with $A_i \cong A_j$ for any two $i, j \in I$, all projections $\Pi_i\mathbf{A}$ are isomorphisms. Given a span $\mathbf{u}$, universality of products means a uniquely determined arrow $\mathbf{u}!\colon {}^{\bullet}\mathbf{u} \to {}^{\bullet}\Pi(\mathbf{u}^{\bullet})$ commuting with projections (i.e., a span morphism). Specifically, any function $f\colon J \to I_{\mathbf{u}}$ gives rise to a unique span morphism $f!\colon \Pi\mathbf{u} \to \Pi(f^*\mathbf{u})$.

A functor $f\colon \boldsymbol{C} \to \boldsymbol{D}$ between categories maps any span $\mathbf{u}$ to a span $f\mathbf{u} = \{\!\{f(u_i)\colon i{\in}I_{\mathbf{u}}\}\!\}$. Functor f is said to *preserve products (up to iso)* if for any family of $\boldsymbol{C}$-objects $\mathbf{A}{=}\{\!\{A_i\colon i \in I\}\!\}$, we have $f(\Pi\mathbf{A}) = \Pi(f\mathbf{A})$ (or just $f(\Pi\mathbf{A}) \cong \Pi(f\mathbf{A})$).

4.1.3 Triangle functors. We will heavily use the following simple construction.

Given an object $A{\in}\boldsymbol{C}^{\bullet}$, symbol $\boldsymbol{T}A$ denotes a *triangle* category, whose objects are arrows $u\colon A \to *$, and arrows are triples (u, u', b) with $u, u' \in \boldsymbol{C}(A, *)$ and $b\colon u^{\bullet} \to u'^{\bullet}$. Such a triple can be seen as a triangle with vertexes A (called the *main* vertex, $u^{\bullet}$, $u'^{\bullet}$, *sides* u and u', and the *base* b; we will call them *triangles* and denote by double arrows $\tau_b\colon u \Rightarrow u'$ with the subscript referring to the base. Thus, $\boldsymbol{T}^{\bullet}A\,(u, u') = \boldsymbol{C}(u, u')$ where we write $\boldsymbol{T}^{\bullet}A$ for $(\boldsymbol{T}A)^{\bullet}$, and $\boldsymbol{T}A\,(u, u') \cong \boldsymbol{C}(u^{\bullet}, u'^{\bullet})$. Note that we cannot assume equality $\boldsymbol{T}A\,(u, u') = \boldsymbol{C}(u^{\bullet}, u'^{\bullet})$ because two triangles from different triangle categories, say, $(u, u', b){\in}\boldsymbol{T}A$ and $(v, v', b){\in}\boldsymbol{T}B$, can share the same base b. To make this explicit, we will sometimes write triangles as quadruples (A, u, u', b) with the first component being the main vertex, or via arrows as $\tau_b^A\colon u \Rightarrow^{\preccurlyeq} u'$. Triangle composition is given by tiling: for $\tau_b\colon u \Rightarrow u'$ and $\tau_{b'}\colon u' \Rightarrow u''$, we have $\tau_{b;b'}\colon u \Rightarrow u''$. The identity $\mathsf{id}_u\colon u \Rightarrow u$ is the triangle with base $\mathsf{id}_{u^{\bullet}}$. The subcategory of commutative triangles is denoted by $\boldsymbol{T}_{[=]}A$, and it is nothing but the slice category $A\backslash\boldsymbol{C}$.

Any arrow $f\colon A \leftarrow B$ gives rise to a *reindexing functor* $f^*\colon \boldsymbol{T}A \to \boldsymbol{T}B$ in an obvious way: an arrow $u\colon A \to *$ is mapped to $f^*u \stackrel{\text{def}}{=} f; u\colon B \to *$, and triangle $\tau_b^A\colon u \Rightarrow^{\preccurlyeq} u'$is mapped to triangle $\tau_c^B\colon f^*u \Rightarrow^{\preccurlyeq} f^*u'$ with the same base b. We will call this construction *reindexing by precomposition*. Thus, any category $\boldsymbol{C}$ is equipped with a *triangle* functor $\boldsymbol{T}\colon \boldsymbol{C} \to \boldsymbol{Cat}^{\mathsf{op}}$. We will write $\boldsymbol{T}_{\boldsymbol{C}}$ when we need to make the carrier category explicit.

Given an object-full subcategory $\boldsymbol{C}' \subseteq_{\bullet} \boldsymbol{C}$, for each object A we have an arrow subcategory $\boldsymbol{T}_{\boldsymbol{C}'}A \subseteq_{\bullet} \boldsymbol{T}_{\boldsymbol{C}}A = \boldsymbol{T}A$ whose triangle bases are taken from $\boldsymbol{C}'$, i.e., $\boldsymbol{T}_{\boldsymbol{C}'}A$ arrows are triples (u, v, b') with $u, v \in \boldsymbol{C}(A, *)$ and $b'{\in}\boldsymbol{C}'(u^{\bullet}, v^{\bullet})$.[2] As $\boldsymbol{C}'$ is a subcategory of $\boldsymbol{C}$, triangle tiling and identities are inherited from $\boldsymbol{T}A$. Reindexing along $f{\in}\boldsymbol{C}(A, B)$ is obviously a functor $f^*\colon \boldsymbol{T}_{\boldsymbol{C}'}A \leftarrow \boldsymbol{T}_{\boldsymbol{C}'}B$, and we thus have an indexed category $\boldsymbol{T}_{\boldsymbol{C}'}\colon \boldsymbol{C} \to \boldsymbol{Cat}^{\mathsf{op}}$ such that $\boldsymbol{T}_{\boldsymbol{C}'} \subseteq_{\bullet} \boldsymbol{T}$.

Finally, any functor $f\colon \boldsymbol{C} \to \boldsymbol{D}$ gives rise to a natural transformation $\boldsymbol{T}f\colon \boldsymbol{T}_{\boldsymbol{C}} \Rightarrow f; \boldsymbol{T}_{\boldsymbol{D}}$ with a family of functors $\boldsymbol{T}f_A\colon \boldsymbol{T}_{\boldsymbol{C}}(A) \to \boldsymbol{T}_{\boldsymbol{D}}(fA)$, $A{\in}\boldsymbol{C}^{\bullet}$ defined in an obvious way. We will write $f_A^{\boldsymbol{T}}$ for $\boldsymbol{T}f_A$ to ease notation when $\boldsymbol{T}f_A$ is applied to an argument.

4.2 Model spaces with uncertainty, I: uncertain models and their completions

Definition 1 (U-spaces) A *space of uncertain models* or just an *u-space* is a pair $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ with $\boldsymbol{M}$ a category of *(possibly uncertain) models* and their *updates (deltas)*, and $\boldsymbol{M}_{\preccurlyeq} \subseteq_{\bullet} \boldsymbol{M}$ an object-full subcategory of $\boldsymbol{M}$. Arrows of $\boldsymbol{M}_{\preccurlyeq}$ are called *certainty non-decreasing* updates, or *(model) completions*, or just $\preccurlyeq$-arrows, and we write $c\colon A \to^{\preccurlyeq} B$ in order to say that $c \in \boldsymbol{M}_{\preccurlyeq}(A, B)$.

[2] Warning: a more accurate notation for $\boldsymbol{T}_{\boldsymbol{C}'}$ would be $\boldsymbol{T}_{C'\subseteq_{\bullet}C}$ as $\boldsymbol{T}_{\boldsymbol{C}'}$ could be understood as the triangle functor of category $\boldsymbol{C}'$ alone, whose triangles have all three sides taken from $\boldsymbol{C}'$, whereas we deal with the situation when only the triangle bases are C'-arrows. A similar concern appears later with reindexing, which is considered along any $\boldsymbol{C}$-arrows, not just $\boldsymbol{C}'$-arrows. As we will never deal with the case of $\boldsymbol{C}'$ considered as an independent category knowing nothing about $\boldsymbol{C}$, we can safely use the shorter notation.

A *morphism* from u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ to u-space $(\boldsymbol{N}, \boldsymbol{N}_{\preccurlyeq})$ is a functor $f: \boldsymbol{M} \to \boldsymbol{N}$ that preserves $\preccurlyeq$-arrows: if $c \in \boldsymbol{M}_{\preccurlyeq}$, then $f(c) \in \boldsymbol{N}_{\preccurlyeq}$. Below are some motivational discussions and additional details.

Models and updates. A more accurate name for $\boldsymbol{M}$'s arrows would be *co-deltas*, as we understand them as spans specifying the common part of the source and the target models rather than their difference (for which the term *delta* typically used), but we will stick to the established bx terminology. We also call arrows *updates* as in our context the target model of an arrow is understood as an updated version of the source model, and the delta specifies the non-changed data common for both models.

Having isomorphism arrow $i: A \to B$ in $\boldsymbol{M}$ means that models A and B are basically the same; in our main instantiation of the framework, isomorphic models are isomorphic attributed graphs that only differ in OIDs and null labels, and have the same values in all certain attribute slots.

Both models and updates are understood as possibly *uncertain* or *incomplete* as demonstrated by our examples above. More formally, objects can be optional (marked with question marks), and attribute values can be *labeled nulls* rather than actual values (in the database literature, actual values are often called *constants*). In addition, both models and updates include a set of logical formulas regulating possible completions of its uncertain elements. We will consider several examples below in Ex. 3 (see also [BDA+13]).

Model completions. Subcategory $\boldsymbol{M}_{\preccurlyeq}$ encompasses very special updates. Intuitively, we assume that $\preccurlyeq$-updates can do the following four types of changes: (i) narrow the possible value range for a null value (specifically, make it a singleton, which means substituting a constant for the null); (ii) glue together two nulls into a null with a suitable value range; (iii) delete an optional OID; (iv) glue together two optional OIDs. Example 3 below shows these possibilities. Of course, updates that delete or glue together non-optional model elements are also possible, but such updates are not $\preccurlyeq$-arrows. Updates that add new elements to a model are not $\preccurlyeq$-arrows either. In this sense, our uncertainty semantics is of the CWA type (Closed World Assumption), while the OWA semantics (Open World Assumption) can be simulated by composing a $\preccurlyeq$-arrow with an insert update. Thus, assuming the CWA semantics means that $\preccurlyeq$-arrows are surjections, which we formalize by requiring them to be $\boldsymbol{M}$-epimorphisms, i.e., we require $\boldsymbol{M}_{\preccurlyeq} \subset \boldsymbol{M}_{\mathsf{epi}}$.

To formalize the intuition of $\preccurlyeq$-arrows as certainty non-decreasing , we impose two additional conditions. First, we require any isomorphism $i: A \to B$ to be an $\preccurlyeq$-arrow: being isomorphic, both models should be equally (un)certain, hence, $\boldsymbol{M}_{\mathsf{iso}} \subset \boldsymbol{M}_{\preccurlyeq}$. Second, we require that having $c: A \to^{\preccurlyeq} B$ and $c': A \stackrel{\succcurlyeq}{\leftarrow} B$ imply $c \in \boldsymbol{M}_{\mathsf{iso}}$ and $c' \in \boldsymbol{M}_{\mathsf{iso}}$ (although mutual invertibility of c and c' isn't required). Finally, we require that $\boldsymbol{M}^{\bullet} = \boldsymbol{M}^{\bullet}_{\preccurlyeq}$ as any model can be the source or the target of a $\preccurlyeq$-arrow, e.g., the model's identity loop.

Definition 2 (full completions) If the only $\preccurlyeq$-arrows leaving model A are isomorphisms, $\boldsymbol{M}_{\preccurlyeq}(A, *) = \boldsymbol{M}_{\mathsf{iso}}(A, *)$, we call A *(fully) certain* or *complete*. We denote the class of all complete models by $\boldsymbol{M}^{\bullet}_{\dashv}$, where symbol $\dashv$ is used to suggest the termination of the completion process at complete models. We write $c: A \to^{\preccurlyeq} \dashv$ to say that $c^{\bullet} \in \boldsymbol{M}^{\bullet}_{\dashv}$, and call such an update c a *(full) completion* of A; let $\mathsf{FC}_{\preccurlyeq}(A)$ denotes the class of all such. We call model A *consistent* if $\mathsf{FC}_{\preccurlyeq}(A) \neq \varnothing$.

The following simple example illustrates the notions introduced above.

Example 3 Suppose given a metamodel that specifies a class R with two attributes a, b, whose values are positive integers. Suppose that for objects instantiating class R, both attributes are declared to be optional (in this sense R is a semi-structured relation, which can be instantiated with tuples having one or no components). Moreover, we assume that the metamodel can be instantiated with uncertain data, in which the existence of both objects and attribute slots can be optional, and attribute slots can hold nulls rather than actual values. We call such an instantiation an uncertain model, and Fig. 6 shows several simple examples.

Model A (the second left) consists of two objects $r1, r2$, both are assumed optional (note two ?-marks) and can be deleted in a legal completion. Attribute a of object $r1$ (we will write $a@r1$) is uncertain and represented by a labeled null $\perp_1$: in a legal completion, any value allowed by the metamodel (i.e., a positive integer in our case) can be substituted for $\perp_1$. (Thus, an attribute domain is the union $\mathsf{Val} \cup \mathsf{Null}$ where Val is the set of values for the attribute specified in the metamodel, and Null is a countable set of *labeled nulls*.) Attribute $b@r1$ is certain but optional: if it exists, its value is known and given, but the existence is not guaranteed. Object $r2$ is optional and has two mandatory but uncertain attributes. Model A' (the leftmost) is an isomorphic copy of A: its OIDs and labeled nulls are in bijective correspondence with OIDs and nulls of model A, whereas the real value 3 is kept. In practice, models are normally considered up to their isomorphism, and below we will freely rename OIDs and nulls without mentioning.

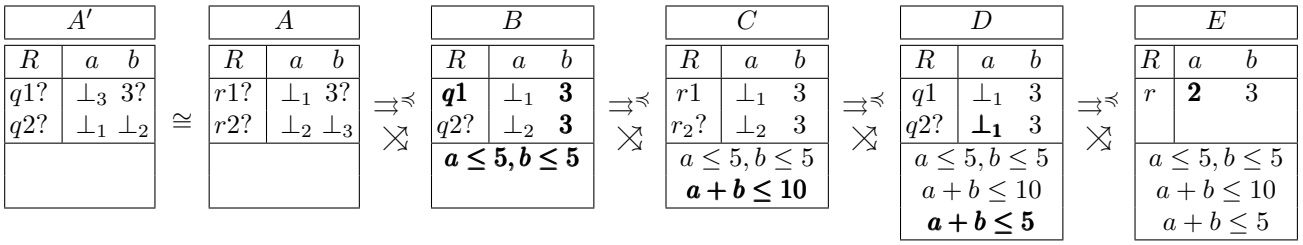

Figure 6: Model completions vs. model updates

Models B can be seen as a completion of A, in which the existence of object $r1$ and its attribute b became certain, and null $\perp_3@r2$ was substituted by value 3 (ignore the empty bottom compartment of the figure for a moment). However, the above understanding of the update silently assumes that object ri was updated to object qi for $i = 1, 2$. This update delta is denoted by the two-parallel-arrows symbol with superscript $\preccurlyeq$. However, it is also possible to consider an update from A to B as updating $r1$ to $q2$ and $r2$ to $q1$ (which is denoted by the crossed-arrows symbol), which changes the meaning of the update: now object $q2$ is just $r1$ with optional value 3 promoted to really existing in B, whereas object $q1$ is certainly existing $r2$, in which, additionally, the null value at the b-slot is completed with value 3. Note also that while the logical part of model A is empty (any positive integer can be substituted for nulls), model B has constraints that lessens the set of possible completions. Hence, both deltas are $\preccurlyeq$-arrows indeed (but the superindex $\preccurlyeq$ near the crossed-arrows delta is omitted). Thus, we have two different completion arrows between the same models, and the provenance of objects and values in model B is different for these deltas.

In the rest of the figure we will continue to consider two deltas for the same pair of models: the *parallel* delta (which maps objects in parallel: upper to upper and lower to lower), and the *crossed* one (which swaps the object positions).

The parallel delta from B to C does nothing with the structure, but there is a change in the logical part: formula $a + b \leq 10$ was added to the model. Syntactically, this change is an update, moreover, a completion as we added a new restriction to possible completions. However, as the new constraint can be inferred from the previous constraints, nothing was changed semantically. That is, although models B and C are syntactically different (and the parallel update is not an isomorphisms), semantically they are same, i.e., they have the same set of completions. Later we will formally explain such cases.

Now consider the cross delta from B to C. It specifies a general update rather than completion, because it changes a mandatory object $q1$ to an optional one, thus *increasing* uncertainty of the model. On the other hand, the part of uncertainty related to object $q2@B$ is decreased as this object became certain in C. Thus, the sets of full completions of B and C intersect via composing C-completions with the delta, but neither is a subset of the other. That is, B-completions in which object $q2$ is deleted are not achievable from C, and C-completions in which $r2$ is deleted are not achievable from B (later we will consider in detail how C-completions are mapped to B-completions via precomposition with deltas).

The parallel delta from C to D shows the only change in the structure: null $\perp_2@r2$ was replaced by null $\perp_1@q2$, but this change is *not* a trivial null renaming because null $\perp_1$ was already used in model C for attribute $a@r1$ and is kept in model D. In fact, having the same null in different slots is equivalent to declaring an equality constraint $a@q1 = a@q2$ for model D, which reduces the set of possible completions. The new constraint $a+b \leq 5$ added to D also reduces the set of completions, and hence the parallel delta from C to D is a legal $\preccurlyeq$-arrow. Indeed, any completion of D can be transformed to a completion of C by precomposition with the delta, while model C evidently has more completions than D.

The meaning of the crossed delta from C to D is also clear, but it is again not a completion as it changed the status of object $r1$ from mandatory in C to optional in D, and thus increased uncertainty of the model. On the other hand, the cross delta decreases uncertainty by (a) resolving optionality of $r2$, and (b) equalizing two nulls. Hence, again, the sets of full completions of C and D "intersect" but neither is a subset of the other if the relation between C and D is specified by a cross delta.

Finally, model E is fully certain as the only allowed completions are renaming of the OID. The parallel delta can be seen as a full completion that substitutes value 2 for $\perp_1@q1$ and deletes $q2$. In contrast, the cross delta deletes non-optional object $r1$ (and hence is a general update rather than a completion), resolves optionality of $q2$ and substitutes 2 for $\perp_1@q2$. We can also consider yet another delta that glues objects $q1$ and $q2$ into one

object $r@E$. This delta is a completion as it decreases uncertainty and any completion of E (only renaming of r are allowed) is mapped to a completion of D.

Examples above show that the comparative certainty of two models can only be considered a binary relation if there is a predefined way to relate models and compute the delta between them based on some information contained in the models, e.g., using some unique keys. If we do not have such keys (which is quite possible in MDE practice), certainty increasing or decreasing or non-comparability is a property of *deltas* relating the two models rather than pairs of models, and this is exactly the idea of u-spaces—designating a subcategory of arrows considered as certainty non-decreasing.

Definition 4 (Pre-well-behaved u-spaces) An u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ is called *pre-well-behaved (pre-wb)* if it satisfies the four laws on the right. The last one says that any model can be fully completed, i.e., seen as knowledge, is a consistent theory.

(iso)	$\boldsymbol{M}_{\mathsf{iso}} \subseteq_\bullet \boldsymbol{M}_{\preccurlyeq}$
(preord)	$c\colon A \to^{\preccurlyeq} B$ and $c'\colon A \overset{\succcurlyeq}{\leftarrow} B$ imply $c \in \boldsymbol{M}_{\mathsf{iso}}$ and $c' \in \boldsymbol{M}_{\mathsf{iso}}$
(epi)	$\boldsymbol{M}_{\preccurlyeq} \subseteq_\bullet \boldsymbol{M}_{\mathsf{epi}}$
(cons)	for any A there is $c\colon A \to^{\preccurlyeq} \dashv$

Given a model A, let $\mathsf{FC}^\bullet_{\preccurlyeq}(A) = \{\!\{c^\bullet\colon c \in \mathsf{FC}_{\preccurlyeq}(A)\}\!\}$ be the respective family of complete models (we recall that double figure brackets denote a family, i.e., a function from an indexing set ($\mathsf{FC}_{\preccurlyeq}(A)$ in our case) to the carrier set ($\boldsymbol{M}^\bullet$). The corresponding member set $(\mathsf{FC}^\bullet_{\preccurlyeq}A)^\#$ is denoted by $\mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A) = \{c^\bullet\colon c \in \mathsf{FC}_{\preccurlyeq}(A)\}$. If A is complete (fully certain), then $\mathsf{FC}^\bullet_{\preccurlyeq}(A)$ consists of all isomorphic copies of A (perhaps repeated in the family).

Lemma 1 (FC-properties) Any pre-wb u-space has the following properties¿
(i) *iso-closure:* $B^{\cong} \subset \mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A)$ for any $A \in \boldsymbol{M}^\bullet$ and $B \in \mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A)$.
(ii) *iso-ignorance:* $A \cong B$ implies $\mathsf{FC}^\bullet_{\preccurlyeq}(A) \cong \mathsf{FC}^\bullet_{\preccurlyeq}(B)$ and $\mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A) = \mathsf{FC}^{\bullet\#}_{\preccurlyeq}(B)$.
iii) *reindexing:* Any $c\colon A \to^{\preccurlyeq} B$ gives rise to injection $c^*\colon \mathsf{FC}_{\preccurlyeq}(A) \hookleftarrow \mathsf{FC}_{\preccurlyeq}(B)$

Proof. (i) and (ii) are direct consequences of the definition. For (iii), we use pre-composition (see Sect. 4.1) that lifts any $\preccurlyeq$-arrow $c\colon A \to^{\preccurlyeq} A'$ to a function $c^*\colon \mathsf{FC}_{\preccurlyeq}(A) \leftarrow \mathsf{FC}_{\preccurlyeq}(A')$. The latter is injection due to (epi). □

Thus, $\preccurlyeq$-arrows indeed do not decrease certainty. However, even non-isomorphic completion can be non-increasing either, e.g., recall the parallel delta from B to C in Ex. 3.

Definition 5 (semantic equivalence) A $\preccurlyeq$-arrow $c\colon A \to^{\preccurlyeq} B$ is a *semantic equivalence*, if function $c^*\colon \mathsf{FC}_{\preccurlyeq}(A) \hookleftarrow \mathsf{FC}_{\preccurlyeq}(B)$ is bijective; then we write $c\colon A \to^{\approx} B$.

4.3 Logic for uncertainty

In the database literature, class $\mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A)$ is usually called the *semantics* of incomplete model A and often denoted by $[\![\, A \,]\!]$. As the example above shows, the class of completion arrows $\mathsf{FC}_{\preccurlyeq}(A)$ and the corresponding family $\mathsf{FC}^\bullet_{\preccurlyeq}(A)$ are more important and actually necessary for an accurate formal semantics for updates (cf. the discussion of deltas in [DXC11]). Libkin's definition of the informational preorder, $A \preceq A'$ iff $[\![\, A' \,]\!] \subseteq [\![\, A \,]\!]$, is subsumed by reindexing specified in Lemma 1(iii). In [Dis16], the relation between *fair database domain* (introduced by Libkin as the universe supporting naive query evaluation on incomplete databases) and u-spaces is discussed in more detail, and several results are proven.

The two major operations of the classical model theory are Mod and Th derived from satisfiability relation $\models$ between models and formulas both built over some given signature Σ of operation and predicate symbols. Operator Mod maps a set of formulas, or a theory, Φ, to its set of models $\mathsf{Mod}\Phi = \{X \in \mathsf{Mod}(\Sigma)\colon X \models \phi \text{ for all } \phi \in \Phi\}$. Operator Th maps a class of models $\mathcal{X}$ to its theory $\mathsf{Th}\mathcal{X} = \{\phi \in \Phi(\Sigma)\colon X \models \phi \text{ for all } X \in \mathcal{X}\}$. Operator $\mathsf{FC}_{\preccurlyeq}$ is a $\preccurlyeq$-arrow counterpart of Mod, but we also need a $\preccurlyeq$-arrow counterpart of Th. Uncertain models are themselves theories, and satisfiability between a model A and a fully certain model C can be defined via the existence of $\preccurlyeq$-arrow from A to C, i.e., $C \models A$ iff there is $c\colon A \to^{\preccurlyeq} C$. Then given a set of certain models $\mathcal{C} \subset \boldsymbol{M}^\bullet_{\dashv}$, its theory, i.e., model $\mathsf{Th}\,\mathcal{C}$, should be uncertain enough to encode all models in $\mathcal{C}$ and provide the inclusion $\mathcal{C} \subset \mathsf{FC}_{\preccurlyeq}(\mathsf{Th}\,\mathcal{C})$.[3] However, having in $\mathsf{Th}\,\mathcal{C}$ enough uncertainty to encode $\mathcal{C}$, we would not like to have it more than needed, and we require model $\mathsf{Th}\mathcal{C}$ to be maximally certain amongst all models encoding $\mathcal{C}$. In other words, for any model X with $\mathcal{C} \subset \mathsf{FC}_{\preccurlyeq}X$, there should be a uniquely defined completion arrow $X!\colon X \to^{\preccurlyeq} \mathsf{Th}\,\mathcal{C}$. Categorically, the last statement means that $\mathsf{Th}\,\mathcal{C}$ is nothing but the product of class $\mathcal{C}$, so that we can define $\mathsf{Th}\,\mathcal{C}$ as $\Pi^\bullet\mathcal{C}$. It

[3] The exact equality rather than subsetting is desirable, but in general is not achievable as semantics is usually richer than logic, and we normally have $\mathcal{C} \subset \mathsf{Mod}\,\mathsf{Th}\,\mathcal{C}$. Equality only holds for special classes of models called *closed*.

is precisely the categorical reformulation of Libkin's consideration of theories as greatest lower bounds in the information pre-order [Lib15].

Definition 6 (well-behaved u-spaces: products) A pre-wb u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ is called *well-behaved (wb)* if category $\boldsymbol{M}_{\preccurlyeq}$ has products (denoted by $\Pi_{\preccurlyeq}$). Products provide that for any model A, there is a model ${}^{\bullet}\Pi_{\preccurlyeq}\mathsf{FC}^{\bullet}_{\preccurlyeq}(A)$ (understood as $\mathsf{ThMod}A$) and denoted by $A^{\models}$. Dually, for any family of certain models $\mathbf{C} = \{\!\{C_i\colon i{\in}I_{\mathbf{C}}\}\!\}$, there is a family $\mathsf{FC}^{\bullet}_{\preccurlyeq}({}^{\bullet}\Pi_{\preccurlyeq}\mathbf{C})$ (understood as $\mathsf{ModTh}\mathbf{C}$ and) denoted by $\mathbf{C}^{\Vdash}$. □

The following lemma (a straightforward consequence of the universality of products) shows that we indeed have a version of the abstract model theory.

Lemma 2 (Galois correspondence for models)

$\models$-closure operator	Law name	$\Vdash$-closure operator
for any $A{\in}\boldsymbol{M}^{\bullet}$ there is unique $A!\colon A \to^{\approx} A^{\models}$	(unit)	for any $\mathbf{C}{\in}\mathbf{Fam}(\boldsymbol{M}^{\bullet}_{\dashv})$ there is unique $\mathbf{C}!\colon \mathbf{C} \hookrightarrow \mathbf{C}^{\Vdash}$
$\mathsf{FC}_{\preccurlyeq}(\Pi_{\preccurlyeq}\mathsf{FC}^{\bullet}_{\preccurlyeq}(A)) = \mathsf{FC}_{\preccurlyeq}(A)$	(idemp)	$\Pi_{\preccurlyeq}\mathsf{FC}^{\bullet}_{\preccurlyeq}({}^{\bullet}\Pi_{\preccurlyeq}\mathbf{C}) \cong \Pi_{\preccurlyeq}\mathbf{C}$
$c\colon A \to^{\preccurlyeq} B$ gives rise to $c^{\models}\colon A^{\models} \to^{\preccurlyeq} B^{\models}$	(monot)	$f\colon \mathbf{C} \to \mathbf{D}$ gives rise to $f^{\Vdash}\colon \mathbf{C}^{\Vdash} \to \mathbf{D}^{\Vdash}$

Definition 7 (closed and equivalent objects) Model A is *closed* if $A!\colon A \to^{\approx} A^{\models}$ is an isomorphism in $\boldsymbol{M}$, i.e., A and $A^{\models}$ are isomorphic via $A!$. Two models are *semantically equivalent*, $A_1 \approx A_2$, if $A_1^{\models} = A_2^{\models}$.

A family $\mathbf{C}$ is called *closed* if $\mathbf{C}!$ is a bijection, i.e., families $\mathbf{C}$ and $\mathbf{C}^{\Vdash}$ only differ in the choice of the indexing set but otherwise are the same.

Lemma 3 (certain objects and semantic closure) (i) If model A is certain, then $A!\colon A \to^{\approx} A^{\models}$ is an isomorphism. (ii) If $A_1{\approx}A_2$ and both updates are certain, then $A_1{\cong}A_2$. (iii) If $A_1{\approx}A_2$ and A_1 is certain, then A_2 is certain as well.

4.4 Model spaces with uncertainty, II: uncertain updates and their completions

Similarly to how model-based (a.k.a. state-based) updating can be ambiguous without deltas [DXC11], our examples above show that model completions are also ambiguous without deltas. But the latter can themselves be uncertain. (For example, if deltas are spans, their heads are ordinary models that can be uncertain.) Hence, our next goal is to extend the notion of u-space with uncertain updates.

For an u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$, the triangle functor $\boldsymbol{T}_{\boldsymbol{M}_{\preccurlyeq}}$ generated by subcategory $\boldsymbol{M}_{\preccurlyeq}$ (Sect. 4.1.3) will be denoted by $\boldsymbol{T}_{\preccurlyeq}$.

Definition 8 (uu-spaces) A *model space with uncertain models and uncertain updates*, or *uu-space*, is a triple $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leqslant})$ such that the pair $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ is an u-space, and $\boldsymbol{T}_{\leqslant}$ is an object-full subfunctor of $\boldsymbol{T}_{\preccurlyeq}$ (i.e., $\boldsymbol{T}_{\leqslant} \subseteq_{\bullet} \boldsymbol{T}_{\preccurlyeq}$ see Sect. 4.1) such that for any $A{\in}\boldsymbol{M}^{\bullet}$, the pair $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leqslant}A)$ is an u-space too. Moreover, reindexing $\boldsymbol{T}_f\colon \boldsymbol{T}_{\preccurlyeq}A \leftarrow \boldsymbol{T}_{\preccurlyeq}B$ along any update $f{\in}\boldsymbol{M}(A,B)$ is an u-space morphism.

In detail, for any object $A{\in}\boldsymbol{M}^{\bullet}$, we have a category $\boldsymbol{T}_{\leqslant}A$, whose objects are updates, $\boldsymbol{T}_{\leqslant}{}^{\bullet}A = \boldsymbol{M}(A,*)$, and morphisms are some of triangles based on $\preccurlyeq$-arrows (see Sect. 4.1.3), i.e., any $\boldsymbol{T}_{\leqslant}A$ morphism is a triangle (A, u, u', b) whose base b is an $\boldsymbol{M}_{\preccurlyeq}$-arrow (but, in general, not all such triangles are $\boldsymbol{T}_{\leqslant}A$-arrows). We call $\boldsymbol{T}_{\leqslant}A$ morphisms *update completions*, or *certainty non-decreasing triangles*, or just $\leqslant$-arrows, and denote them by a double arrow $\tau_c^A\colon u \Rightarrow^{\leqslant} u'$ with the base arrows denoted by c rather than b to recall that this arrow is a model completion (but the sides of the triangle are general updates, i.e., $\boldsymbol{M}$-arrows). Normally, the superscript A will be omitted. Thus, $\boldsymbol{T}_{\leqslant}A \subseteq_{\bullet} \boldsymbol{T}_{\preccurlyeq}A \subseteq_{\bullet} \boldsymbol{T}A$ for all $A{\in}\boldsymbol{M}^{\bullet}$.

Note that all three functors share the same reindexing along $\boldsymbol{M}$-arrows via precomposition. Requiring reindexing to be an u-space morphism means that for any update $f\colon A \leftarrow B$ and update completion (A, u, u', c), the triangle $(B,\ f;u,\ f;u',\ c)$ is an update completion as well. In other words, an arrow $\tau_c^A\colon u \Rightarrow^{\leqslant} u'$ in category $\boldsymbol{T}_{\leqslant}A$ is mapped to arrow $\tau_c^B\colon f;u \Rightarrow^{\leqslant} f;u'$ in $\boldsymbol{T}_{\leqslant}B$.

Definition 9 (well-behaved uu-spaces) Uu-space is called *well-behaved (wb)* if u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ and all u-spaces $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leqslant}A)$, $A{\in}\boldsymbol{M}^{\bullet}$ are wb, and, particularly, have products. We recall that products in category $\boldsymbol{M}_{\preccurlyeq}$ are denoted by $\Pi_{\preccurlyeq}$, and products in categories $\boldsymbol{T}_{\leqslant}A$ will be denoted by $\Pi_A{}_{\leqslant}$ with index A often omitted. In addition, all products $\Pi_{\leqslant}$ are required to be idempotent, and conditions in the inset table on the right must hold. □

(Comm)	$\boldsymbol{T}_{[=]} \cap \boldsymbol{T}_{\preccurlyeq} \subset \boldsymbol{T}_{\leqslant}$
(Id)	if $\tau_c^A\colon \mathsf{id}_A \Rightarrow^{\leqslant} *$ then $(\tau_c^A)^{\bullet} = c$
(Iso)	$(\boldsymbol{T}_{\leqslant}A)_{\mathsf{iso}} = \boldsymbol{T}_{\mathsf{iso}}A \cap \boldsymbol{T}_{[=]}A$
(CertCert)	$\boldsymbol{M}_{\mapsto} \subset \boldsymbol{M}$.
($\mathsf{FC}_{\leqslant}$ vs. $\preccurlyeq$)	if $u{\in}\boldsymbol{M}_{\preccurlyeq}$ then $\mathsf{FC}^{\bullet}_{\leqslant}(u) \subset \boldsymbol{M}_{\preccurlyeq}$
($\Pi_{\leqslant}$ vs. $\preccurlyeq$)	if $\mathcal{U} \subset \boldsymbol{T}^{\bullet}_{\dashv} \cap \boldsymbol{M}_{\preccurlyeq}$ then ${}^{\bullet}\Pi_{\leqslant}\mathcal{U} \in \boldsymbol{M}_{\preccurlyeq}$.

These conditions are explained below.

(Comm) We require any commutative triangle based on a $\preccurlyeq$-arrow (i.e., a quadruple $\tau_c = (A, u, u', c)$ with $c\colon u^\bullet \to^{\preccurlyeq} u'^\bullet$ and $u' = u\,;c$) to be a $\leqslant$-arrow $\tau_c\colon u \Rightarrow^{\leqslant} u'$. Indeed, commutativity means that update u' is, essentially, u, but its target is a completion of u's target. However, although any commutative triangle is a completion, there can be non-commutative completion triangles as we have seen in Fig. 5. Hence, the formulation of the (Comm) law as above.

(Id) Given model $A{\in}\boldsymbol{M}^\bullet$, completions of the identity update are nothing but completions of A (and all completion triangles are commutative).

(Iso) To define full update completions, i.e., to apply the general u-space Def. 2 to updates-as-objects, we need to discuss update isomorphisms in categories $\boldsymbol{T}_{\leqslant}A$. It is easy to see that any commutative triangle $(A, u, u; i, i)$ based on isomorphism $i\colon u^\bullet \to^{\cong} u'^\bullet$ is an isomorphism in $\boldsymbol{T}_{\leqslant}A$. We require the converse to be true as well, which provides the law (Iso) (in which $\boldsymbol{T}_{\mathsf{iso}}$ denotes $\boldsymbol{T}_{\boldsymbol{M}_{\mathsf{iso}}}$).

Now we repeat the general definition for updates as objects. If the only $\leqslant$-triangles from update $u\colon A \to *$ are its isomorphisms, $\boldsymbol{T}_{\leqslant}A(u, *) = \boldsymbol{T}_{\leqslant_{\mathsf{iso}}}A(u, *)$, then u is called *(fully) certain* or *complete*, and let $\boldsymbol{T}^\bullet_{\dashv}A$ denotes the class of all such in the category $\boldsymbol{T}_{\leqslant}A$.. We write $\tau\colon u \Rightarrow^{\leqslant} \dashv$ to say that $\tau^\bullet \in \boldsymbol{T}^\bullet_{\dashv}A$, and call τ a *full completion* of u.

Let $\mathsf{FC}_{A\leqslant}(u)$ or just $\mathsf{FC}_{\leqslant}(u)$ denote the class of all such, and $\mathsf{FC}^\bullet_{\leqslant}(u) \stackrel{\text{def}}{=} \{\tau^\bullet\colon \tau{\in}\mathsf{FC}_{\leqslant}(u)\}$ is the respective set of complete updates (note a distinction from $\mathsf{FC}^\bullet_{\preccurlyeq}$, which was a family rather than a set). Thus, update u is complete iff $\mathsf{FC}^\bullet_{\leqslant}(u)$ contains all isomorphic copies of u and nothing else. It is easy to see that if u is complete, then $u^\bullet$ is a complete model (otherwise, composition $u;c$ for a non-trivial completion $c\colon A \to *$ would be a non-trivial completion of u). However, our examples in section 3 show that there can be non-complete updates, whose target is a complete model.

(CertCert) Let $\boldsymbol{M}_{\mapsto} \subset \boldsymbol{M}$ denote the class of all (fully) certain models and certain updates between them: $\boldsymbol{M}^\bullet_{\mapsto} = \boldsymbol{M}^\bullet_{\dashv}$, and $\boldsymbol{M}_{\mapsto}(A, *) = \boldsymbol{T}^\bullet_{\dashv}(A, *)$ for $A{\in}\boldsymbol{M}^\bullet_{\mapsto}$. Law (CertCert) states that composition of two certain updates is certain, i.e., the universe of certain models and updates is a subcategory, $\boldsymbol{M}_{\mapsto} \subset \boldsymbol{M}$.

($\leqslant$ vs. $\preccurlyeq$) Finally, there are two laws regulating interaction between $\preccurlyeq$- and $\leqslant$-arrows. The first says that update completions of a model completion are themselves model completions. The second says that the update encoding a set of model completions is itself a model completion. □

As categories $\boldsymbol{T}_{\leqslant}A$ have idempotent products, we can introduce semantics closure $u^{\models}$ of an update $u\colon A \to *$, and syntactic closure $\mathcal{U}^{\Vdash}$ of a class $\mathcal{U}$ of certain updates of A. The following is a rewrite of general u-space constructions on page 12 for u-spaces $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leqslant}A)$, $A{\in}\boldsymbol{M}^\bullet$.

Lemma 4 ($\mathsf{FC}_{\leqslant}$-properties) Any wb uu-space has the following properties:
(i) *iso-closure:* $u'^{\cong} \subset \mathsf{FC}^\bullet_{\leqslant}(u)$ for any $u\colon A \to *$ and $u'{\in}\mathsf{FC}^\bullet_{\leqslant}(u)$.
(ii) *iso-ignorance:* $u \cong v$ implies $\mathsf{FC}^\bullet_{\leqslant}(u) = \mathsf{FC}^\bullet_{\leqslant}(v)$.
(iii) *reindexing:* Any $\tau_c\colon u \Rightarrow^{\leqslant} u'$ gives rise to injection $\tau_c^*\colon \mathsf{FC}_{\leqslant}(u) \hookleftarrow \mathsf{FC}_{\leqslant}(u')$
(iv) *id-certainty:* If model A is certain, then update id_A is certain as well (in the u-space $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leqslant}A)$, $A{\in}\boldsymbol{M}^\bullet$).

Proof. (i) and (ii) are direct consequences of the definition. For (iii), we use pre-composition (see Sect. 4.1) that lifts any $\leqslant$-arrow $\tau_c\colon A \to^{\preccurlyeq} A'$ to a function $\tau_c^*\colon \mathsf{FC}_{\leqslant}(A) \leftarrow \mathsf{FC}_{\leqslant}(A')$. The latter is injection due to law (epi). □

Definition 10 (semantic equivalence for updates) A completion triangle $\tau_c\colon u \Rightarrow^{\leqslant} v$ is a *semantic equivalence*, if function $\tau_c^*\colon \mathsf{FC}_{\leqslant}(u) \hookleftarrow \mathsf{FC}_{\leqslant}(v)$ is bijective; then we write $\tau_c\colon u \Rightarrow^{\approx} v$.

Lemma 5 (Galois correspondence for updates)

$\models$-closure operator	Law name	$\Vdash$-closure operator
for any $u{\in}\boldsymbol{M}$ there is unique $u!\colon u \Rightarrow^{\approx} u^{\models}$	(unit)	$\mathcal{U} \subset \mathcal{U}^{\Vdash}$ for any $\mathcal{U} \subset \boldsymbol{T}^\bullet_{\dashv}$
$\mathsf{FC}_{\leqslant}({}^\bullet\Pi_{\leqslant}\mathsf{FC}^\bullet_{\leqslant}(u)) = \mathsf{FC}_{\leqslant}(u)$	(idemp)	$\Pi_{\leqslant}\mathsf{FC}^\bullet_{\leqslant}({}^\bullet\Pi_{\leqslant}\mathcal{U}) \cong \Pi_{\leqslant}\mathcal{U}$
$\tau_c\colon u \Rightarrow^{\leqslant} u'$ gives rise to $\tau_c^{\models}\colon u^{\models} \Rightarrow^{\leqslant} u'^{\models}$	(monot)	$\mathcal{U} \subset \mathcal{U}'$ implies $\mathcal{U}^{\Vdash} \subset \mathcal{U}'^{\Vdash}$

Definition 11 (closed and equivalent objects) Update u is called *closed* if $u!\colon u \Rightarrow^{\approx} u^{\models}$ is an isomorphism in $\boldsymbol{T}_{\leqslant}A$, i.e., u and $u^{\models}$ are isomorphic via $u!$. Two update are *semantically equivalent*, $u_1 \approx u_2$, if $u_1^{\models} = u_2^{\models}$. A class $\mathcal{U}$ of updates is *closed* if $\mathcal{U}!$ is a bijection. Two classes are *logically equivalent*, $\mathcal{U}_1 \approx \mathcal{U}_2$, if $\mathcal{U}_1^{\Vdash} = \mathcal{U}_2^{\Vdash}$.

Lemma 6 (certain objects and semantic closure) (i) If update u is certain, then $u!\colon u \to^{\approx} u^{\models}$ is an isomorphism. (ii) If $u_1 \approx u_2$ and both updates are certain, then $u_1 \cong u_2$. (iii) If $u_1 \approx u_2$ and u_1 is certain, then u_2 is certain as well.

Morphisms of uu-spaces are nothing but view functors compatible with uu-structure

Definition 12 (uu-space morphisms or views) Let $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leqslant})$ and $\mathcal{N} = (\boldsymbol{N}, \boldsymbol{N}_{\preccurlyeq}, \boldsymbol{U}_{\leqslant})$ be two wb uu-spaces (i.e., $\boldsymbol{U}_{\leqslant}$ is an object-full subfunctor of the triangle functor $\boldsymbol{U}_{\preccurlyeq}$ generated by subcategory $\boldsymbol{N}_{\preccurlyeq}$as explained in Sect. 4.1.3). A *well-behaved (wb) view* $\mathcal{M} \to \mathcal{N}$ is a surjective on objects and full functor $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ compatible with the uu-structure in the following way.

a) Model completions are preserved, i.e., the restriction of get to $\boldsymbol{M}_{\preccurlyeq}$ is a functor $\mathsf{get}_{\preccurlyeq}\colon \boldsymbol{M}_{\preccurlyeq} \to \boldsymbol{N}_{\preccurlyeq}$.

a') Moreover, we require $\mathsf{get}_{\preccurlyeq}$ to be full.

b) Update completions are preserved, i.e., for any $A \in \boldsymbol{M}^{\bullet}$, the restriction of triangle functor $\mathsf{get}^{\boldsymbol{T}}_A$ (see Sect. 4.1.3) to $\boldsymbol{T}_{\leqslant}A$ is a functor $\mathsf{get}_{A\leqslant}\colon \boldsymbol{T}_{\leqslant}A \to \boldsymbol{U}_{\leqslant}B$ (where B stands for $\mathsf{get}A$).

b') Moreover, we require all functors $\mathsf{get}_{A\leqslant}$ to be full.

c) Product preservation: Functor get preserves $\preccurlyeq$-products, and all functors $\mathsf{get}_{A\leqslant}$ (for $A \in \boldsymbol{M}^{\bullet}$) preserve $\leqslant$-products.

We define *uu-space morphisms* to be wb views, and denote them by arrows $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$. □

In the lens framework, functor get is normally considered together with put, and the latter provides fullness conditions. However, views are themselves interesting (e.g., for databases), and we would like to state the next result independently of put.

Lemma 7 (wb view properties) *(i) A wb view* $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$ *maps certain models and certain updates in space* $\mathcal{M}$ *to certain models and certain updates in* $\mathcal{N}$*. Thus, there is a functor* $\mathsf{get}_{\vdash\!\dashv}\colon \boldsymbol{M}_{\vdash\!\dashv} \to \boldsymbol{N}_{\vdash\!\dashv}$.

(ii) For any model $A \in \boldsymbol{M}^{\bullet}$ *and update* $u \in \mathsf{FC}^{A^{\bullet}}_{\leqslant}(\mathsf{id}_A)$*, there is* $\mathsf{get}u \cong \mathsf{id}_{\mathsf{get}A}$

(iii) For any update $u\colon A \to *$*, equation* $\mathsf{get}_A(u^{\models}) \cong (\mathsf{get}_A u)^{\models}$ *holds.*

4.5 Lenses over uu-spaces

First, we recall the notion of an ordinary (delta) lens.

Definition 13 (delta lenses) Let $\boldsymbol{M}$, $\boldsymbol{N}$ be two categories considered as model spaces.

(i) A *lens* from $\boldsymbol{M}$ to $\boldsymbol{N}$ is a pair $\ell = (\mathsf{get}, \mathsf{put})$ with $\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ a functor, and $\mathsf{put} = \{\!\{\mathsf{put}_A\colon A \in \boldsymbol{M}^{\bullet}\}\!\}$ a family of functions $\mathsf{put}_A\colon \boldsymbol{M}(A, *) \leftarrow \boldsymbol{N}(\mathsf{get}A, *)$ indexed by $\boldsymbol{M}$-objects. We will often write $\mathsf{put}_A v$ for $\mathsf{put}_A(v)$, and $\mathsf{put}^{\bullet}_A v$ for $(\mathsf{put}_A v)^{\bullet}$.

(ii) A lens is called *well-behaved (wb)* if the following two conditions (laws) are satisfied for all $A \in \boldsymbol{M}^{\bullet}$ (below B stands for $\mathsf{get}A$):

Identity preservation: $\mathsf{put}_A(\mathsf{id}_B) = \mathsf{id}_A$.

PutGet law: $\mathsf{get}(\mathsf{put}_A v) = v$ for all updates $v\colon B \to *$.

Note that the definition does not require put_A to map a triangle of updates on the view side, $v\colon B \to B'$, $v'\colon B' \to B''$, and $v''\colon B \to B''$, to a triangle on the source side, i.e., the case when $\mathsf{put}^{\bullet}_A v'' \neq \mathsf{put}^{\bullet}_{A'} v'$ with A' standing for $\mathsf{put}^{\bullet}_A v$ is not prohibited (and actually appears in practice, see [DXC11] for an example).

(iii) A wb lens is called *very well-behaved (vwb)* if the put-image of any commutative triangle of view updates is mapped to a commutative triangle on the source side.

PutPut law: $\mathsf{put}_A(v; v') = (\mathsf{put}_A v); (\mathsf{put}_{A'} v')$ where $A' = \mathsf{put}^{\bullet}_A v$. (Note that mapping of an arbitrary triangle to a triangle is still not required, but *commutative* triangles are mapped to triangles (which are also commutative).

The definition above is equivalent to the delta lens definition in [DXC11] with a minor distinction that in [DXC11], functoriality of get is attributed to the very wb rather than wb lenses. □

The ordinary lenses are based on very strict update (propagation) policies: for a given view update $v\colon \mathsf{get}A \to *$, in the entire set $\mathsf{get}^{-1}(v) \subset \boldsymbol{M}(A, *)$ of updates satisfying the Putget law, only one is chosen. The following two constructs relax this requirement. The first one, *update policies*, do it in a direct semantic way with the notion of a multi-valued operation. The second one, *uu-lenses*, do it indirectly by encoding a multitude of models by a suitable uncertain model.

Definition 14 (update policies) Let $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leqslant})$ and $\mathcal{N} = (\boldsymbol{N}, \boldsymbol{N}_{\preccurlyeq}, \boldsymbol{U}_{\leqslant})$ be two uu-spaces (i.e., spaces of uncertain models and uncertain updates with $\boldsymbol{U}$ the triangle functor for $\boldsymbol{N}$), and $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$ be a wb view as described in Def. 12.

(i) A *(multi-valued) update policy* for get is a family Put of set-valued functions $\{\!\{\mathsf{Put}_A\colon 2^{\boldsymbol{T}^{\bullet}_{\lhd A}} \leftarrow \boldsymbol{U}^{\bullet}_{\lhd B}\colon\ A \in \boldsymbol{M}^{\bullet}, B = \mathsf{get}A\}\!\}$ indexed by $\boldsymbol{M}$-objects (note that both the input and the output deal with certain updates). We will often write $\mathsf{Put}_A v$ for $\mathsf{Put}_A(v)$, and $\mathsf{Put}^{\bullet}_A v$ for $\{u^{\bullet}\colon u{\in}\mathsf{Put}_A v\}$.

(ii) A policy is called *well-behaved (wb)* if the following conditions are satisfied for all $A{\in}\boldsymbol{M}^{\bullet}$ and $v{\in}\boldsymbol{N}(B,*)$ (where B stands for $\mathsf{get}A$. and update isomorphisms are considered in the respective triangle categories $\boldsymbol{U}_{\leqslant}B$ and $\boldsymbol{T}_{\leqslant}A$):

o) *PutGet law (up to iso)*: $\mathsf{get}u \cong v$ for any $u \in \mathsf{Put}_A v$. In other words, $\mathsf{Put}_A(v) \subset \mathsf{get}^{-1}(v^{\cong})$.

a) $\preccurlyeq$*-preservation*: if $v{\in}\boldsymbol{N}_{\preccurlyeq}$, then $\mathsf{Put}_A v \subset \boldsymbol{M}_{\preccurlyeq}$

b1) *Iso-closedness*: if $u{\in}\mathsf{Put}_A v$, then $u^{\cong} \subset \mathsf{Put}_A v$

b2) *Iso-ignorance*: If $v \cong v'$, then $\mathsf{Put}_A v = \mathsf{Put}_A v'$.

c) *Identity*: if $v = \mathsf{id}_B$, then $\mathsf{Put}_A v = \mathsf{FC}^{\bullet}_{\leqslant}(\mathsf{id}_A)$ (Note that Lemma 7(ii) ensures that the specific policy c) for identities satisfies the general Putget law o).)

(iii) Given two policies for the same view get, Put and Put', we say that Put is *not more certain* than Put' and write $\mathsf{Put} \sqsubseteq \mathsf{Put}'$, if $\mathsf{Put}_A v \supseteq \mathsf{Put}'_A v$ for all A and all $v\colon \mathsf{get}A \to *$. This gives us a posetal category $\textsc{Pol}_{\mathsf{get}}$.

Definition 15 (uu-lenses) Let $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leqslant})$ and $\mathcal{N} = (\boldsymbol{N}, \boldsymbol{N}_{\preccurlyeq}, \boldsymbol{U}_{\leqslant})$ be two uu-spaces as above.

(i) An *(asymmetric) uu-lens* (read *a lens with uncertainty*) from $\mathcal{M}$ to $\mathcal{N}$ is a delta lens $\ell = (\mathsf{get}, \mathsf{put})\colon \boldsymbol{M} \rightleftharpoons \boldsymbol{N}$ between the carrier categories as defined above in Def. 13(i).

(ii) An uu-lens is called *(almost) well-behaved (wb)* if it is compatible with the uu-structure as follows.

(uu-get) Functor get is a wb view (uu-space morphism) as defined in Def. 12

(uu-put) Family put must satisfy the following set of conditions to hold for any $A{\in}\boldsymbol{M}^{\bullet}$ and $v\colon B \to *$ (B is $\mathsf{get}A$):

o) A version of Putget law called *uu-Putget*: $\mathsf{get}(\mathsf{put}_A v) = v^{\models}$ for any $v\colon B \to *$.

a) if $v{\in}\boldsymbol{N}_{\preccurlyeq}$ then $\mathsf{put}_A v{\in}\boldsymbol{M}_{\preccurlyeq}$, i.e., model completions are preserved;

b1) for any completion triangle on the view side, (B, v, v', c), its put-image $(A, \mathsf{put}_A v, \mathsf{put}_A v', \mathsf{put}_{A'} c)$ (where $A' = \mathsf{put}_A v^{\bullet}$) is a completion triangle on the source side.

In other words, we require an $\boldsymbol{U}_{\leqslant}B$-arrow $\tau^B_c\colon v \Rightarrow^{\leqslant} v'$ on the view side to be mapped to an $\boldsymbol{T}_{\leqslant}A$-arrow $\tau^A_{\mathsf{put}_{A'}c}\colon \mathsf{put}_A v \Rightarrow^{\leqslant} \mathsf{put}_A v'$ on the source side. This allows us to define a mapping $\mathsf{put}_{A\leqslant}\colon \boldsymbol{T}_{\leqslant}A \leftarrow \boldsymbol{U}_{\leqslant}B$, and we will often omit the subindex $\leqslant$ (but note that we do not require from put to map general triangles based on $\boldsymbol{U}_{\preccurlyeq}$-arrows on the $\mathcal{N}$-side to triangles on the $\mathcal{M}$-side).

b2) All mappings $\mathsf{put}_A\colon \boldsymbol{T}_{\leqslant}A \leftarrow \boldsymbol{U}_{\leqslant}B$ must preserve composition of update completions (i.e., tiling).

c) if $v = \mathsf{id}_B$, then $\mathsf{put}_A v = \mathsf{id}^{\models}_A$, i.e., identities are preserved up to semantic equivalence.

As for a general update v, v is not isomorphic to $v^{\models}$, we cannot say that uu-lens satisfying the laws above is absolutely well-behaved. However, as $v \cong v^{\models}$ for certain updates by Lemma 6(i), and id_A is certain for a certain model A by Lemma 4(iv), uu-Putget and identity preservation up to $\_^{\models}$ do provide isomorphism $\mathsf{get}(\mathsf{put}_A v) \cong v$ for certain v and $\mathsf{put}_A(\mathsf{id}_B) \cong \mathsf{id}_A$ for certain A. We thus may say that the lens satisfying the laws above is almost wb, and, as a rule, we will omit the adjective 'almost'.

We will denote an uu-lens by a double arrow $\ell\colon \mathcal{M} \rightleftharpoons \mathcal{N}$, and say lens $\ell = (\mathsf{get}, \mathsf{put})$ is defined *over* the view get.

Definition 16 (arrows between uu-lenses) Let $\ell = (\mathsf{get}, \mathsf{put})$ and $\ell' = (\mathsf{get}, \mathsf{put}')$ be two wb uu-lenses over the same view $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$. A *completion* arrow from ℓ to ℓ' is a family of product preserving natural transformations $\sigma_A\colon \mathsf{put}_A \Rightarrow \mathsf{put}'_A$ (read *tau-to-tau*) indexed by $A{\in}\boldsymbol{M}^{\bullet}$. That is, for any A and update $v\colon \mathsf{get}A \to *$, we have an update completion (i.e., a triangle) $\sigma_A(v)\colon \mathsf{put}_A v \Rightarrow^{\leqslant} \mathsf{put}'_A v$, and naturality means that for any completion $\tau\colon v \Rightarrow^{\leqslant} v'$, we have a commutative square: $\sigma_A(v)\,;\mathsf{put}'_A(\tau) = \mathsf{put}_A(\tau)\,;\sigma_A(v')$.

This gives us a category $\textsc{Lens}_{\mathsf{get}}$ of all wb u-lenses and their completion arrows over a given view $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$.

Theorem 8 (from lenses to policies and back) Let $\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$ be a wb view between wb uu-model spaces. Then

(i) Any wb uu-lens $\ell = (\mathsf{get}, \mathsf{put})\colon \mathcal{M} \rightleftharpoons \mathcal{N}$ over get gives rise to a wb update policy Put_{ℓ} for get. We will say that policy Put_{ℓ} is *implicit* in ℓ. Moreover, this correspondence extends to a functor $\boldsymbol{pol}_{\mathsf{get}}\colon \textsc{Lens}_{\mathsf{get}} \to \textsc{Pol}_{\mathsf{get}}$.

(ii) Any wb update policy Put for get gives rise to a wb uu-lens $\ell_{\mathsf{Put}} = (\mathsf{get}, \mathsf{put})$ over get. We will say that the lens ℓ_{Put} is *based* on Put. Moreover, this correspondence extends to a functor $\boldsymbol{lens}_{\mathsf{get}}\colon \textsc{Pol}_{\mathsf{get}} \to \textsc{Lens}_{\mathsf{get}}$.

(iii) Functors above are adjoint: $\boldsymbol{pol}_{\mathsf{get}} \dashv \boldsymbol{lens}_{\mathsf{get}}$.

The unit of the adjunction is given by the family of update completions $u!_{A,v}\colon u_{A,v} \Rightarrow^{\approx} u^{\models}_{A,v} = u^{**}_{A,v}$, where $u_{A,v} = \mathsf{put}_A v$ and $u^{**}_{A,v} = \mathsf{put}^{**}_A v$ with $\mathsf{put}^{**} = \boldsymbol{lens}_{\mathsf{get}}(\boldsymbol{pol}_{\mathsf{get}}(\mathsf{put}))$ being the lens based on the policy implicit in

$\ell = (\mathsf{get}, \mathsf{put}) \in \textsc{Lens}^{\bullet}_{\mathsf{get}}$. The counit of the adjunction is given by the family of inclusions $i_{A,v}\colon\ \mathcal{U}^{**}_{A,v} = \mathcal{U}^{\Vdash}_{A,v} \hookleftarrow \mathcal{U}_{A,v}$ where $\mathcal{U}_{A,v} = \mathsf{Put}_A v$ and $\mathcal{U}^{**}_{A,v} = \mathsf{Put}^{**}_{A,v}$ with $\mathsf{Put}^{**} = \boldsymbol{pol}_{\mathsf{get}}(\boldsymbol{lens}_{\mathsf{get}}(\mathsf{Put}))$ being the policy implicit in the lens based on the policy $\mathsf{Put}{\in}\textsc{Pol}^{\bullet}_{\mathsf{get}}$.

Thus, multi-valued update policies and single-valued lenses with uncertainty are basically equivalent up to "semantic" closure $x \mapsto x^{\models}$ on one side, and "syntactic closure" $x \mapsto x^{\Vdash}$ on the other side.

5 Related Work

Uncertainty is one of the main problems in software engineering, which typically occurs when the designer does not have the complete, consistent and accurate information required to make a decision during software development. Uncertainty management has been studied in many works, often with the intention to express and represent it in models. In [FSC12, SCHS13], the notion of *partial model* is introduced to let the designer specify uncertain information by suitable annotations of the model. The machinery (called MAVO) is essentially based on FOL and can be seen as a specific instantiation of (some of the) constructs developed in our paper. Some categorical elements were introduced into the MAVO modeling in [SC15], but the latter is still based on FOL syntactically and semantically. A comparison with Libkin's work on uncertainty modeling [Lib14b, Lib14a, Lib15] is discussed above in Sect. 4, see also [Dis16] for more details. Understanding incomplete objects as theories (metamodels), which encode objects' possible completions, was proposed in [BDA+13]. The leading role of the notion of models spaces for the lens framework was emphasized and discussed by Johnson and Rosebrugh in several of their papers and talks [JR16, JR13].

Uncertainty in bidirectional transformations has been often discussed as non-determinism. On the practical side, novel declarative approaches to bidirectionality have been recently proposed for dealing with non-deterministic transformations. Among them, the Janus Transformation Language (JTL) [EMM+10, CDREP10] is a constraint-based model transformation language specifically tailored to support bidirectionality and change propagation. Its relational semantics is able to generate a multitude of models as a solution of a non-deterministic transformation. In [RE15], the JTL engine is revised to accommodate the new *intensional* semantics. This permits a transformation to natively generate a model with uncertainty instead of a myriad of models. In [BHLW07], the authors propose PROGRES, a bx solution based on TGG, which is able to recognize ambiguous mappings and resolve them by interacting with the user, who needs to be an experts in these techniques. An attempt in making bidirectional transformation deterministic by means of intentional updates is represented by the BiFluX language [ZPH14]; however, as a transformation cannot be tested for non-determinism at static-time, the effectiveness of the approach is reduced. In [MC13] a bidirectional transformation approach is proposed, in which the QVT-R semantics is implemented by means of Alloy. Different generated alternatives are obtained from the execution of a model transformation and reduced by either adding extra OCL constraints or by limiting the upper-bound search criteria.

6 Conclusion

Most of the current bx tools and frameworks (including lenses) typically require the transformation writer to consider only one particular consistency restoration strategy among many possible alternatives. Hence, the developers have little or no control over the inherent uncertainty of update propagation. In this paper, we propose an approach to the problem by enriching the delta lens framework with an uncertainty mechanism, which allows managing underspecification and incompleteness within the usual algebra of single-valued operations. Our future plans include the possibility of representing uncertainty in a practically handy way, and assist the modeler with appropriate visualization and tools. For instance, the set of consistency-restoring updates may be encoded by a single feature model, which is convenient for the user to gradually resolve the underspecification later. The formal framework of delta-lenses with uncertainty opens several interesting directions for mathematical modeling of uncertainty. Specifically, update completions make the lens framework essentially bicategorical. Employing the bicategorical conceptual framework and machinery should make the naive (and somewhat bulky and diffuse) formal framework presented in the paper better structured and clearer.

Acknowledgement. We are grateful to BX'16 reviewers for careful reviewing and numerous comments on improving the presentation. Thanks go to Harald Koenig for reading and commenting on some parts of the formal section. Special thanks are for the PC Chairs for their patience in softening several hard deadlines. Financial support for the Canadian part of the author team was provided by Automotive Partnership Canada via the Network on Engineering Complex Software Intensive Systems (NECSIS).

References

[BDA+13] Kacper Bak, Zinovy Diskin, Michal Antkiewicz, Krzysztof Czarnecki, and Andrzej Wasowski. Partial instances via subclassing. In *SLE13*, pages 344–364, 2013.

[BHLW07] Simon M. Becker, Sebastian Herold, Sebastian Lohmann, and Bernhard Westfechtel. A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. *Software and System Modeling*, 6(3):287–315, 2007.

[BS81] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981.

[CDREP10] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. JTL: a bidirectional and change propagating transformation language. In *SLE10*, pages 183–202, 2010.

[Dis16] Zinovy Diskin. Asymmetric Delta-Lenses with Uncertainty: Towards a Formal Framework for Flexible BX. Technical Report GSDLab-TR 2016-03-01, University of Waterloo, 2016. http://gsd.uwaterloo.ca/node/660.

[DXC11] Zinovy Diskin, Yingfei Xiong, and Krzysztof Czarnecki. From State- to Delta-Based Bidirectional Model Transformations: the Asymmetric Case. *Journal of Object Technology*, 10:6: 1–25, 2011.

[EMM+10] R. Eramo, I. Malavolta, H. Muccini, P. Pelliccione, and A. Pierantonio. A model-driven approach to automate the propagation of changes among Architecture Description Languages. *SOSYM*, 1(25):1619–1366, 2010.

[FGM+07] J Nathan Foster, Michael B Greenwald, Jonathan T Moore, Benjamin C Pierce, and Alan Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM TOPLAS*, 29(3):17, 2007.

[FSC12] Michalis Famelis, Rick Salay, and Marsha Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *ICSE*, pages 573–583, 2012.

[JR13] Michael Johnson and Robert D. Rosebrugh. Delta lenses and opfibrations. *ECEASST*, 57, 2013.

[JR16] Michael Johnson and Robert D. Rosebrugh. Unifying set-based, delta-based and edit-based lenses. In *Proceedings BX'16*, 2016. To appear.

[Lib14a] Leonid Libkin. Certain answers as objects and knowledge. In Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter, editors, *Procs of KR 2014*. AAAI Press, 2014.

[Lib14b] Leonid Libkin. Incomplete data: what went wrong, and how to fix it. In Richard Hull and Martin Grohe, editors, *Procs of PODS'14*, pages 1–13. ACM, 2014.

[Lib15] Leonid Libkin. How to define certain answers. In Qiang Yang and Michael Wooldridge, editors, *Procs of IJCAI 2015*, pages 4282–4288. AAAI Press, 2015.

[MC13] Nuno Macedo and Alcino Cunha. Implementing QVT-R Bidirectional Model Transformations Using Alloy. In *FASE*, volume 7793, pages 297–311, 2013.

[RE15] Gianni Rosa Romina Eramo, Alfonso Pierantonio. Managing uncertainty in bidirectional model transformations. In *SLE 2015*, 2015.

[SC15] Rick Salay and Marsha Chechik. A generalized formal framework for partial modeling. In *FASE 2015*, pages 133–148, 2015.

[SCHS13] Rick Salay, Marsha Chechik, Jennifer Horkoff, and Alessio Di Sandro. Managing requirements uncertainty with partial models. *Requir. Eng.*, 18(2):107–128, 2013.

[Ste09] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *SOSYM*, 8, 2009.

[ZPH14] Tao Zan, Hugo Pacheco, and Zhenjiang Hu. Writing bidirectional model transformations as intentional updates. In *ICSE Companion*, pages 488–491, 2014.