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

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* [10]. Its immediate implication is 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 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

In: A. Editor, B. Coeditor (eds.): Proceedings of the XYZ Workshop, Location, Country, DD-MMM-YYYY, published at http://ceur-ws.org

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. In the next section, we present an example to motivate the importance of modeling incompleteness and uncertainty in bx, and 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 [17, 18]. 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* [3]. 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. Since HR and FA work together on the budgeting process, they need an automated synchronizing system (a bx-engine), which was developed by an IT-department star Ian.

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 `Engineers` (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.

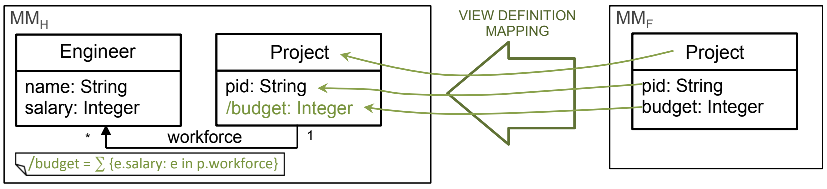


Figure 1: Metamodels and view definition mapping.

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.

Here is the details. 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'.

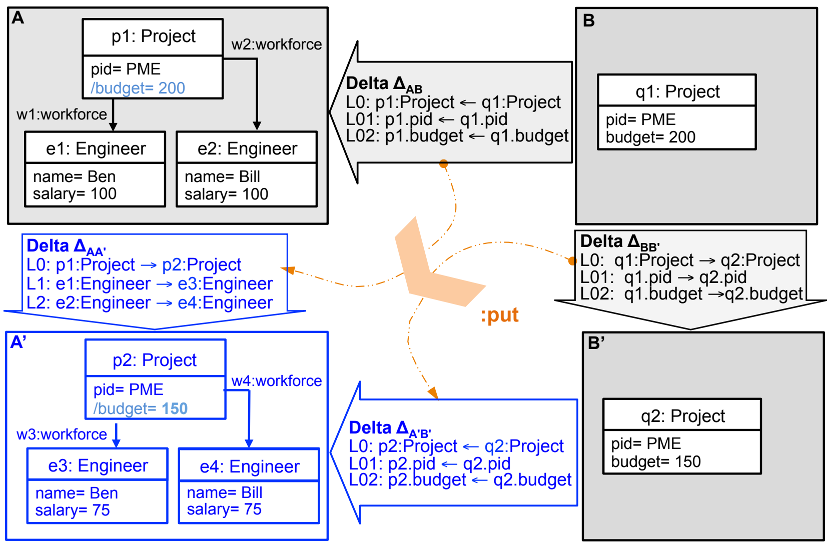


Figure 2: An example of a single-valued delta lens.

As models A and B' are inconsistent (their 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. Note that given models and deltas are shaded, while the derived model and two deltas are blank (and blue with a color display).

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.

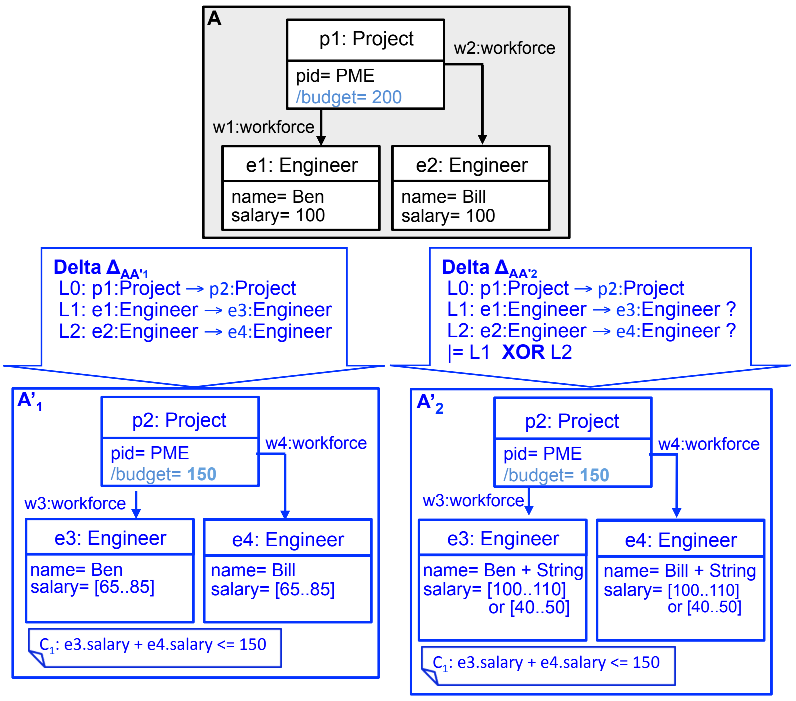


Figure 3: The incomplete models A'_1 and A'_2

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 is not certain yet, and would decide this later.

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.

[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.

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 [2, 16]). 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 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 $\Delta_{AA'}: A \leftarrow A'$ 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 $\Delta_{AA'_1}: A \leftarrow A'_1$ and $\Delta_{AA'_1}: A \leftarrow A'_2$ from Fig. 3 can be seen as *non-final completions* of update $\Delta_{AA'}: A \leftarrow A'$: 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., $C\text{-}Delta$). 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.

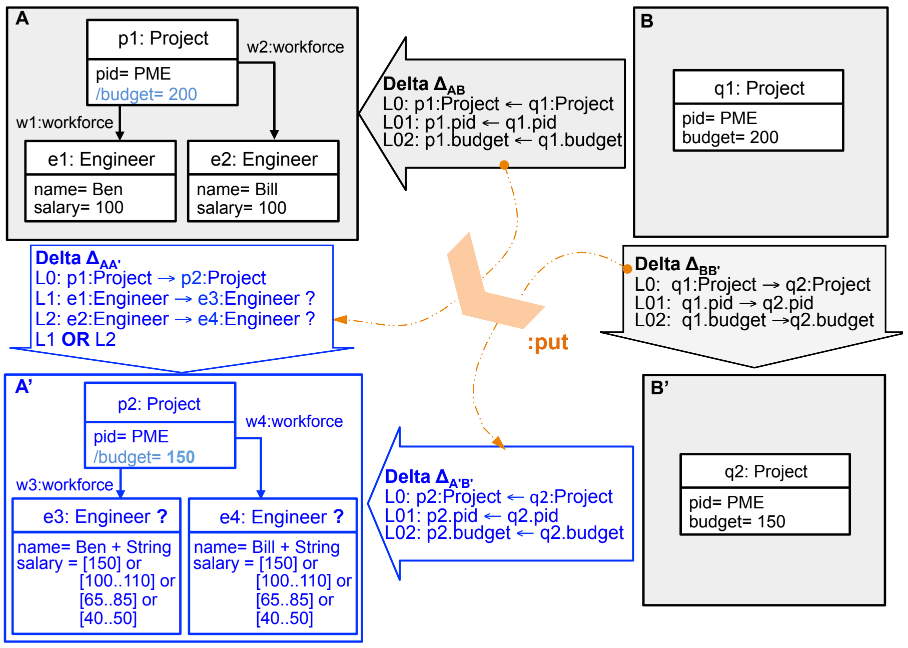


Figure 4: A round-trip scenario with uncertainty.

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 $(e2 \to e4)_{\in \Delta_{AA'_1}}; (e4 \to e4)_{\in C\Delta_{A'_1A'_2}}$ 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.

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 $\Delta_{AA'}: A \leftarrow A'$, 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.

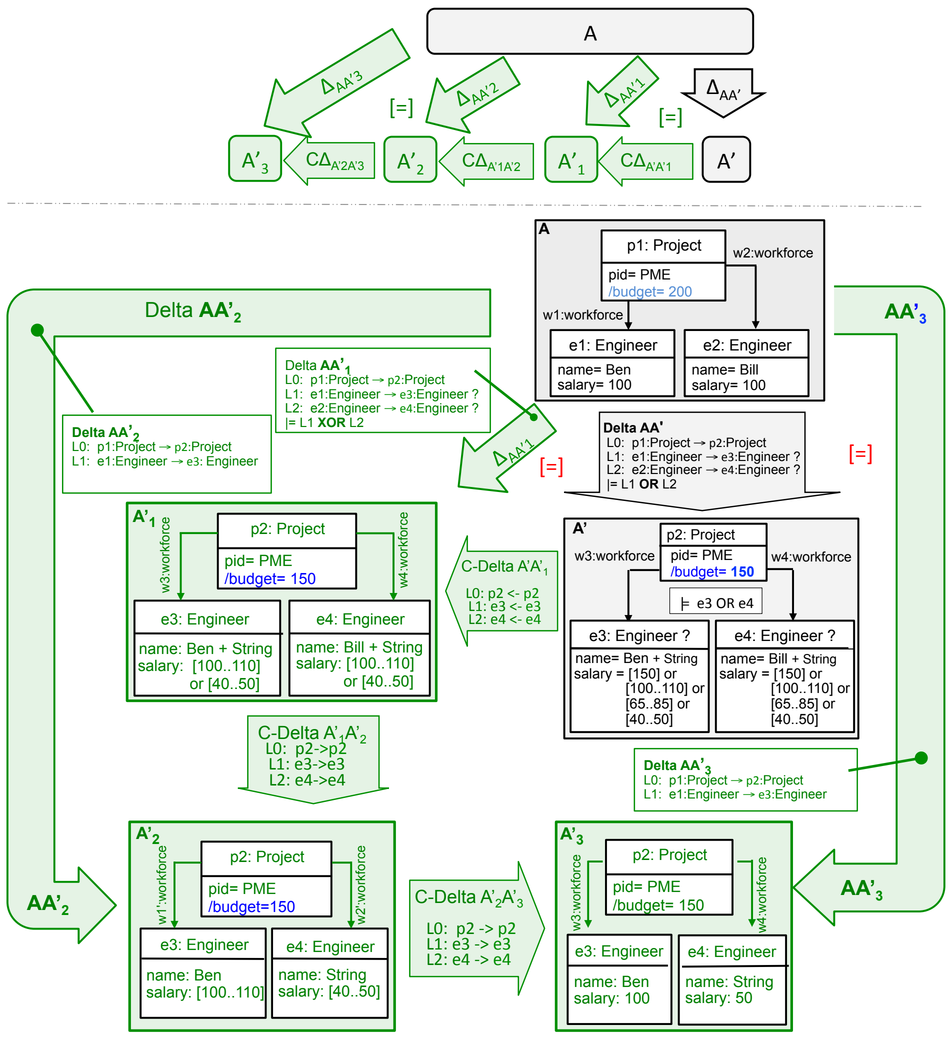


Figure 5: A multi-valued model and its completions within the delta-lenses framework

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 still not decided as which of the engineers is to be kept. 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.

4 Formal framework: Introducing asymmetric delta-lenses with uncertainty

The first issue in building incomplete bx is a suitable enrichment of the notion of a model space. After that, as we will see in this section, the notion of a lens operating over enriched spaces is extracted from the ordinary notion of a delta lens in a more or less direct way. 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 the database and AI research, whose foundations are still debated despite of enormous literature. Specifically, in a recent series of seminal papers published at DB and AI conferences [13, 12, 14, 11], 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 logical 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 [2]. In a nutshell, we formally model the process of uncertain 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 $\mathsf{c} = (A = A^{\mathsf{c}}_0 \to A^{\mathsf{c}}_1 \to A^{\mathsf{c}}_2 \to ... \to A^{\mathsf{c}}_{\dashv})$ with the last term being a fully certain model. Each model A_k in this chain ($k = 0, 1, ...$) can be a branching point of the completion process, which can go to into another direction $\mathsf{c}' = (A_k = A^{\mathsf{c}'}_0 \to A^{\mathsf{c}'}_1 \to ... \to A^{\mathsf{c}'}_{\dashv})$ terminating at another fully certain model $A'_{\dashv}$, 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}. 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 and use uncertainty.

Our plan is as follows. In Sect. 4.2, we formalize the completion process sketched above with the notion of *u-space* ('u' stands for uncertainty), and discuss its relation to Libkin's database domains. A logic (representation system) for u-spaces is introduced in Sect. 4.3. However important are uncertain models and their completions, our main object of interest in BX is updates, and they can also be uncertain as we have seen in our runnong 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, and the necessary categorical preliminaries for dealing with triangles are described in Sect. 4.1 (they are all elementary, but introduce an accurate notation and terminology). Finally, in Sect. 4.5, we introduce *uu-lenses* (read *delta lenses with uncertainty*) as basically ordinary lenses operating over uu-spaces in a uu-structure respecting way. We show that although an uu-lens is a pair of single-valued operations $\ell = (\mathsf{get}, \mathsf{put})$, its restriction to certain models and updates amounts to a pair $(\mathsf{get}, \mathsf{Put}_\ell)$ with a multi-valued operation Put_ℓ respecting the Putget law if ℓ respects it. Conversely, we also show how a view get and a multivalued update policy Put for certain models can be extended for an uu-lens operating over uncertain models and updates. Our main result states that the two notions are equivalent in some precisely defined sense, and hence uu-lenses indeed form a framework for managing uncertain bx, which Ian needs.

Several words about the formalities as such are in order. As is traditional for the lens framework, the mathematical structures are very abstract and specify basic relationships and operations over models and updates without saying what models and updates 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 and XML documents, or models of a FOL theory, enriched with an uncertainty specification mechanism. Model updates are mainly understood as (co)deltas between models, which specify the part of the original model that is kept in the updated model (speaking categorically, such deltas are spans). In stating the axioms our structures should satisfy, we were trying to respect the following requirements. First, to allow for what we feel are right examples, and exclude (what we feel are) wrong ones. Checking the axioms for concrete instantiations is a work in progress, some preliminary results can be found in [TR]. The second requirement is to allow us to prove our main results. Finally, we were trying to comply to some categorical design patterns (whatever that might be). We did not strive to present an economical system of axioms—it is possible that some of them can be inferred from the other, nor we strived to find a most economical presentation. Overall, as checking our formal requirements against major instantiations of the framework is a work in progress, our formal defintions are somewhat experimental and subject to change.

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.

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 $\boldsymbol{C}_{\mathsf{prop}}$. Specifically, this gives us three subcategories $\boldsymbol{C}_{\mathsf{iso}}, \boldsymbol{C}_{\mathsf{epi}}, \boldsymbol{C}_{\mathsf{mono}}$.

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 $f'A$.

Families and products. Given a set X, the range set of a family $\mathbf{x}=\{\!\{\mathbf{x}_i\in X\colon i\in I\}\!\}$ (note the double brackets) is denoted by $\mathbf{x}^{\#}=\{\mathbf{x}_i\in X\colon i\in X\}$ (note single brackets that say the collection is a set). A family of arrows $\mathbf{u}=\{\!\{\mathbf{u}_i\in\boldsymbol{C}\colon i\in I\}\!\}$ gives rise to a family of objects $\mathbf{u}^{\bullet}=\{\!\{\mathbf{u}_i{}^{\bullet}\in\boldsymbol{C}^{\bullet}\colon i\in I\}\!\}$. Note that if even $\mathbf{u}$ is a set (injective family), collection $\mathbf{u}^{\bullet}$ can still be a (non-injective) family. A *span* is a family of arrows (called the *legs* of the span) with a common source (called the *head*).

Given a family of $\boldsymbol{C}$-objects $\mathbf{A}=\{\!\{\mathbf{A}_i\in\boldsymbol{C}^{\bullet}\colon i\in I\}\!\}$, its (chosen) product is a span $\Pi\mathbf{A}=\{\!\{\Pi_i\mathbf{A}\in\boldsymbol{C}(\Pi^{\bullet}\mathbf{A},*)\colon i\in I\}\!\}$, whose head is denoted by $\Pi^{\bullet}\mathbf{A}$, and legs $\Pi_i\mathbf{A}$ are called *projections*. (Of course, a more accurate notation would be $(\Pi\mathbf{A})^{\bullet}$ and $(\Pi\mathbf{A})_i$.) Universality of products means that for any span $\mathbf{u}=\{\!\{\mathbf{u}_i\in\boldsymbol{C}(A,*)\colon i\in I\}\!\}$ (where A denotes the head of the span), there is a uniquely defined arrow $\mathbf{u}!\colon A\to\Pi^{\bullet}\mathbf{u}^{\bullet}$ such that $\mathbf{u}!\,;\Pi_i\mathbf{u}=\mathbf{u}_i$.

A functor $f\colon\boldsymbol{C}\to\boldsymbol{D}$ between categories with products is said to *preserve products* (or *continuous* if for any family of $\boldsymbol{C}$-objects $\mathbf{A}=\{\!\{A_i\colon i\in I\}\!\}$, we have $f(\Pi\mathbf{A})=\Pi(f\mathbf{A})$, where $f\mathbf{A}=\{\!\{fA_i\colon i\in I\}\!\}$. In detail, the equality means that $f(\Pi^{\bullet}\mathbf{A})=\Pi^{\bullet}f\mathbf{A}$ and $f(\Pi_i\mathbf{A})=\Pi_i(f\mathbf{A})$.

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 $A/\boldsymbol{C}$.

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 $f^*u\overset{\mathrm{def}}{=}f;u\colon B\to *$, 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.

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

Definition 1 (U-spaces) A *space of uncertain models (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 $\boldsymbol{T}_{\boldsymbol{C}'\subseteq_{\bullet}\boldsymbol{C}}$ 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 $f\colon \boldsymbol{M} \to \boldsymbol{N}$ 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 $i\colon A \to B$ 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 valuers 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 [2]).

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$i\colon A \to B$ 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 $c\colon A \to^{\preccurlyeq} B$ and $c'\colon A \,{}^{\succcurlyeq}\!\!\leftarrow B$ 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.

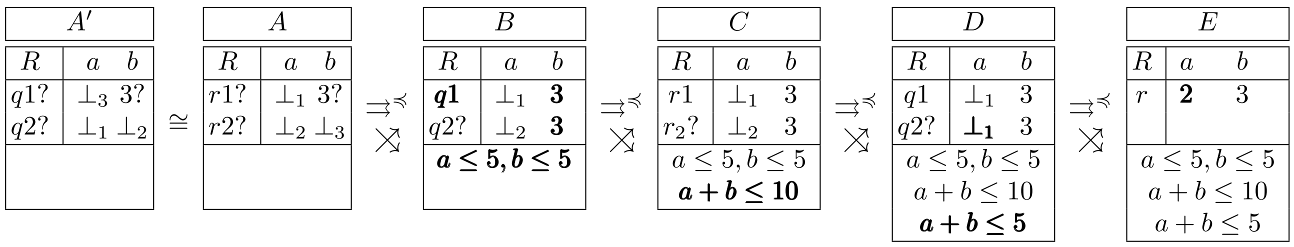
Definition 2 (full completions) If the only $\preccurlyeq$-arrows leaving model A are isomorphisms, $\boldsymbol{M}_{\preccurlyeq}(A,*) = \boldsymbol{M}_{\mathsf{iso}}(A,*)$, we call A *(fully) certain* or *complete*. We denote the class of all complete models by $\boldsymbol{M}^{\bullet}_{\dashv}$, where symbol $\dashv$ is used to suggest the termination of the completion process at complete models. We write $c\colon A \to^{\preccurlyeq} \dashv$ 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 both attributes are declared to be optional for objects instantiating class R (in this sense R is a semi-structured relation, which can be instantiated which tuples having one or no columns). 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) contains 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 $\bot_1$: in a legal completion, any value allowed by the metamodel (i.e., a positive integer in our case) can be substituted for $\bot_1$. (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.

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 $\bot_3@r2$ was substituted by value 3. However, the above understanding of the update silently assumes that object ri



A': R | a b; $q1?$ $\bot_3$ $3?$; $q2?$ $\bot_1$ $\bot_2$
$\cong$
A: R | a b; $r1?$ $\bot_1$ $3?$; $r2?$ $\bot_2$ $\bot_3$
$\rightrightarrows^{\preccurlyeq}$
B: R | a b; $\boldsymbol{q1}$ $\bot_1$ $\mathbf{3}$; $q2?$ $\bot_2$ $\mathbf{3}$; $\boldsymbol{a \le 5, b \le 5}$
$\rightrightarrows^{\preccurlyeq}$
C: R | a b; $r1$ $\bot_1$ 3; $r_2?$ $\bot_2$ 3; $a \le 5, b \le 5$; $\boldsymbol{a + b \le 10}$
$\rightrightarrows^{\preccurlyeq}$
D: R | a b; $q1$ $\bot_1$ 3; $q2?$ $\boldsymbol{\bot_1}$ 3; $a \le 5, b \le 5$; $a + b \le 10$; $\boldsymbol{a + b \le 5}$
$\rightrightarrows^{\preccurlyeq}$
E: R | a b; r $\mathbf{2}$ 3; $a \le 5, b \le 5$; $a + b \le 10$; $a + b \le 5$

Figure 6: Model completions vs. model updates

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. 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 $a + b \le 10$ 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 $\bot_2@r2$ was replaced by null $\bot_1@q2$, but this change is *not* a trivial null renaming because null $\bot_1$ was already used in model C for attribute $a@r1$. 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. Similarly acts the new constraint $a+b \le 5$ added to D, and hence the parallel delta from C to D is a legal $\preccurlyeq$-arrow. Indeed, any full completion of D can be transformed to a full completion of C by precomposition with the delta, while model C evidently has more full 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 $\bot_1@q1$ and deletes $q2$. In contrast, the cross delta deletes mandatory object $r1$ (and hence is a general update rather than a completion), resolves optionality of $q2$ and substitutes 2 for $\bot_1@q2$. 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 (which would glue two objects but leave the null uncertain). $\square$

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, 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 laws (iso), (preord), and (epi) on the right. We say an u-space *excludes inconsistency* if (consist) holds as well (i.e., every model is consistent).

(iso)	$\boldsymbol{M}_{\mathsf{iso}} \subseteq_{\bullet} \boldsymbol{M}_{\preccurlyeq}$
(preord)	$c\colon A \rightarrow^{\preccurlyeq} B$ & $c'\colon A \;{}^{\succcurlyeq}\!\!\leftarrow B$ imply $c{\in}\boldsymbol{M}_{\mathsf{iso}}$ & $c'{\in}\boldsymbol{M}_{\mathsf{iso}}$
(epi)	$\boldsymbol{M}_{\preccurlyeq} \subseteq_{\bullet} \boldsymbol{M}_{\mathsf{epi}}$
(consist)	for any $A{\in}\boldsymbol{M}^{\bullet}$, there is a full completion $c\colon A \rightarrow^{\preccurlyeq} \dashv$

4.3 Logic for uncertainty

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; double figure brackets (and double bullet) indicate that the expression denotes 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 image 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 fully complete, then $\mathsf{FC}^{\bullet\#}_{\preccurlyeq}(A)$ consists of all isomorphic copies of A.

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 [6]). Libkin's definition of the informational preorder, $A \preceq A'$ iff $[\![\, A' \,]\!] \subseteq [\![\, A \,]\!]$, is subsumed by the pre-composition construction (see Sect. 4.1) that lifts any $\preccurlyeq$-arrow $c\colon A \rightarrow^{\preccurlyeq} A'$ to a function $c^{*}\colon \mathsf{FC}_{\preccurlyeq}(A) \leftarrow \mathsf{FC}_{\preccurlyeq}(A')$. It gives an immediate arrow counterpart of Libkin's pre-order if c^{*} is injection, which is indeed the case as we require $\preccurlyeq$-arrows to be epimorphisms. 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. We call a $\preccurlyeq$-arrow $c\colon A \rightarrow^{\preccurlyeq} B$ a *semantic equivalence*, if function $c^{*}\colon \mathsf{FC}_{\preccurlyeq}(A) \leftarrow \mathsf{FC}_{\preccurlyeq}(B)$ is bijective; then we write $c\colon A \rightarrow^{\approx} B$. In the TR [TR], the relation between fair database domain (introduced in [11] 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 $\mathsf{Mod}\Phi = \{X{\in}\, \mathsf{Mod}(\Sigma)\colon X \models \phi$ for all $\phi{\in}\Phi\}$. Operator Th maps a class of models $\mathcal{X}$ to its theory $\mathsf{Th}\mathcal{X} = \{\phi{\in}\Phi(\Sigma)\colon X \models \phi$ for all $X{\in}\mathcal{X}\}$. Operator $\mathsf{FC}_{\preccurlyeq}$ is an arrow counterpart of Mod, but we also need an 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 $c\colon A \rightarrow^{\preccurlyeq} C$. 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 $X!\colon X \rightarrow^{\preccurlyeq} \mathsf{Th}\,\mathcal{C}$. 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 is precisely the categorical reformulation of Libkin's consideration of theories as greatest lower bounds in the information pre-order [14].

Definition 5 (Model spaces cont'd: products) A pre-wb u-space $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ is *well-behaved (wb)* if category $\boldsymbol{M}_{\preccurlyeq}$ has products denoted by $\Pi^{\bullet}_{\preccurlyeq}$.

Specifically, for any model A, there is a model $\Pi^{\bullet}_{\preccurlyeq}\mathsf{FC}^{\bullet}_{\preccurlyeq}(A)$ denoted by $A^{\models}$ (the notation is suggested by the discussion above as $\Pi^{\bullet}_{\preccurlyeq}\mathsf{FC}^{\bullet}_{\preccurlyeq}(A)$ can be understood as $\mathsf{ThMod}A$). Dually, for any family of certain models $\mathbf{A} = \{\!\{A_i\colon i{\in}I_{\mathbf{A}}\}\!\}$ there is a family $\mathsf{FC}^{\bullet}_{\preccurlyeq}(\Pi^{\bullet}_{\preccurlyeq}\mathbf{A})$ (understood as $\mathsf{ModTh}\mathbf{A}$) and denoted by $\mathbf{A}^{\vdash}$. □

The following lemma (a straightforward consequence of the universality of products) provides the arrow counterparts of the well-known model-theoretic inclusions: $\Phi \subset \mathsf{ThMod}\,\Phi$ for any theory Φ, and $\mathbf{A} \subset \mathsf{ModTh}\,\mathbf{A}$ for any class of models $\mathbf{A}$. It shows that the arrow completion construction can indeed be seen as a logic.

Lemma 1 (model-theoretic inclusions) *(a)* For any model A, there is a uniquely defined semantic equivalence $A!\colon A \rightarrow^{\approx} A^{\models}$. That is, sets $\mathsf{FC}_{\preccurlyeq}(A)$ and $\mathsf{FC}_{\preccurlyeq}(A^{\models})$ are bijective via $A!^{*}$, and any full completion of A is factorized through a full completion of $A^{\models}$.

(b) For any family of models $\mathbf{A} = \{\!\{A_i\colon i \in I\}\!\}$, there is a uniquely defined injection $\mathbf{A}!\colon I \rightarrow \mathsf{FC}_{\preccurlyeq}(\Pi^{\bullet}\mathbf{A})$ such that $\mathbf{A}!(i) = \Pi_i\mathbf{A}$ so that all projections are included into $\mathbf{A}^{\vdash}$.

[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*.

Definition 6 (closed objects) Model A is called *closed* if $A!\colon A \twoheadrightarrow^{\approx} A^{\models}$ is isomorphism in $\boldsymbol{M}$, i.e., A and $A^{\models}$ are isomorphic via $A!$. A family $\mathbf{A}$ is called *closed* if $\mathbf{A}!$ is bijection, i.e., families $\mathbf{A}$ and $\mathbf{A}^{\vdash}$ only differ in the indexing set but otherwise are the same.

Lemma 2 (certain objects are closed) If model A is certain, then $A!\colon A \twoheadrightarrow^{\approx} A^{\models}$ is an isomorphism. (Hint: any completion of a complete update is an iso.)

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 [6], 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. **??**) will be denoted by $\boldsymbol{T}_{\preccurlyeq}$.

Definition 7 (uu-spaces) A *model space with uncertain models and uncertain updates*, or *uu-space*, is a triple $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leq})$ such that the pair $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ is an u-space, and $\boldsymbol{T}_{\leq}$ is an object-full subfunctor of $\boldsymbol{T}_{\preccurlyeq}$ (i.e., $\boldsymbol{T}_{\leq} \subseteq_{\bullet} \boldsymbol{T}_{\preccurlyeq}$ see Sect. 4.1) such that for any $A{\in}\boldsymbol{M}^{\bullet}$, the pair $(\boldsymbol{T}_{\leq}A, \boldsymbol{T}_{\preccurlyeq}A)$ 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 $\boldsymbol{T}_{\leq}A$, whose objects are updates, $\boldsymbol{T}_{\leq}^{\bullet}A = \boldsymbol{M}(A,*)$, and morphisms are some of triangles based on $\preccurlyeq$-arrows (see the triangle subsection in Sect. 4.1), i.e., any $\boldsymbol{T}_{\leq}A$ morphism is a triangle (A, u, u', b) whose base b is an $\boldsymbol{M}_{\preccurlyeq}$-arrow, but not all such triangles are $\boldsymbol{T}_{\leq}A$ morphisms. We call $\boldsymbol{T}_{\leq}A$ morphisms *update completions*, or *certainty non-decreasing triangles*, or just $\leq$-arrows, and denote such an arrow from update $u\colon A \to *$ to update $u'\colon A \to *$ by a double arrow $\tau_c^A\colon u \Rightarrow^{\leq} u'$ with 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, $\boldsymbol{T}_{\leq}A \subseteq_{\bullet} \boldsymbol{T}_{\preccurlyeq}A \subseteq_{\bullet} \boldsymbol{T}A$ 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 $\tau_c^A\colon u \Rightarrow^{\leq} u'$ in category $\boldsymbol{T}_{\leq}A$ is mapped to arrow $\tau_c^B\colon f;u \Rightarrow^{\leq} f;u'$ in $\boldsymbol{T}_{\leq}B$.

Definition 8 (well-behaved uu-spaces) Uu-space is called *well-behaved (wb)* if all u-spaces $(\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq})$ and $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leq}A)$, $A{\in}\boldsymbol{M}^{\bullet}$ are wb, and, in addition, conditions in the inset table on the right hold. These conditions are explained below. □

(Comm)	$(\boldsymbol{T}_{[=]} \cap \boldsymbol{T}_{\leq}) \subset \boldsymbol{T}_{\leq}$
(Iso)	$(\boldsymbol{T}_{\leq}A)_{\mathsf{iso}} = \boldsymbol{T}_{\mathsf{iso}}A \cap \boldsymbol{T}_{[=]}A$
(CertCert)	$\boldsymbol{M}_{\vdash\!\dashv} \subset \boldsymbol{M}$.

We require any commutative triangle based on a $\preccurlyeq$-arrow (i.e., a quadruple $\tau_c = (A, u, c, u')$ with $c\colon u^{\bullet} \twoheadrightarrow^{\preccurlyeq} u'^{\bullet}$ and $u' = u;i$) to be a $\leq$-arrow $\tau_c\colon u \Rightarrow^{\leq} u'$. 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.

To define full update completions, i.e., to apply the general u-space Definition 2 to updates-as-objects, we need to discuss update isomorphisms in categories $\boldsymbol{T}_{\leq}A$. It is easy to see that any commutative triangle $(A, u, u; i, i)$ based on isomorphism $i\colon u^{\bullet} \twoheadrightarrow^{\cong} u'^{\bullet}$ is an isomorphism in $\boldsymbol{T}_{\leq}A$. 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 the general definition for updates as objects. If the only $\leq$-triangles from update $u\colon A \to *$ are its isomorphisms, $\boldsymbol{T}_{\leq}A(u,*) = \boldsymbol{T}_{\leq\mathsf{iso}}A(u,*)$, then u is called *(fully) certain* or *complete*, and let $\boldsymbol{T}_{\dashv}^{\bullet}A$ denotes the class of all such in the category $\boldsymbol{T}_{\leq}A$.. We write $\tau\colon u \Rightarrow^{\leq} \dashv$ to say that $\tau^{\bullet} \in \boldsymbol{T}_{\dashv}^{\bullet}A$, and call τ a *full completion* of u.

Let $\mathsf{FC}_{\leq}^{A}(u)$ or just $\mathsf{FC}_{\leq}(u)$ denote the class of all such, and $\mathsf{FC}_{\leq}^{\bullet}(u) \overset{\text{def}}{=} \{\!\{\tau^{\bullet}\colon \tau{\in}\mathsf{FC}_{\leq}^{\bullet\#}(u)\}\!\}$ is the respective family of complete updates. Thus, update u is complete iff $\mathsf{FC}_{\leq}^{\bullet\#}(u)$ 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.

Finally, let $\boldsymbol{M}_{\vdash\!\dashv} \subset \boldsymbol{M}$ denotes the class of all (fully) certain models and certain updates between them: $\boldsymbol{M}_{\vdash\!\dashv}^{\bullet} = \boldsymbol{M}_{\dashv}^{\bullet}$ and $\boldsymbol{M}_{\vdash\!\dashv}(A,*) = \boldsymbol{T}_{\dashv}^{\bullet}(A,*)$. Our last law (CertCert) states that composition of two certain updates is certain, i.e., the sub-universe of certain models and updates is a subcategory, $\boldsymbol{M}_{\vdash\!\dashv} \subset \boldsymbol{M}$.

The following is the rewrite of general u-space constructions on page 11 for u-spaces $(\boldsymbol{T}_{\preccurlyeq}A, \boldsymbol{T}_{\leq}A)$, $A{\in}\boldsymbol{M}^{\bullet}$.

Lemma 3 (model-theoretic inclusions) *(a)* For any model $A \in \boldsymbol{M}^\bullet$ and update $u\colon A \to *$, there is a uniquely defined semantic equivalence $u!\colon u \Rightarrow^{\approx} u^{\models}$. That is, sets $\mathsf{FC}^A_{\leq}(u)$ and $\mathsf{FC}^A_{\leq}(u^{\models})$ are bijective via $u!^*$, and any full completion of u is factorized through a full completion of $u^{\models}$.

(b) For any family of updates $\mathbf{u} = \{\!\{u_i\colon i \in I\}\!\}$, there is a uniquely defined injection $\mathbf{u}!\colon I \to \mathsf{FC}_{\leq}(\Pi^\bullet_{\leq}\mathbf{u})$ such that $\mathbf{u}!(i) = (\Pi_{\leq}\mathbf{u})_i$ so that all projections are included into $\mathbf{u}^{\vdash}$.

Definition 9 (closed objects) Update $u\colon A \to *$ is called *closed* if $u!\colon u \Rightarrow^{\approx} u^{\models}$ is isomorphism in $\boldsymbol{T}_{\leq}A$, i.e., u and $u^{\models}$ are isomorphic via $u!$. A family $\mathbf{u}$ is called *closed* if $\mathbf{u}!$ is bijection, i.e., families $\mathbf{u}$ and $\mathbf{u}^{\vdash}$ only differ in the indexing set but otherwise are the same.

Lemma 4 (certain objects are closed) If update u is certain, then $u!\colon u \Rightarrow^{\approx} u^{\models}$ is an isomorphism. *(Hint: any completion of a complete update is an iso.)* $\square$

4.5 Lenses over uu-spaces

We turn to lenses working over model spaces with uncertainty. We will call them *u-lenses*, and in this paper only consider the asymmetric case.

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

Definition 10 (delta lenses) Let $\boldsymbol{M}$, $\boldsymbol{N}$ are two categories considered as model spaces. A *lens* $\ell\colon \boldsymbol{M} \rightleftharpoons \boldsymbol{N}$ 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 $\mathsf{put} = (\mathsf{put}_A)_{A\in\boldsymbol{M}^\bullet}$ 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 $(\mathsf{put}_A v)^\bullet$.

A lens is called *well-behaved (wb)* if following two conditions (laws) are satisfied for all $A\in\boldsymbol{M}^\bullet$ (below B stands for $\mathsf{get}A$):
(i) *Identity preservation*: $\mathsf{put}_A(\mathsf{id}_B) = \mathsf{id}_A$.
(ii) *PutGet law*: $\mathsf{get}(\mathsf{put}_A v) = v$ for all updates$v\colon B \to *$.

Note that, in general, having a triangle of updates on the view side,$v\colon B \to B', v'\colon B' \to B''$, and$v''\colon B \to B''$, we don't require its put-image to be a triangle on the source side, i.e., we admit the situation,when $\mathsf{put}^\bullet_A v'' \neq \mathsf{put}^\bullet_{A'} v'$ with A' standing for $\mathsf{put}^\bullet_A v$.

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.
(iii) *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, moreover, also commutative.)

□

This definition is equivalent to the delta lens definition in [6] with a minor distinction that in [6], 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 a 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 construct relaxes this requirement.

Definition 11 (update policies) Let$\mathsf{get}\colon \mathcal{M} \to \mathcal{N}$ be a wb view between uu-spaces. An *update policy* for get is a family of set-valued functions

$$\mathsf{Put}_A\colon 2^{\boldsymbol{M}(A,*)} \leftarrow \boldsymbol{N}(\mathsf{get}A,*)$$

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 $\{u^\bullet\colon u\in\mathsf{Put}_A v\}$.

A policy is called *well-behaved (wb)* if the following conditions are satisfied for all $A\in\boldsymbol{M}^\bullet$ and $v\in\boldsymbol{N}(B,*)$ ((below B stands for $\mathsf{get}A$):
(i) *PutGet law*: $\mathsf{Put}_A(v) \subset \mathsf{get}^{-1}(v)$
(ii) *Identity preservation*: $\mathsf{Put}_A(\mathsf{id}_B) = \{\mathsf{id}_A\}$. (iii) *Closedness*: $\mathsf{Put}_A(v)$ is a closed set of updates, i.e., $\mathsf{Put}^{\vdash}_A(v) = \mathsf{Put}_A(v)$, where we write $\mathsf{Put}^{\vdash}_A(v)$ for $(\mathsf{Put}_A v)^{\vdash}$

Definition 12 (multi-lenses) A *lens based on a multi-valued update policy*, or just a *multilens* is a pair $\mu = (\mathsf{get}, \mathsf{Put})$ with$\mathsf{get}\colon \boldsymbol{M} \to \boldsymbol{N}$ a functor, and $\mathsf{Put} = (A\in\boldsymbol{M}^\bullet)_{\mathsf{Put}_A}$ $\mathsf{Put} = \{\!\{A\in\boldsymbol{M}^\bullet\colon \mathsf{Put}_A\}\!\}$ a family of set-valued functions

$$\mathsf{Put}_A\colon 2^{\boldsymbol{M}(A,*)} \leftarrow \boldsymbol{N}(\mathsf{get}A,*)$$

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 $\{u^\bullet\colon u\in\mathsf{Put}_A v\}$.

A multilens is called *well-behaved (wb)* if the following two laws are satisfied (below B stands for $\mathsf{get}A$):
(i) *Identity preservation*: $\mathsf{Put}_A(\mathsf{id}_B) = \{\mathsf{id}_A\}$.
(ii) *PutGet law*: For all$v\colon B \to *$ and $u\in\mathsf{Put}_A(v)$, equality$\mathsf{get}(u) = v$ holds. In other words, $\mathsf{Put}_A(v) \subset \mathsf{get}^{-1}(v)$.

The main idea of lenses with uncertainty is to realize a mutlilens $\mu\colon \boldsymbol{X} \rightleftharpoons \boldsymbol{Y}$ between ordinary spaces (i.e., just categories) via a *single-valued* lens $\mu^+\colon \boldsymbol{X}^+ \rightleftharpoons \boldsymbol{Y}^+$ operating over u-spaces extending the original model spaces with the uncertainty mechanism described in Sect. 4.2. Below we will realize a dual version of this idea by going in the opposite direction of reduction rather than expansion: first, we will define a notion of a well-behaved lens $\ell\colon \mathcal{M} \rightleftharpoons \mathcal{N}$ *over u-spaces*, and then will show how a wb multilens $\ell^-\colon \mathcal{M}^- \rightleftharpoons \mathcal{N}^-$ (where $\mathcal{M}^-$, $\mathcal{N}^-$ are just categories) can be extracted from ℓ. The expansion direction is left for future work.

Definition 13 (u-lenses) Let $\mathcal{M} = (\boldsymbol{M}, \boldsymbol{M}_{\preccurlyeq}, \boldsymbol{T}_{\leq})$, $\mathcal{N} = (\boldsymbol{N}, \boldsymbol{N}_{\preccurlyeq}, \boldsymbol{U}_{\leq})$ be two model spaces with uncertainty and products (where $\boldsymbol{U}$ is the triangle functor for $\boldsymbol{N}$). An *(asymmetric wb) u-lens* $\ell\colon \mathcal{M} \rightleftharpoons \mathcal{N}$ is a wb delta lens $\ell = (\mathsf{get}, \mathsf{put})\colon \boldsymbol{M} \rightleftharpoons \boldsymbol{N}$ between the carrier categories as defined above in Def. 10, which additionally satisfies the following conditions of compatibility with uncertainty:
(u-get) get is a wb u-functor as defined in Def. **??**
(u-put) $\mathsf{put}_A\colon \boldsymbol{T}_{\mathcal{M}} A \leftarrow \boldsymbol{T}_{\mathcal{N}}\mathsf{get}(A)$ is a product preserving functor for all A.
(2-PutGet law) ????

Note that this "horizontal" functoriality of put assumes some sort of "vertical" functoriality (the infamous PutPut law) wrt. compositions of general vertical updates and special model completion horizontal updates (where directions of vertical and horizontal correspond to the upper schema in Fig. 5).

Definition 14 (arrows between lenses) Let $\ell 1 = (\mathsf{get}, \mathsf{put1})$ and $\ell 2 = (\mathsf{get}, \mathsf{put2})$ be two lenses over the same view get: $\mathcal{M} \to \mathcal{N}$. A *completion* arrow from $\ell 1$ to $\ell 2$ is a family of product preserving natural transformations $\tau\tau_A\colon \mathsf{put1}_A \Rightarrow \mathsf{put2}_A$ indexed by $A \in \boldsymbol{M}^\bullet$.[4] That is, for any A and $v\colon \mathsf{get}A \to *$, we have a triangle completion $\tau\tau_A(v)\colon \mathsf{put1}_A(v) \Rightarrow \mathsf{put2}_A(v)$, and naturality means that for any completion $\tau\colon v \Rightarrow w$, we have a commutative square: $\tau\tau_A(v); \mathsf{put2}_A(\tau) = \mathsf{put1}_A(\tau); \tau\tau_A(w)$.

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

Theorem 5 Given a wb view get: $\mathcal{M} \to \mathcal{N}$ between u-model spaces, any update policy for get gives rise to a wb lens over get. Moreover, this correspondence extends to a functor $lens\colon Pol(\mathsf{get}) \to Lens(\mathsf{get})$.

Proof. Let Put be an update policy for get, $A \in \boldsymbol{M}^\bullet$, and having these data, we need to define mapping $\mathsf{put}_A\colon \boldsymbol{M}(A,*) \leftarrow \boldsymbol{N}(\mathsf{get}A,*)$. Below model A remains fixed, and we will skip index A to simplify notation, and use B for $\mathsf{get}A$. We will build mapping put in three consecutive steps. First, we define mapping $\mathsf{put}^{\dashv}$ for fully certain updates $v \in \boldsymbol{T}^\bullet_{\dashv} B$ by setting $\mathsf{put}^{\dashv}(v) = \Pi\mathsf{Put}(v)$ (note that it maps certain updates into uncertain ones to properly represent the update policy). Note that the Putget law holds by construction itself.

The next step is to define mapping $\mathsf{put}^{\models}$ for uncertain but closed updates (such that $v = v^{\models}$) by the continuous extension of mapping $\mathsf{put}^{\dashv}$. We set $\mathsf{put}^{\models}(v) = \Pi\mathsf{put}^{\dashv}(\mathsf{FC}^{\bullet\#}(v))$, where the argument of Π denotes the family $\mathsf{FC}^{\bullet\#}(v); \mathsf{put}^{\dashv}$ of updates in $\boldsymbol{T}^\bullet_{\dashv}A$. For a full completion $\tau\colon v \Rightarrow^{\dashv} w$, we define $\mathsf{put}^{\models}(\tau)$ to be the corresponding projection $\pi_\tau\colon \mathsf{put}^{\models}(v) \Rightarrow \mathsf{put}^{\dashv}(w)$. To make $\mathsf{put}^{\models}$ a functor, we need to define $\mathsf{put}^{\models}(\tau)$ for an arbitrary completion arrow $\tau\colon v \Rightarrow w$ in the category $\boldsymbol{T}_{\mathcal{N}}(B)$. Recall that precomposition with τ gives us a function $\tau^*\colon \mathsf{FC}(v) \leftarrow \mathsf{FC}(w)$, and a commutative diagram

$$\begin{tikzcd}
\mathsf{FC}_w \arrow[r, "\tau^*"] \arrow[d, "f"'] \arrow[dr, "\mathsf{FC}^\bullet(w)"] & \mathsf{FC}_v \arrow[d, "\mathsf{FC}^\bullet(v)"] \\
\boldsymbol{M}(A,*) & \boldsymbol{T}^\bullet_{\dashv}(B) \arrow[l, "\mathsf{put}^{\dashv}"']
\end{tikzcd} \tag{1}$$

Then

$$\begin{array}{rcll}
\mathsf{put}^{\models}(w) & = & \Pi\mathsf{FC}^\bullet(w); \mathsf{put}^{\dashv} & \text{by def of } \mathsf{put}^{\models} \\
 & = & \Pi(\tau^*; \mathsf{FC}^\bullet(v)); \mathsf{put}^{\dashv} & \text{diagram (1)} \\
 & = & \Pi\tau^*; (\mathsf{FC}^\bullet(v); \mathsf{put}^{\dashv}) & \text{associativity}
\end{array}$$

And as $\mathsf{put}^{\models}(v) = \Pi(\mathsf{FC}^\bullet(v); \mathsf{put}^{\dashv})$, universality of products gives us arrow $\tau^*!\colon \mathsf{put}^{\models}(v) \Rightarrow \mathsf{put}^{\models}(w)$. Thus, we have extended mapping $\mathsf{put}^{\dashv}\colon \boldsymbol{M}(A,*) \leftarrow \boldsymbol{T}^\bullet_{\dashv}(B)$ to a functor $\mathsf{put}^{\models}\colon \boldsymbol{T}^\bullet_{\models}(A) \leftarrow \boldsymbol{T}^\bullet_{\models}(B)$. The Putget law holds by continuity.

The last step is to chose a representation of $\mathsf{put}^{\models}(v)$
===============

Lemma 6 (correctness of uncertain put) *Mappings* put *in a wb lens restore consistency: if update* $v\colon \mathsf{get}(A) \to *$ *is fully certain, although* $u = \mathsf{put}_A(v)$ *can be (and as a rule is) uncertain, any full u's completion* $\tau\colon u \Rightarrow^{\preccurlyeq} \dashv$ *is mapped back to v, and thus* $\mathsf{get}(\tau^{\bullet\bullet}) = v^\bullet$.

Proof. PutGet law implies that $\mathsf{get}\tau\colon v \Rightarrow^{\preccurlyeq} *$, and as get preserves completeness, we actually have $\mathsf{get}\tau\colon v \Rightarrow^{\preccurlyeq} \dashv$. However, v is complete, hence, $\mathsf{get}(\tau)$ is identity and $\mathsf{get}(\tau^\bullet) = v$. Particularly, $\mathsf{get}(\tau^{\bullet\bullet}) = v^\bullet$. □

The following easy result is important.

[4]Read $\tau\tau$ as double τ rather than π.

Definition 15 (u-lens composition) Let $\ell_1 : \boldsymbol{M} \rightleftharpoons \boldsymbol{N}$, $\ell_2 : \boldsymbol{N} \rightleftharpoons \boldsymbol{O}$ be two u-lenses. Their *composition* $\ell_{12} : \boldsymbol{M} \rightleftharpoons \boldsymbol{O}$ is the following u-lens. The view get_{12}: $\boldsymbol{M} \to \boldsymbol{O}$ is given by the ordinary composition of functors: $\mathsf{get}_{12} = \mathsf{get}_1; \mathsf{get}_2$ (we skip symbol ℓ in indexes). The family put_{12} is defined as follows. Let $A \in \boldsymbol{M}^\bullet$, $B = \mathsf{get}_1(A)$, $C = \mathsf{get}_2(B)$, and $w \in \boldsymbol{O}(C, *)$. To define $\mathsf{put}_{12A}(w)$, we first define $v = \mathsf{put}_{2B}(w) \in \boldsymbol{N}(B, *)$, and then $u = \mathsf{put}_{1A}(v) \in \boldsymbol{M}(A, *)$.

Theorem 7 *a) The construction above defines an u-lens. b) Composition is associative, and the* identity lens $(\mathsf{id}_{\boldsymbol{M}}, \mathsf{id}_{\boldsymbol{M}}) : \boldsymbol{M} \rightleftharpoons \boldsymbol{M}$ *is its unit. c) If both lenses ℓ_1, ℓ_2 are wb, their composition is wb as well. Any identity u-lens is trivially wb.*

Proof. (a) Definition 15 actually defines $\mathsf{put}_{12\,A}$ as the composition of graph morphisms. (b) is straightforward. (c) Identity and $\preccurlyeq$-arrow preservations are obvious. For checking the PutGet law, let $\tau \in \mathsf{FC}_{\boldsymbol{TM}A}(u)$ (with the notation in Def. 15).Then $\mathsf{get}_1(\tau) \in \mathsf{FC}_{\boldsymbol{TN}B}(v)$ by the PutGet law for ℓ_1, and $\mathsf{get}_2(\mathsf{get}_1(\tau)) \in \mathsf{FC}_{\boldsymbol{TO}C}(w)$ by the PutGet law for ℓ_2. Thus, the PutGet law for $(\mathsf{get}_{12}, \mathsf{put}_{12})$ holds as well.

Corollary 8 *Asymmetric wb u-lenses with their composition and identity as defined above form a category.*

4.5.1 Future work.

Graph-morphisms put_A may enjoy various functorial properties: composition preservation (of updates and completion triangles), and product preservation. Understanding semantic and formal aspects of these properties is an important issue. Specifically, the possibilities of up-to-isomorphism definedness of the main constructs are to be understood. A more flexible and more general formalization of model spaces is possible based on 2-categories (or bicategories) with 2-arrows defined axiomatically rather than by triangles.

5 Related Work

In software engineering, uncertainty is one of the main problems that 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 [8], the notion of *partial model* is introduced in order to let the designer specify uncertain information by means of a base model enriched with annotations and first-order logic. Model transformation techniques typically operate under the assumption that models do not contain uncertainty. Nevertheless, the work in [9] proposes a technique for adapting existing model transformations in order to deal with models containing uncertainty. In particular, a lifting operation allows to adapt unidirectional transformations to be used over models with uncertainty preserving their original behavior.

Uncertainty in bidirectional transformations has been discussed and handled in different forms. In [1], formal and generalized study of bx is given. The proposed techniques also extend bx to non-deterministic transformations. Recently, novel declarative approaches to bidirectionality have been proposed for dealing with non-deterministic transformations. Among them, the Janus Transformation Language (JTL) [7, 5] 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 solution of a non-deterministic transformation. In [16], 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. An attempt in making bidirectional transformation deterministic by means of intentional updates is represented by the BiFluX language [18]; however, as a transformation cannot be tested for non-determinism at static-time, the effectiveness of the approach is reduced. In [15] 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. Similar results are obtained in [4] by using integer linear programming.

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. Understanding incomplete objects as theories (metamodels), which encode objects' possible completions, was proposed and developed in [2]. A detailed comparison with Libkin's work on uncertainty modeling [13, 12, 14, 11] is provided above in Sect. 4, and can be extended to comparison with other work on uncertainty in databases via Libkin's framework as the "junction" point.

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 inherit 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 variability within the usual algebra of single-valued operations.

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.

References

[1] F. Abou-Saleh, J. Cheney, J. Gibbons, J. McKinna, and P. Stevens. Notions of Bidirectional Computation and Entangled State Monads. *MPC*, pages 187–214, 2015.

[2] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski. Partial instances via subclassing. In *SLE13*, pages 344–364, 2013.

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

[4] G. Callow and R. Kalawsky. A Satisficing Bi-Directional Model Transformation Engine using Mixed Integer Linear Programming. *JOT*, 12(1):1: 1–43, 2013.

[5] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. JTL: a bidirectional and change propagating transformation language. In *SLE10*, pages 183–202, 2010.

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

[7] 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.

[8] M. Famelis, R. Salay, and M. Chechik. Partial models: Towards modeling and reasoning with uncertainty. In *ICSE*, pages 573–583, 2012.

[9] M. Famelis, R. Salay, A. Di Sandro, and M. Chechik. Transformation of models containing uncertainty. In *MoDELS*, pages 673–689, 2013.

[10] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM TOPLAS*, 29(3):17, 2007.

[11] A. Gheerbrant, L. Libkin, and C. Sirangelo. When is naive evaluation possible? In R. Hull and W. Fan, editors, *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013, New York, NY, USA - June 22 - 27, 2013*, pages 75–86. ACM, 2013.

[12] L. Libkin. Certain answers as objects and knowledge. In C. Baral, G. D. Giacomo, and T. Eiter, editors, *Procs of KR 2014*. AAAI Press, 2014.

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

[14] L. Libkin. How to define certain answers. In Q. Yang and M. Wooldridge, editors, *Procs of IJCAI 2015*, pages 4282–4288. AAAI Press, 2015.

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

[16] G. R. Romina Eramo, Alfonso Pierantonio. Managing uncertainty in bidirectional model transformations. In *SLE 2015*, 2015.

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

[18] T. Zan, H. Pacheco, and Z. Hu. Writing bidirectional model transformations as intentional updates. In *ICSE Companion*, pages 488–491, 2014.