Learning as Change Propagation with Delta Lenses

Zinovy Diskin

diskinz@mcmaster.ca

McMaster University, Hamilton, Canada

July 8, 2019

Delta lenses are an established mathematical framework for modelling and designing bidirectional model transformations (Bx). To adapt it for machine learning (ML) problems (following the recent observations by Fong et al), the paper introduces a new ingredient of the lens framework — parameterization and learning, defines the notion of an asymmetric learning delta lens with amendment (ala-lens), and shows how ala-lenses can be organized into a symmetric monoidal category. On the Bx side, ala-lenses describe bidirectional model transformations, in which change propagation can also change the transformation definition. The paper discusses several possibilities of a fruitful Bx-ML interaction, and proposes several directions for future work.

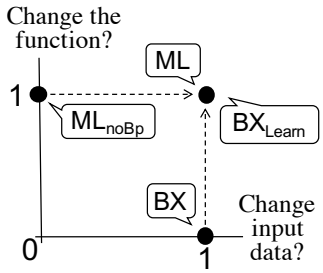
1 Introduction

In a seminal paper [9], Fong, Spivak and Tuyéras showed how to compose supervised machine learning (ML) algorithms so that the latter form a symmetric monoidal (sm) category ***Learn***, and built an sm-functor L: ***Para*** $\to$ ***Learn***, which maps a parameterized differentiable function f: $P \times \mathbb{R}^m \to \mathbb{R}^n$ (with the parameter space $P = \mathbb{R}^k$) to a learning algorithm that improves an initially given function $f(p_0, \_)$: $\mathbb{R}^m \to \mathbb{R}^n$ by learning from a set of training pairs $(a, b) \in \mathbb{R}^m \times \mathbb{R}^n$. Recently, Fong and Johnson noticed in [8] (quoting them directly) "surprising links between two apparently disparate areas": ML (treated compositionally as above) and bidirectional model transformations or BX (also treated compositionally in a framework of mathematical structures called *lenses* [10]), whereas "naively at least, there seemed to be little reason to expect them to be closely related mathematically".[1] Fong and Johnson also built an sm-functor ***Learn*** $\to$ ***sLens*** which maps learning algorithms to so called symmetric lenses.

[1] Term BX abbreviates "bidirectional X (something)" and refers to bidirectional change propagation in different contexts in different domains: file synchronization in versioning, data exchange in databases, model synchronization

The goal of the present paper is to introduce a framework in which similarities between ML and BX become apparent. We will see that by abstracting away some details (and ignoring the contextual differences), both problems can be seen as two different instantiations of the same algebraic structure defined in the paper and called *asymmetric learning lens, al-lens*. Indeed, lenses are devices specifying change propagation, and learning can be seen as a special change propagation case. For example, given (i) sets A, B, P and a family of functions $f = (p \in P \mid f_p\colon A \to B)$ as the learning space, and (ii) a training datum (a, b') with $b' \neq b = f_p(a)$ for an initially given parameter value p, we can interpret it as a change from a consistent state of the system (a, p, b) to an inconsistent state (a, p, b') caused by the change $b \rightsquigarrow b'$. BX would fix this inconsistency by changing the input data $a \rightsquigarrow a'$ so that the state (a', p, b') is consistent, i.e., $f_p(a') = b'$, and term this action as *propagating* change $b \rightsquigarrow b'$ to change $a \rightsquigarrow a'$. ML would call the change $b \rightsquigarrow b'$ an error, and fix it by changing the parameter value $p \rightsquigarrow p'$ so that the state (a, p', b') is either consistent, or at least less inconsistent than (a, p, b') w.r.t. some metrics, e.g., we can add metrics to set B and require that $||f_{p'}(a), b'|| < ||f_p(a), b'||$. In ML terms, the system has *learnt* a better approximation $f_{p'}$ of the function; in BX terms, change $b \rightsquigarrow b'$ is propagated to $p \rightsquigarrow p'$.

Abstract learners defined in [9] do both propagations so that the updated/learnt state is (a', p', b'). The story is schematically described in the inset figure, in which the horizontal dashed arrow shows ML's evolution towards having back propagation (Bp) and the vertical arrow shows BX's evolution towards BX with learning ($\mathrm{BX_L}$, still to be developed; the work on deriving model transformations from examples [11] can be seen as $\mathrm{BX_L}$). In the lens framework developed for BX, point (1,1) corresponds to the notion of a learning lens, which covers both ML and BX. In the paper, we will work with asymmetric rather than symmetric lenses, which is a) technically much simpler and b) conceptually justified: compositional ML considered in [9] is about composing functions rather than relations, and an asymmetric model seems to be more appropriate. The main result of [8] can then be seen as an "internal lens business": it says that any codiscrete al-lens can be encoded by a special symmetric non-learning lens.



Thus, the notion of a learning lens was actually discovered in [9]. The present paper makes two further contributions to the learning lens idea. The first could be called cate-

and model transformation in Model-Driven software Engineering (MDE), see [2] for some of these contexts. What makes BX a special and united community amongst many circles studying synchronization and change propagation, is a foundational lens-based algebraic framework. But within BX , the application context can vary, sometimes significantly. In the present paper, BX will mainly refer to BX in the MDE context — the author's area of expertize in BX .

gorification: we will define and work with learning *delta* lenses, in which a change/delta between objects x, y is an arrow $u\colon x \to y$ rather than a pair (x, y); moreover, multiple different updates/deltas are possible between the same objects x, y so that $Hom(x, y)$ is a set (non-empty for a typical BX application in MDE). Each species of a delta lens has its codiscrete version, in which $Hom(x, y) = \{(x, y)\}$, and learners defined in [9] are exactly codiscrete al-lenses (moreover, category ***Learn*** is a full subcategory of the category of asymmetric delta learning lenses ***alLens***). In this paper, a lens will always mean a delta lens by default.

In the MDE context, objects x, y are complex structures so that their delta/update $u\colon x \to y$ is a complex structure too. Its ignorance in the codiscrete lens formalism may create fundamental problems for applying codiscrete lenses to practical problems (see [6]). The role of deltas in the ML context, where objects x, y are tuples of real numbers, is less clear — we will discuss it in Sect. 2.2.

The second technical contribution of the paper is specification of algebraic laws for learning lenses and studying their preservation under lens composition (sequential and parallel). For BX, this is a crucial aspect: it says that if several synchronization modules satisfy some laws (are *well-behaved* in the lens parlance), then their composition is automatically well-behaved and hence the (usually expensive) integration testing is not needed. However, a major asymmetric BX lens law that requires consistency of the system after change propagation (called the PutGet law in the lens parlance) does not always hold in practice and needs to be relaxed [4]. The relaxed PutGet is based on the idea of self-propagation: the change $b \rightsquigarrow b'$ is amended by change $b' \rightsquigarrow b'^{@}$ so that the state $(a', p', b'^{@})$ is (more) consistent (than (a', p', b')). Thus, BX needs lenses with amendment, and the main construct of the paper is the notion of an *asymmetric learning lens with amendment, ala-lens*, for which we will formulate three basic laws. We will define sequential and parallel composition of ala-lenses, study their compatibility with the three laws, and build an sm-category of ala-lenses, ***alaLens***. The role of algebraic laws in the ML context is an intriguing subject left for future work.

Our plan for the paper is as follows. The next section is motivational (why delta lenses can be useful in ML) and semi-formal; all formal results are presented in Sect. 3 and 4, and in Sect. 5 we discuss future work. In more detail, Sect. 2.1 is a short primer on delta lenses with amendments, and in Sect. 2.2 we discuss possible roles of metric and even non-metric deltas in ML. Section 3 defines ala-lenses and several algebraic laws for them motivated by the MDE context. Section 4 defines sequential and parallel composition of ala-lenses and proves two main results: a) preservation of lens laws by lens compositions, and b) the possibility to organize ala-lenses into an sm-category (proofs are placed into the Appendix).

Several remarks about the notation used in the paper. Many formulas specify terms built from operations going in the opposite directions (this is in the nature of the lens formalism). To ease reading formulas,minimize the number of brackets, and relate a formula to its supporting diagram, we denote the application of function f to argument x by either $f(x)$, $f.x$ or $x.f$, and sometimes we will even use different ways within the same formula. For example, if operation get maps from the left to the right, operation put maps in the opposite direction, and x is an argument in the domain of get, we tend to write formula $x' = \mathsf{put}(\mathsf{get}(x))$ as $x' = \mathsf{put}(x.\mathsf{get})$ while if y is an argument in the domain of put, we tend to write the formula $y' = \mathsf{get}(\mathsf{put}(y))$ as $(\mathsf{put}.y).\mathsf{get} = y'$ or $\mathsf{get}(\mathsf{put}.y) = y'$. Unfortunately, this discipline was recognized too late in the process of writing, and thus is not always followed so that some notational mix remained.

Given a category $\mathbf{C}$, its collection of objects is denoted by $|\mathbf{C}|$. An arrow with domain $A \in |\mathbf{C}|$ is denoted by $u\colon A \to \_$ and similarly $u\colon \_ \to B$ is an arrow with codomain B. A subcategory $\mathbf{B} \subset \mathbf{C}$ is called *wide* if it has the same objects.

Colours in diagrams help to restore the scope of groups of operations, and partially their order (workflow), various sub- and super-scripts aim at helping this too.[2] Yet another means to facilitate understanding the workflow underlying a complex diagram is to use animation (unfortunately, only applicable for presentations): the present author's experience in presenting complex diagrammatic terms (typical for categorical reasoning) for non-categorical audiences shows that animation can be quite useful, hence the use of Powerpoint for complex diagrams in the paper.

2 From BX to ML

Learners defined in [9] are codiscrete learning lenses, and in this section we semi-formally discuss enriching them with deltas: metrical deltas in Sect. Sect. 2.2.1 (based on the notion of Lawvere's metric space) and non-metrical deltas in Sect. Sect. 2.2.2 – the latter may perhaps be an essential extension of the standard ML. In short Sect. Sect. 2.2.3, we discuss amendments.

2.1 Background: BX world of model spaces and delta lenses

In the (delta-based) BX, the counterpart of values in Euclidean spaces dealt with in ML are complex artifacts called *models*. They can be seen as models of logical theories (of the first- or even higher-order) called *metamodels*. A typical way of specifying metamodels is to use graphs and constraints; in fact, metamodels can be seen as Makkai's generalized

[2] This resulted in perhaps excessive indexing but, again, it was understood too lately in the process of writing.

sketches [12], see also [5] for definitions and discussion in the MDE context. For our current discussion, it's enough to think about models as graphs with some (unspecified) extra structure, which is good enough to ensure the required categorical properties (the reader can think about *attributed typed graphs* as defined in [7]); we will call them 'graphs'. In MDE applications, it normally does not make sense to distinguish isomorphic 'graphs' (indeed, objects identifiers are invisible) and thus we identify models with isomorphisms classes of 'graphs' and assume categories of models to be skeletal.

Let $\mathbf{G}$ be a category of 'graphs'/models, $G, G' \in |\mathbf{G}|$ are two objects, and $\tilde{u}\colon G \rightsquigarrow G'$ is some informal notion of a model update (delta, change). We can formalize it by a span of monics $u = (G \xleftarrow{i} O \xrightarrow{i'} G')$ whose head O specifies what part of G is preserved in G' so that the elements of G beyond the image $i(O)$ are deleted, and the elements of G' beyond $i'(O)$ are added during the update. We assume that $\mathbf{G}$ has pullbacks so that spans are composable (and produce spans of monics), and we obtain a skeletal category $\mathsf{Span}_{\mathsf{mon}}(\mathbf{G})$,whose arrows are called *updates* or *deltas*. The delta lens framework abstracts away these details and simply identifies model spaces with skeletal categories and forward transformations between model spaces with functors; the latter are usually called *get functors* or just *gets* (read " get the view") — a detailed discussion and examples can be found in [6].[3]

Now we schematically discuss how ala-lenses work and relate them to the ML setting in [9].While an ordinary asymmetric lens "inverts" a given functor $\mathsf{get}\colon \mathbf{A} \to \mathbf{B}$ between model spaces, an ala-lens "inverts" ***and*** "learns" over a family of get functors as specified in the top row of diagram Fig. 1(a). This row specifies "types" of the story, while the story itself is unravelled on the level of their instances shown by the rectangular diagram below.

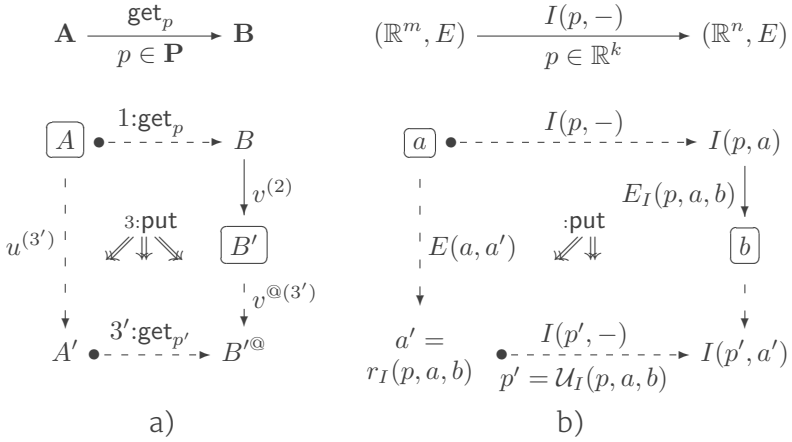


Figure 1: Change propagation in ala-lenses. Derived/computed elements are specially visualized: nodes are not framed, arrows are dashed.

Suppose a given model $A \in \mathbf{A}$ and its view, model $A.\mathsf{get}_p = B \in \mathbf{B}$, so that the state (A, p, B) is consistent. Note our notation for the fact $A.\mathsf{get}_p = B$: bulleted arrows $x \bullet\!\!\longrightarrow y$ don't have an internal structure and just visualize pairs (x, y) — the UML calls them

[3]Not all useful transformations are functors, but many are, and it is a standard assumption in the delta lens framework.

links. The label over the link (A, B) in Fig. 1(a) says that the link named 1 is an instance of executing the function get_p at object A.[4] Now suppose that B is updated to B' with delta $v^{(2)} \in \mathbf{B}(B, B')$ as shown. We use numeric indexes for elements of the diagram to additionally convey the order of operations producing them, e.g., 1 means that production of link 1 by operation get_p is the first step of the story, and update $v^{(2)}$ is the second step. There is no intention to make this notation fully consistent (the more so formal); it is a syntactic sugar with some semantic connotations useful for reading complex diagrams further in the paper. In the asymmetric lens setting, any change of view B destroys consistency (it's not so for the symmetric version of lenses), and an ala-lens restores it by calling an operation put (step 3) which a) changes the parameter from p to p', b) changes the source model A with update $u^{(3')}$, and c) amends the update-violator v with an *amendment* update $v^{@}$ so that the result of these changes is a consistent system $A'.\mathsf{get}_{p'} = B^{@}$ as shown in the diagram. Thus, put is a triple of operations $(\mathsf{put}^{\mathsf{upd}}, \mathsf{put}^{\mathsf{req}}, \mathsf{put}^{\mathsf{self}})$, which for a given pair (A, v) returns a corresponding triple of updates $e = \mathsf{put}^{\mathsf{upd}}(A, v)\colon p \to p'$ (not shown in the diagram), $u = \mathsf{put}^{\mathsf{req}}(A, v)\colon A \to A'$, $v^{@} = \mathsf{put}^{\mathsf{self}}(A, v)\colon B' \to B'^{@}$; the superscripts upd, req abbreviate the corresponding names used in [9]. Moreover, we require a much stronger consistency condition to hold—the equality $u.\mathsf{get}_{p'} = v; v^{@}$ between updates—because it is update/delta $v; v^{@}$ which gives model $B'^{@}$ its real meaning in a typical MDE scenario. In the lens parlance, the last equation and its versions are called the Putget law: if we substitute $\mathsf{put}(v)$ for u, we obtain equality $(\mathsf{put}.v).\mathsf{get}_{p'} = v; v^{@}$ to hold for any v, hence, the name of the law.

Diagram Fig. 1(b) describes the signature of an ML-learner as defined in [9], where Euclidean spaces are turned into (enriched) categories (Lawvere metric spaces) by using some metric/error functions E—in the next section, we will discuss this in more detail. The notation in diagram Fig. 1(b) is borrowed from [9] to ease seeing the parallelism (but note a discrepancy: B' in Fig. 1(a) corresponds to b without prime in Fig. 1(b)).

2.2 Enriching ML with deltas

2.2.1 Metric deltas

The main result of [9] states the existence of an sm-functor $L_{\varepsilon,\mathsf{err}}\colon \boldsymbol{Para} \to \boldsymbol{Learn}$, where $\boldsymbol{Para}$ is the category whose objects are Euclidean spaces $\mathbb{R}^n$, $n \in \mathbb{N}$, and arrows are differentiable parameterized functions between them, i.e., (equivalence classes of) differentiable

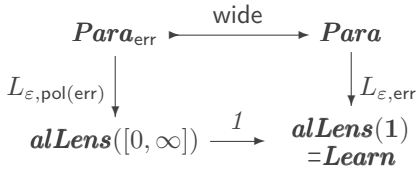
[4] In a fuller version of the story with the type side explicitly given by a category $\mathcal{D}ef$ of metamodels and transformation definitions, over which a category of models is fibred, link 1 would be the Cartesian lifting of transformation definition $get_p\colon \mathcal{M} \leftarrow \mathcal{N}$ at object A living in fibre $\mathbf{A}$ over metamodel $\mathcal{M}$, and arriving at B in the fibre $\mathbf{B}$ over $\mathcal{N}$. Although the syntactical side is important for applications, lenses traditionally ignore it and we follow this style in the paper.

functions $f\colon P \times \mathbb{R}^m \to \mathbb{R}^n$ with the parameter space $P = \mathbb{R}^k$. Functor $L_{\varepsilon,\mathsf{err}}$ is determined by two parameters: a *step size* $0 < \varepsilon \in \mathbb{R}$ and an *error function* $\mathsf{err}(x,y)\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$ such that $\frac{\partial \mathsf{err}}{\partial x}(x_0, \_)\colon \mathbb{R} \to \mathbb{R}$ is invertible for any given $x_0 \in \mathbb{R}$. In BX terms, ε and err determine a *propagation policy* necessary to determine the put operations in the lens $L_{\varepsilon,\mathsf{err}}(f)$. Importantly, while err is needed for all put operations as it determines the cost function, i.e., the *total error* function $E_I(p,a,b) = \sum_{i\le n} \mathsf{err}(I_i(p,a), b_i)$ to be minimized by changes in (p,a,b), the step size ε is only needed for computing the change of p while changing a is based on a predefined idea of invertibility (see [9, Sect.III] for details). In this light, a more suggestive notation for L would be $L_{\mathsf{err},\varepsilon,\mathsf{inv}}$.

The delta lens framework, gives us a somewhat different interpretation. Given an *error* function $\mathsf{err}\colon \mathbb{R}\times\mathbb{R} \to \mathbb{R}$, an n-dimensional Euclidean space $\mathbb{R}^n$ can be converted into a metric space by defining the *total* error $\mathsf{Err}\colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ by $\mathsf{Err}(x,x') = \sum_{i\le n} \mathsf{err}(x_i, x'_i)$. This also makes space $\mathbb{R}^n$ a category enriched over Lawvere's monoidal poset $([0,\infty], \ge, +, 0)$ (Lawvere's metric space) $\mathbf{R}^n_{\mathsf{err}}$, and get-view functors $f_p\colon \mathbf{R}^m_{\mathsf{err}} \to \mathbf{R}^n_{\mathsf{err}}$ are exactly non-expanding (i.e., distance non-increasing) functions. It gives us a category $\boldsymbol{Para}_{\mathsf{err}}$, which is widely embedded into $\boldsymbol{Para}$ but has fewer arrows: only non-expanding functions from $\boldsymbol{Para}$ occur into $\boldsymbol{Para}_{\mathsf{err}}$. Now the learning functors $L_{\mathsf{err},\varepsilon,\mathsf{inv}}\colon \boldsymbol{Para} \to \boldsymbol{Learn}$ are reshaped as functors $L_{\varepsilon,\mathsf{inv}}\colon \boldsymbol{Para}_{\mathsf{err}} \to \boldsymbol{Learn}$.

The invertibility condition for err is essential for the above formulation as it ensures definability of $\mathsf{put}^{\mathsf{req}}$ via the policy inv. In a more general setting, function err can be not invertible at some points, e.g., cross entropy distance is such (see [9, Sect.VIIA]). If this more general err is "good enough", we can still define $\mathsf{put}^{\mathsf{req}}$ by using a trickier propagation policy $\mathsf{pol}(\mathsf{err})$ but, importantly, it must be compositional and give rise to an sm-functor $L_{\varepsilon,\mathsf{pol}(\mathsf{err})}\colon \boldsymbol{Para}_{\mathsf{err}} \to \boldsymbol{alLens}([0,\infty])$, where $\boldsymbol{alLens}([0,\infty])$ denotes the category of $[0,\infty]$-enriched al-lenses, i.e., lenses defined for model spaces and get-functors being categories and functors enriched over $[0,\infty]$ rather than $\boldsymbol{Set}$ (and hence being metric spaces and non-expanding maps). Actually, the existence of a compositional policy $\mathsf{pol}(\mathsf{err})$ can serve as a formal definition of being "good enough".

The discussion above is summarized by the commutative diagram on the right, in which the following constructs are used. We first note that a codiscrete al-lens, i.e., a learner, can be seen as an al-lens defined for categories and functors enriched over a singleton category $\mathbf{1} = \{*\}$, which is also a monoidal poset in a trivial way. Then, the canonic morphisms $1\colon [0,\infty] \to \mathbf{1}$ of monoidal posets gives rise to the change of base functor $1\colon \boldsymbol{Cat}([0,\infty]) \to \boldsymbol{Cat}(\mathbf{1})$ between enriched categories and functors and, similarly, to a functor $1\colon \boldsymbol{alLens}([0,\infty]) \to \boldsymbol{alLens}(\mathbf{1})(= \boldsymbol{Learn})$. That is, we assume that we have a general



definition of an al-lens, whose ingredients are categories and functors and operations enriched over an sm-category $\mathcal{V}$ (this definition is still to be developed — see remarks in Future work section 5), and thus we have a category of such lenses $\boldsymbol{alLens}(\mathcal{V})$. If $\mathcal{V}$ is not mentioned, it is assumed to be $\boldsymbol{Set}$ (and $\boldsymbol{alLens}$ will be defined and shown to be sm in Sect.3-4). Note that $\boldsymbol{Cat}(\mathbf{1}) \cong \boldsymbol{Set}$ and $\boldsymbol{alLens}(\mathbf{1}) \cong \boldsymbol{alLens}_0$ with 0 meaning codiscreteness.

.

2.2.2 Non-metric deltas

In this section, we consider (somewhat speculatively) possibilities of non-metrical deltas between values in Euclidean spaces. The parameter space $\mathbf{P}$ is determined by the learning object — a neural network $\mathcal{N}$, and the gradient descent algorithm (Gd) optimizes the parameter value $p \in \mathbb{R}^k = |\mathbf{P}|$ built from biases of neurons in internal layers and weights of connections to and from them; the number k is determined by the network configuration and remains constant. Suppose, however, that on the learning path determined by Gd and back propagation (Bp), some internal layer neuron X always fires; then it makes sense to remove it from the internal layer but add an input neuron constantly producing the corresponding input data. Then numbers k and m (the number of input variables) change. Or, an intelligent learning algorithm can detect that learning improvement goes slowly due to the data from the same input neuron Y so that removal of Y and its connections significantly decreases the total error (e.g., think of a broken pixel in an image recognition scenario). This removal will change m and k (which depends on the number of connections). In general, we can assume intelligent learning algorithms that restructure the network $\mathcal{N}$ and change the number of variables in the input $\mathbb{R}^m$, parameter $\mathbb{R}^k$, and output $\mathbb{R}^n$, spaces. To make intelligent decisions related to restructuring of the network, the learning algorithm should have a richer notion of deltas at the input and at the output. Below is a sketch of what such deltas could be as suggested by delta lenses.

Consider a model space to be $\mathbb{R}^X$ with X a finite set of variables $\{x_1 ... x_{n_X}\}$ (with n_X the cardinality of X). An object of the space is a valuation $a\colon X \to \mathbb{R}$. An updated object is a valuation $a'\colon X' \to \mathbb{R}$ where $X' = \{x'_1 ... x'_{n_{X'}}\}$ is an updated set of variables, which, in general, somehow overlaps with X. The overlap is specified by a span of inclusions $X \xleftarrow{\iota_u} O_u \xrightarrow{\iota'_u} X'$, where index u refers to the update/delta arrow $u\colon a \to a'$. A delta thus consists of three parts: set of deleted variables $X \setminus \iota_u(O_u)$, set of inserted variables $X' \setminus \iota'_u(O_u)$, and an error $E_u = \sum_{x \in O_u} ||a(\iota.x), a'(x.\iota')||$.

2.2.3 ML with amendments

Backward propagation to the input neuron layer does not make sense as input data are given, but backward propagation to an internal layer turned out a fruitful idea. Dually, it does not make sense to consider amendment to the output layer as output data are also given, but amendments to an internal layer may turn out useful. In fact, backward propagation and amendments can be seen as two versions of basically the same idea: an intelligent optimization should be given the possibility to change the internal layer data irrespective to whether this layer is the input or the output of the training cell (= delta lens square as shown in Fig. 1(a).

3 Asymmetric Learning Lenses with Amendment (ALA-Lenses)

Definition 1 (Model spaces) *A* model space *is a small skeletal category whose objects and arrows are called, resp.,* models *and* deltas *(or* updates*).*

Definition 2 (ala-lenses) *Let* $\mathbf{S}$ *and* $\mathbf{T}$ *be model spaces. An* ala-lens *from* $\mathbf{S}$ *(the* source *of the lens) to* $\mathbf{T}$ *(the* target*) is given by the following three components a-c.*

a) Parameters: *A category* $\mathbf{P}$ *of* parameters *(think about function/view definitions parameterized by objects of* $\mathbf{P}$*). We denote objects of* $\mathbf{P}$ *by small letters* $p, p', ..$ *rather than capital ones to avoid confusion with [9], in which capital* P *is used for the entire parameter set.*

b) Gets: *A functor* $\mathsf{get}\colon \mathbf{P} \to [\mathbf{S}, \mathbf{T}]$*, where* $[\mathbf{S}, \mathbf{T}]$ *is the category of functors from* $\mathbf{S}$ *to* $\mathbf{T}$ *and their natural transformations. We will write* $\mathsf{get}(p, \_)$ *or* get_p *or (or even shorter* p^**) for the functor* $\mathsf{get}(p)\colon \mathbf{S} \to \mathbf{T}$ *(read "get* $\mathbf{T}$*-views of* $\mathbf{S}$*"). Given a parameter delta* $e\colon p \to p'$ *and a source model* $S \in |\mathbf{S}|$*, the model delta* $\mathsf{get}(e)\colon \mathsf{get}_p(S) \to \mathsf{get}_{p'}(S)$ *will be denoted in formulas by* $e.\mathsf{get}_S$ *(rather than* $\mathsf{get}_e(S)$*) to smoothly use the dot notation for substitution; in diagrams, we write* e_S*.*

c) Puts: *A family of operations* $\mathsf{put}_{p,S}$ *indexed by pairs* $(p \in |\mathbf{P}|, S \in |\mathbf{S}|)$*. Each operation in the family takes a delta* $v\colon \mathsf{get}_p(S) \to T'$ *as its input, and outputs three deltas* $(\mathsf{put}^{\mathbf{x}}_{p,S}(v) \mid \mathbf{x} \in \{\mathsf{req}, \mathsf{upd}, \mathsf{self}\})$ *specified below (names* req *and* upd *are chosen to match the terminology in [9]).*

- **c1)** $u = \mathsf{put}^{\mathsf{req}}_{p,S}(v)\colon S \to S'$ *is a* source *update (read "put the view update* v *back")*
- **c2)** $e = \mathsf{put}^{\mathsf{upd}}_{p,S}(v)\colon p \to p'$ *is a parameter update or transformation* evolution*.*
- **c3)** $v^{@} = \mathsf{put}^{\mathsf{self}}_{p,S}(v)\colon T' \to T^{@}$ *is an* amendment *to the original update,*

A lens is called codiscrete *if categories* $\mathbf{S}$, $\mathbf{T}$, $\mathbf{P}$ *are codiscrete.*

Diagram in Fig. 2 shows how these operations are interrelated. The upper part shows an arrow $e\colon p \to p'$ in category $\mathbf{P}$ and two corresponding functors from $\mathbf{S}$ to $\mathbf{T}$. The lower part is to be seen as a 3D-prism with visible front $ST_{p'}T^{@}S'$ and upper $ST_pT_{p'}$ faces, the bottom and back side faces are invisible and the corresponding arrows are dashed. The prism denotes an algebraic term (workflow): given elements are shown with black fill and white font, while derived elements are blue (recalls being mechanically computed) and blank (double-body arrows are considered to be blank). The two pairs of arrows originating from S and S' are not blank because they denote pairs of nodes (the UML says *links*) rather than mappings/deltas between nodes —we explained this notation in the previous section (where derived arrows were dashed but now we use dashing for 3D-drawing). As in diagram Fig. 1(a), we again use numeric names for links to convey the order of operations, but now we write $i{:}p$ rather than $i{:}\mathsf{get}_p$ to ease reading (below we will see diagrams that have more such arrows). By the same reason, we omit numeric scripts near vertical and diagonal arrows (deltas), and label the deltas realizing natural transformations very briefly, e.g., e_S for the full name $\mathsf{get}(e,S)$– the latter will also be written in formulas as $\mathsf{get}_S(e)$ or $e.\mathsf{get}_S$. Two callouts near $1{:}p$ and E_S recall these notational tricks.

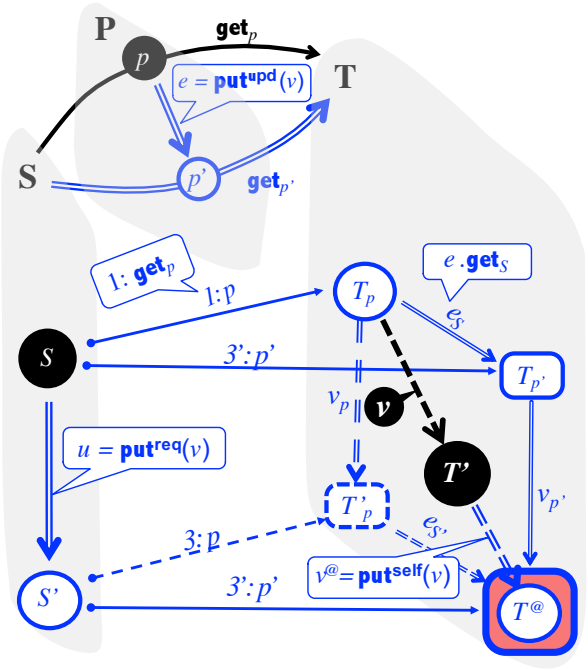


Figure 2: Arity schema of apa-lens operations and its derivatives:

The label of invoking operation put in the third step of the scenario is omitted, but instead, the equational definitions of deltas $e, u, v^{@}$ are written (note the three callouts near them). The diagram also shows four derived deltas forming the right back face of the prism: $v_p = \mathsf{get}_p(u)$, $v_{p'} = \mathsf{get}_{p'}(u)$, and two matching them deltas $e_S = \mathsf{get}_S(e)$, $e_{S'} = \mathsf{get}_{S'}(e)$, which together form a commutative square due to naturality of $\mathsf{get}(e)$ (note the common target of $e_{S'}$ and $v_{p'}$, i.e., $S'.\mathsf{get}_{p'}$, shown as a rounded square). Finally, the diagram also shows coincidence of this "square" object and the target $T^{@}$ of the amendment $v^{@}$ (the inner circle), which is a special requirement (a constraint) that does not hold automatically and to be specially postulated (the red filler in-between the square and the circle refers to this constraint). Thus, the arity schema of put operations includes the following Putget_0 equations:

(Putget)$_0$ $\mathsf{cod}(v.\mathsf{put}^{\mathsf{self}}_{p,S}) = \mathsf{cod}(v.\mathsf{put}^{\mathsf{req}}_{p,S}.\mathsf{get}_{p'}) = \mathsf{cod}(v.\mathsf{put}^{\mathsf{upd}}_{p,S}.\mathsf{get}_{S'})$

where the left equality is a constraint (red) while the right one is automatic (blue). The "co-discreteness" index 0 recalls that the constraint equates objects (nodes) rather than arrows. Equating arrows is the subject of a special Putget law stated below together with two other requirements to ala-lenses, which would allow MDE practitioners to call them "well-behaved".

Definition 3 (Well-behavedness) An ala-lens is called *well-behaved (wb)* if it satisfies the following three laws specified by equations to hold for all $p \in |\mathbf{P}|$, $S \in |\mathbf{S}|$ and $v\colon T_p \to T'$ where T_p denotes $\mathsf{get}_p(S)$:

(Stability) if $v = \mathsf{id}_{T_p}$ then all three propagated updates $e, u, v^{@}$ are identities:
$\mathsf{put}^{\mathsf{upd}}_{p,S}(\mathsf{id}_{T_p}) = \mathsf{id}_p$, $\mathsf{put}^{\mathsf{req}}_{p,S}(\mathsf{id}_{T_p}) = \mathsf{id}_S$, $\mathsf{put}^{\mathsf{self}}_{p,S}(\mathsf{id}_{T_p}) = \mathsf{id}_{T_p}$

(Putget) $\mathsf{get}_S(\mathsf{put}^{\mathsf{upd}}_{p,S}.v) = \mathsf{id}_{T_p}$ and $\mathsf{get}_{p'}(\mathsf{put}^{\mathsf{req}}_{p,S}.v) = v; v^{@}$, where $v^{@}$ denotes $\mathsf{put}^{\mathsf{self}}_{p,S}(v)$

(Hippocr) if $\mathsf{get}_p(u) = v$ for some $u\colon S \to S'$, then $v^{@} = \mathsf{id}_{T'}$

A lens is *stable* if it satisfies Stability, and an *SPg-lens* if it satisfies Stability and Putget. A lens is called *Hippocratic* if the (Hippocr) law holds.[5]

Remark 1 a) Stability says that the lens does nothing if nothing happens on the target side (no trigger–no action, hence, the name of the law)

b) Putget requires the goal of update propagation to be achieved after the propagation act is finished. It may seem a notational trick rather than a law as operation $\mathsf{put}^{\mathsf{req}}_{p,S}$ provides delta u, and functor $\mathsf{get}_{p'}$ sends it to a delta $\mathsf{get}_{p'}(u)$ targeting at some model $T^{@}$ anyway. However, in general, there are many possible deltas between models T' and $T^{@}$, whereas the Putget law states the existence of some special delta $v^{@}\colon T \to T^{@}$ such that equations (Putget) hold. Note also that Putget implies commutativity of the lower triangle in the right back face as the upper one is commutative by Putget.

c) Hippocraticness prohibits amending delta v if consistency can be achieved for v without amendment (hence, the name). Note that neither equality $\mathsf{put}^{\mathsf{upd}}_{p,S}(v) = p$ nor $\mathsf{put}^{\mathsf{req}}_{p,S}(v) = u$ are required: the lens can improve consistency by changing p and u but changing v is disallowed.

***Example* 1 (Identity lenses)** *Any skeletal category* $\mathbf{A}$ *gives rise to an ala-lens* $\mathit{id}_{\mathbf{A}}$ *with the following components. The source and target spaces are equal to* $\mathbf{A}$*, and the parameter space is* $\mathbf{1}$ *(a*

[5]The term Hippocratic and its derivatives were coined for Bx by Perdita Stevens [13] to refer to a family of requirements sharing a common intention of doing nothing if consistency is not violated, which can be seen as an application of a major medical principle "First, do no harm" usually attributed to the Hippocratic oath — see Wikipedia https://en.wikipedia.org/wiki/Hippocratic_Oath for details.

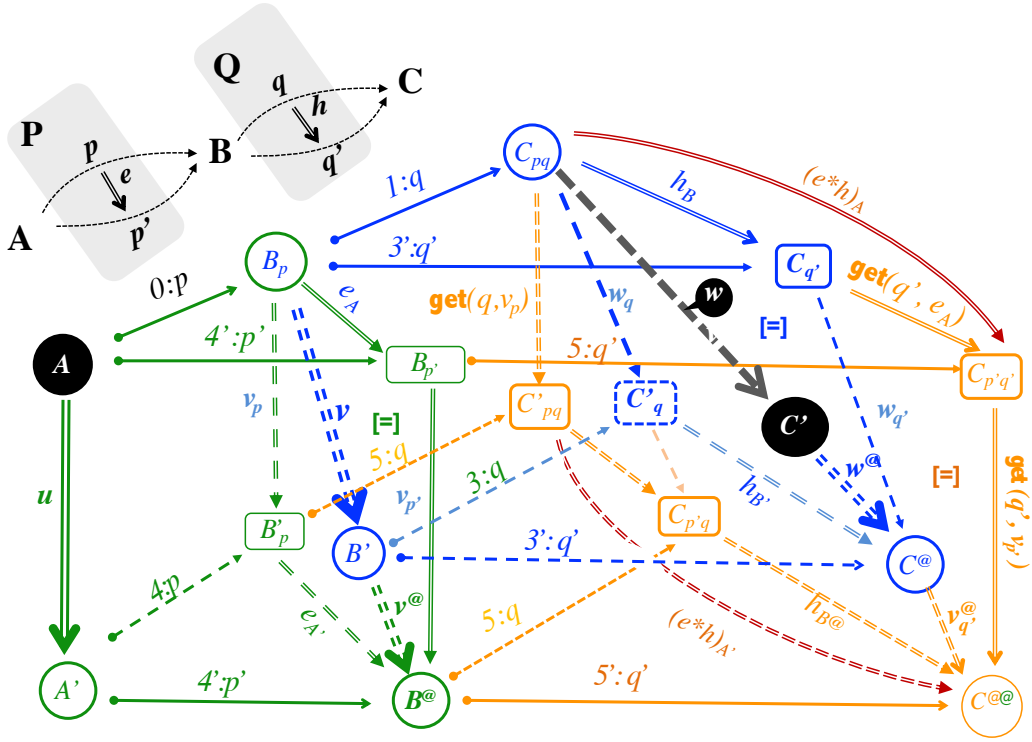


Figure 3: Sequential composition of apa-lenses

*category with a single object * and a single arrow* id_**). Functor* get_* *is the identity functor and all puts are identities. Obviously, this lens is wb.*

***Example* 2 (Iso-lenses)** *Let* $\iota\colon \mathbf{A} \to \mathbf{B}$ *be an isomorphism between model spaces. It gives rise to a wb ala-lens* $\ell(\iota)\colon \mathbf{A} \to \mathbf{B}$ *with* $\mathbf{P}^{\ell(\iota)} = \mathbf{1}$ *as follows. Given any* A *in* $\mathbf{A}$ *and* $v\colon \iota(A) \to B'$ *in* $\mathbf{B}$*, we define* $\mathsf{put}^{\ell(\iota).\mathsf{req}}_{*,A}(v) = \iota^{-1}(v)$ *and the two other put operations send* v *to identities.*

***Example* 3 (Bx lenses)** *Examples of SPg and wb aa-lenses modelling a Bx can be found in [4]: they all can be considered as ala-lenses with a trivial parameter space* $\mathbf{1}$*.*

4 Sequential and parallel composition of ala-lenses, and symmetric monoidal category *alaLens*

Construction 1 (Sequential composition of ala-lenses) Let $\mathcal{K}\colon \mathbf{A} \to \mathbf{B}$ and $\ell\colon \mathbf{B} \to \mathbf{C}$ be two ala-lenses with parameterized functors $\mathsf{get}^{\mathcal{K}}\colon \mathbf{P} \to [\mathbf{A}, \mathbf{B}]$ and $\mathsf{get}^{\ell}\colon \mathbf{Q} \to [\mathbf{B}, \mathbf{C}]$ resp. Their *composition* is the following ala-lens $\mathcal{K};\ell$.

Its parameter space is the product $\mathbf{P} \times \mathbf{Q}$, and the get-family is defined as follows. For any pair of parameters (p, q) (we will write pq), $\mathsf{get}^{\mathcal{K};\ell}_{pq} = \mathsf{get}^{\mathcal{K}}_{p}; \mathsf{get}^{\ell}_{q}\colon \mathbf{A} \to \mathbf{C}$. Given a pair of

parameter deltas, $e\colon p \to p'$ in $\mathbf{P}$ and $h\colon q \to q'$ in $\mathbf{Q}$, their $\mathsf{get}^{\mathcal{K};\ell}$–image is the Godement product of natural transformations, $\mathsf{get}^{\mathcal{K};\ell}(eh) = \mathsf{get}^{\mathcal{K}}(e) * \mathsf{get}^{\ell}(h)$.

Now we define $\mathcal{K};\ell$'s propagation operations puts. Let (A, pq, C_{pq}) with $A \in |\mathbf{A}|$, $pq \in |\mathbf{P}\times\mathbf{Q}|$, $A.\mathsf{get}^{\mathcal{K}}_{p}.\mathsf{get}^{\ell}_{q} = C_{pq} \in |\mathbf{C}|$ be a state of lens $\mathcal{K};\ell$, and $w\colon C_{pq} \to C'$ is a target update as shown in Fig. 3. For the first propagation step, we run lens ℓ as shown in Fig. 3 with the blue colour for derived elements: this is just an instantiation of the pattern of Fig. 2 with the source object being $B_p = \mathsf{get}_p(A)$ and parameter q. The results are deltas

$$h = \mathsf{put}^{\ell.\mathsf{upd}}_{q,B}(w)\colon q \to q', \quad v = \mathsf{put}^{\ell.\mathsf{req}}_{q,B}(w)\colon B_p \to B', \quad w^{@} = \mathsf{put}^{\ell\mathsf{self}}_{q,B}(w)\colon C' \to C^{@}. \tag{1}$$

Next we run lens $\mathcal{K}$ at state (p, A) and the target update v produced by lens ℓ; it is yet another instantiation of pattern in Fig. 2 (this time with the green colour for derived elements), which produces three deltas

$$e = \mathsf{put}^{\mathcal{K}.\mathsf{upd}}_{p,A}(v)\colon p \to p', \quad u = \mathsf{put}^{\mathcal{K}.\mathsf{req}}_{p,A}(v)\colon A \to A', \quad v^{@} = \mathsf{put}^{\mathcal{K}\mathsf{self}}_{p,A}(v)\colon B' \to B^{@}. \tag{2}$$

These data specify the green prism adjoint to the blue prism: the edge v of the latter is half of the right back face diagonal $B_pB^{@}$ of the former. In order to make an instance of the pattern in Fig. 2 for lens $\mathcal{K};\ell$, we need to complete the blue-green diagram by filling-in the empty spaces below the diagonal and above the diagonal with appropriate arrows. These filling-in arrows are provided by functors get^{ℓ} and $\mathsf{get}^{\mathcal{K}}$ and shown in orange (where we have chosen one of the two equivalent ways of forming the Godement product – note two curve brown arrows). In this way we obtain yet another instantiation of the pattern in Fig. 2 denoted by $\mathcal{K};\ell$:

$$\mathsf{put}^{(\mathcal{K};\ell)\mathsf{upd}}_{A,pq}(w) = (e, h), \quad \mathsf{put}^{(\mathcal{K};\ell)\mathsf{req}}_{A,pq}(w) = u, \quad \mathsf{put}^{(\mathcal{K};\ell)\mathsf{self}}_{A,pq}(w) = w^{(\mathcal{K};\ell).@} \tag{3}$$

where $w^{(\mathcal{K};\ell).@} = w^{@}; v^{@}_{q}$, and $v^{@}_{q'}$ denotes $v^{@}.\mathsf{get}_{q'}$. Thus, we built an ala-lens $\mathcal{K};\ell$, which satisfies equation Putget_0 by construction.

The following result is important for practical applications: it ensures that a composed synchronizer satisfies its requirements automatically as soon as its components do, which allows significant reducing of the integration testing (only connectivity is to be checked).

Definition 4 (Get and Put images of a lens) *Given a lens from space* $\mathbf{S}$ *to space* $\mathbf{T}$*, define*

$$\mathsf{Get}(\mathbf{S}) = \{u.\mathsf{get}_p : u \in \mathsf{Arr}\mathbf{S}, p \in |\mathbf{P}|\} \subset \mathsf{Arr}\mathbf{T}$$
$$\mathsf{Put}(\mathbf{T}) = \{\mathsf{put}_{p,S}.v : p \in |\mathbf{P}|, S \in |\mathbf{S}|, v\colon \mathsf{get}_p(S) \to \_ \in \mathsf{Arr}\mathbf{T}\} \subset \mathsf{Arr}\mathbf{S}$$

Two consecutive lenses $\mathcal{K}\colon \mathbf{A} \to \mathbf{B}$, $\ell\colon \mathbf{B} \to \mathbf{C}$ *are called* well-matched *if* $\mathsf{Get}^{\mathcal{K}}(\mathbf{A}) \supset \mathsf{Put}^{\ell}(\mathbf{C})$.

Theorem 1 (Seq. composition and lens laws) *Given are ala-lenses $\mathcal{k}\colon \mathbf{A} \to \mathbf{B}$, $\ell\colon \mathbf{B} \to \mathbf{C}$, and lens $\mathcal{k};\ell\colon \mathbf{A} \to \mathbf{C}$ is their seq. composition as defined above in Constr. 1. Then the following holds.*

(a) Lens $\mathcal{k};\ell$ is SPg as soon as lenses $\mathcal{k}$ and ℓ are such.

(b) If lenses $\mathcal{k}$ and ℓ are wb and additionally well-matched then lens $\mathcal{k};\ell$ is wb too.

A proof can be found in Appendix A

Construction 2 (Parallel composition of ala-lenses) Let $\ell_i\colon \mathbf{A}_i \to \mathbf{B}_i$, $i = 1, 2$ be two ala-lenses with parameter spaces $\mathbf{P}_i$. The lens $\ell_1 \| \ell_2\colon \mathbf{A}_1 \times \mathbf{A}_2 \to \mathbf{B}_1 \times \mathbf{B}_2$ is defined as follows. (We will denote a pair of element (x, y) by $x \| y$.) Parameter space $\ell_1 \| \ell_2.\mathbf{P} = \mathbf{P}_1 \times \mathbf{P}_2$. For any $p_1 \| p_2 \in \mathbf{P}_1 \times \mathbf{P}_2$, define $\mathsf{get}^{\ell_1\|\ell_2}_{p_1\|p_2} = \mathsf{get}^{\ell_1}_{p_1} \times \mathsf{get}^{\ell_2}_{p_2}$. Further, for any $S_1 \| S_2 \in \mathbf{A}_1 \times \mathbf{A}_2$ and delta $v_1 \| v_2\colon \mathsf{get}^{\ell_1\|\ell_2}_{p_1\|p_2}(S_1 \| S_2) \to T_1' \| T_2'$, we define componentwise

$$e = \mathsf{put}^{(\ell_1\|\ell_2)\mathsf{upd}}_{p_1\|p_2, S_1\|S_2}(v_1 \| v_2)\colon p_1 \| p_2 \to p_1' \| p_2' \text{ by setting } e = e_1 \| e_2 \text{ where } e_i = \mathsf{put}^{\ell_i}_{p_i, S_i}(v_i),\ i = 1, 2$$

and similarly for $\mathsf{put}^{(\ell_1\|\ell_2)\mathsf{req}}_{p_1\|p_2, S_1\|S_2}$ and $\mathsf{put}^{(\ell_1\|\ell_2)\mathsf{self}}_{p_1\|p_2, S_1\|S_2}$

The following result is obvious but important (as any compositionality result– see the remark about integration testing above).

Theorem 2 (Parallel composition and lens laws) *Lens $\ell_1 \| \ell_2$ is wb as soon as lenses ℓ_1 and ℓ_2 are such.*

Now our goal is to organize ala-lenses into an sm-category. To make seq. composition of ala-lenses associative, we need to consider them up to some equivalence (indeed, Cartesian product is not strictly associative).

Definition 5 (Ala-lens Equivalence) *Two parallel ala-lenses $\ell, \hat{\ell}\colon \mathbf{S} \to \mathbf{T}$ are called* equivalent *if their parameter spaces are isomorphic via a functor $\iota\colon \mathbf{P} \to \hat{\mathbf{P}}$ such that for any $S \in |\mathbf{S}|$, $p \in \mathbf{P}$ and $v\colon (S.\mathsf{get}_p) \to T'$ the following holds:*

$$S.\mathsf{get}_p = S.\widehat{\mathsf{get}}_{\iota(p)}, \quad \iota(\mathsf{put}^{\mathsf{upd}}_{p,S}(v)) = \widehat{\mathsf{put}}_{\iota(p),S}(v), \text{ and } \mathsf{put}^{\mathbf{x}}_{p,S}(v) = \widehat{\mathsf{put}}^{\mathbf{x}}_{S,\iota(p)}(v) \text{ for } \mathbf{x} \in \{\mathsf{req}, \mathsf{self}\}$$

Remark 2 *In general, we should require isomorphisms rather than equalities above, but isos become equalities for skeletal categories.*

Lemma 1 *Operations of lens' sequential and parallel composition are compatible with lens' equivalence. Hence, these operations are well-defined for equivalence classes.*

Below we will identify lenses with their equivalence classes by default.

Theorem 3 (Ala-lenses form an sm-category) *Operations of sequential and parallel composition of ala-lenses defined above allow building an sm-category* ***alaLens****, whose objects are model spaces (= skeletal categories) and arrows are (equivalence classes) of ala-lenses.*

Proof. It is easy to check that identity lenses $id_{\mathbf{A}}$ defined in Example 1 are the units of the seq. lens composition defined above. The proof of associativity is rather involved and is placed into Appendix B. Thus, $\boldsymbol{alaLens}$ is a category.

Next we define the monoidal structure. The monoidal product of objects is Cartesian product of categories (skeletality is preserved), and the monoidal product of arrows is lens' parallel composition defined above. The monoidal unit is the terminal category $\mathbf{1}$. Associators, left and right unitors, and braiding are iso-lenses generated by the respective isomorphism functors (Example 2). Moreover, it is easy to see that the iso-lens construction from Example 2 is actually a functor $\boldsymbol{isolens}\colon \boldsymbol{Cat}_{\mathsf{iso}} \to \boldsymbol{alaLens}$. Then as a) $\boldsymbol{Cat}$ is symmetric monoidal and fulfils all necessary monoidal equations, and b) $\boldsymbol{isolens}$ is a functor, these equations hold for the ala-lensimages of $\boldsymbol{Cat}_{\mathsf{iso}}$-arrows, and $\boldsymbol{alaLens}$ is symmetric monoidal too (cf. a similar proof in [9] with $(\boldsymbol{Set}, \times)$ instead of $(\boldsymbol{Cat}, \times)$.

5 Future work

5.1 The delta lens framework as such

Enrichment. Generalization of delta lenses for the enriched categorical setting, in which model spaces and get-mappings are $\mathcal{V}$-categories and $\mathcal{V}$-functors for some sm-category $\mathcal{V}$, and put-operations are correspondingly generalized too. If everything is done right, this would give us an sm-category $\boldsymbol{alaLens}(\mathcal{V})$. Three cases are especially important: $\mathcal{V} = [0, \infty]$ seems to be fundamental for ML (see discussion in Sect. 2.2), $\mathcal{V} = \boldsymbol{Set}$ is developed in the present paper, and $\mathcal{V} = \boldsymbol{Cat}$ is a long-time needed generalization of lenses for the bicategorical setting (see, e.g., [3] for the necessity to have 2-arrows for BX and [9, Sect.VII.C] argues for bicategories in ML). The notion of ala-lens should be functorial w.r.t. $\mathcal{V}$, i.e., be specifiable by a functor $\boldsymbol{alaLens}\colon \boldsymbol{smCat} \to \boldsymbol{smCat}$ that maps an sm-category $\mathcal{V}$ to the corresponding sm-category of $\mathcal{V}$-enriched ala-lenses.

Symmetric lens decomposition. Generalizing Fong and Johnson's result [8] from the codiscrete case to the delta lens case. It may happen that a symmetric learning lens is decomposable into a ternary span of asymmetric lenses, whose two legs are responsible for usual change propagation and the third leg provides learning (parameter changing). Then the delta version of the result would be a special case, when the original symmetric learning lens happens to be asymmetric, and hence the span should be binary.

Sequential lens composition as the Kleisli arrow composition. Find a suitable monad so that the associativity of seq. lens composition would be the associativity of substitution in the Kleisli category (cf. a remark on p.9 of [8]). Note also that in diagram Fig. 3, filling-in the orange part somewhat recalls the composition of Kleisli arrows.

Applications to BX. a) Explore applicability of the learning lens framework for model transformation by example work (see [11] for references). b) Check functoriality of common change propagation policies used in TGG-based BX, e.g., those in [1].

Applications to ML. a) Explore applicability of deltas (see Sect. 2.2) in the practical ML setting. b) Explore applicability of lens laws in the practical ML setting.

References

[1] A. Anjorin, S. Rose, F. Deckwerth, and A. Schürr. Efficient model synchronization with view triple graph grammars. In J. Cabot and J. Rubin, editors, *Modelling Foundations and Applications - 10th European Conference, ECMFA 2014, Held as Part of STAF 2014, York, UK, July 21-25, 2014. Proceedings*, volume 8569 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2014.

[2] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, and J. F. Terwilliger. Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations*, pages 260–283. Springer, 2009.

[3] Z. Diskin. Compositionality of update propagation: Lax putput. In R. Eramo and M. Johnson, editors, *Proceedings of the 6th International Workshop on Bidirectional Transformations co-located with The European Joint Conferences on Theory and Practice of Software, BX@ETAPS 2017, Uppsala, Sweden, April 29, 2017.*, volume 1827 of *CEUR Workshop Proceedings*, pages 74–89. CEUR-WS.org, 2017.

[4] Z. Diskin, H. König, and M. Lawford. Multiple model synchronization with multiary delta lenses. In A. Russo and A. Schürr, editors, *Fundamental Approaches to Software Engineering, 2018*, volume 10802 of *Lecture Notes in Computer Science*, pages 21–37. Springer, 2018. An extended version will appear in *Formal Aspects of Computing*, 2019 under the title "Multiple Model Synchronization with Multiary Delta Lenses with Amendment and K-Putput".

[5] Z. Diskin and U. Wolter. A Diagrammatic Logic for Object-Oriented Visual Modeling. *Electr. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008.

[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] H. Ehrig, K. Ehrig, U. Prange, and G. Taenzer. *Fundamentals of Algebraic Graph Transformation*. 2006.

[8] B. Fong and M. Johnson. Lenses and Learners, 2019. arXiv:1903.03671. To appear in Proc. BX'2019.

[9] B. Fong, D. Spivak, and R. Tuyéras. Backprop as Functor: A compositional perspective on supervised learning , 2018. arXiv:1711.10455. To appear in Proc. LICS'2019.

[10] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, and A. Schmitt. Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In J. Palsberg and M. Abadi, editors, *Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2005, Long Beach, California, USA, January 12-14, 2005*, pages 233–246. ACM, 2005.

[11] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, and M. Wimmer. Model transformation by-example: A survey of the first wave. In A. Düsterhöft, M. Klettke, and K. Schewe, editors, *Conceptual Modelling and Its Theoretical Foundations - Essays Dedicated to Bernhard Thalheim on the Occasion of His 60th Birthday*, volume 7260 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2012.

[12] M. Makkai. Generalized sketches as a framework for completeness theorems. *Journal of Pure and Applied Algebra*, 115:49–79, 179–212, 214–274, 1997.

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

A Sequential composition of ala-lenses and lens laws: Proof of Theorem 1 on page 14

Proof. a) Stability of $\mathcal{K};\ell$ is obvious. To prove Putget for $\mathcal{K};\ell$, we first note that as e_A is the identity due to Putget for $\mathcal{K}$, then its image is also the identity (functoriality of $\mathsf{get}_{q'}$) and hence $(e * h)_A$ is the identity as h_B is the identity due to Putget for ℓ. It remains to prove $u.\mathsf{get}^{\mathcal{K};\ell}_{p'q'} = w; w^{(\mathcal{K};\ell).@}$ We compute:

$$
\begin{aligned}
u.\mathsf{get}^{\mathcal{K};\ell}_{p'q'} &= u.\mathsf{get}^{\mathcal{K}}_{p'}.\mathsf{get}^{\ell}_{q'} && \text{by constr.}\\
&= (v; v^{@}).\mathsf{get}^{\ell}_{q'} && \text{Putget for } \mathcal{K}\\
&= (v.\mathsf{get}^{\ell}_{q'})\,;\,(v^2.\mathsf{get}^{\ell}_{q'}) && \text{functoriality}\\
&= (w; w^{@})\,;\, v^{@}_{q'} && \text{Putget for } \ell \mid \text{def. of } v^{@}_{q'}\\
&= w; (w^{@}; v^{@}_{q'}) && \text{associativity of ;}\\
&= w; w^{(\mathcal{K};\ell).@}
\end{aligned}
\tag{4}
$$

Proof of (b). For any given A, p, q in the respective spaces, and delta $w\colon A.\mathsf{get}^{\mathcal{K};\ell}_{pq} \to C'$ such that $u.\mathsf{get}^{\mathcal{K};\ell}_{pq} = w$ for some $u\colon A \to \_$, we need to prove $w^{(\mathcal{K};\ell).@} = \mathit{theidentity}_{C'}$. We have seen

that

$$w^{(k;\ell).@} = w^{\ell.@}\,;\, \mathsf{get}^{\ell}_{q'}(v^{k.@}) \quad \text{where } v = \mathsf{put}^{\ell.\mathsf{req}}_{B_p,q}.$$

We will prove that both components of ; are identities. The first one is identity due to $w = \mathsf{get}^{\ell}_{q}(u.\mathsf{get}^{k}_{p})$ and ℓ is Hippocratic. For the second component, $v^{k.@}$ is identity as i) $v = \mathsf{put}^{\ell.\mathsf{req}}_{B_p,q}$ and lenses are well-matched and ii) lens k is Hippocratic. Now the second component is identity as functor $\mathsf{get}^{\ell}_{q'}$ preserves identities.

B Proof: the sequential ala-lens composition is associative

Let $k\colon \mathbf{A} \to \mathbf{B}$, $\ell\colon \mathbf{B} \to \mathbf{C}$, $\mu\colon \mathbf{C} \to \mathbf{D}$ be three consecutive lenses with parameter spaces $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ resp. We will denote their components by an upper script, e.g., get^{k}_{p} or $\mathsf{put}^{\ell.\mathsf{upd}}_{q,B}$, and lens composition by concatenation: $k\ell$ is $k;\ell$ etc; $\mathsf{put}^{k\ell.\mathsf{upd}}_{p,A}$ denotes $\mathsf{put}^{(k\ell).\mathsf{upd}}_{p,A}$

We need to prove $(k\ell)\mu = k(\ell\mu)$. We easily have associativity for the get part of the construction: $(\mathbf{P}\times\mathbf{Q})\times\mathbf{R} \cong \mathbf{P}\times(\mathbf{Q}\times\mathbf{R})$ (to be identified for equivalence classes), and $(\mathsf{get}^{k}_{p};\mathsf{get}^{\ell}_{q});\mathsf{get}^{\mu}_{r} = \mathsf{get}^{k}_{p};(\mathsf{get}^{\ell}_{q};\mathsf{get}^{\mu}_{r})$, which means that $\mathsf{get}^{(k\ell)\mu}_{(pq)r} = \mathsf{get}^{k(\ell\mu)}_{p(qr)}$, where p, q, r are parameters (objects) from $|\mathbf{P}|, |\mathbf{Q}|, |\mathbf{R}|$ resp., and pairing is denoted by concatenation.

Associativity of puts is more involved. Suppose that we extended the diagram in Fig. 3 with lens μ data on the right, i.e., with a triangle prism, whose right face is a square $D_{pqr}D_{r'}D^{@}D'_{r}$ with diagonal $\omega;\omega^{@}\colon D_{pqr} \to D^{@}$ where $r \in \mathbf{R}$ is a parameter, $D_{pqr} = \mathsf{get}^{\mu}_{r}(C_{pq})$ and $\omega\colon D_{pqr} \to D'$ is an arbitrary delta to be propagated to $\mathbf{P}$ and $\mathbf{A}$, and reflected with amendment $\omega^{@} = \mathsf{put}^{\mu.\mathsf{self}}_{r,C_{pq}}(\omega)$. Below we will omit parameter subindexes near B and C.

We begin with term substitution in equations (1–3) in Constr. 1, which gives us equational definitions of all put operations (we use the function application notation $f.x$ as the most convenient):

$$\mathsf{put}^{k\ell.\mathsf{req}}_{pq,A}.w = (\mathsf{put}^{k.\mathsf{req}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.w)\colon A \to A', \tag{5}$$

$$\mathsf{put}^{k\ell.\mathsf{upd}}_{pq,A}.w = (\mathsf{put}^{k.\mathsf{upd}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.w)\colon p \to p' \parallel (\mathsf{put}^{\ell.\mathsf{upd}}_{q,B}.w)\colon q \to q' \tag{6}$$

$$\mathsf{put}^{k\ell.\mathsf{self}}_{pq,A}.w = (\mathsf{put}^{\ell.\mathsf{self}}_{q,B}.w)\,;\,\mathsf{get}^{\ell}_{q'}(\mathsf{put}^{k.\mathsf{self}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.w)\colon C' \to C^{@} \to C^{@@} \tag{7}$$

(note the interplay between different puts in (6) and (7), and also their "duality": (6) is a $\parallel$-tem while (7) is a ;-term).

Now we apply these definitions to the lens $(k\ell)\mu$ and substitute. Checking $\mathsf{put}^{(k\ell)\mu.\mathsf{req}}$ is straightforward similarly to associativity of gets, but we will present its inference to show how the notation works (recall that $\omega\colon D_{pqr} \to D'$ is an arbitrary delta to be propagated)

$$\begin{aligned}
\mathsf{put}^{(k\ell)\mu.\mathsf{req}}.\omega &= \mathsf{put}^{k\ell.\mathsf{req}}_{pq,A}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega \quad \text{by (5)}\\
&= \mathsf{put}^{k.\mathsf{req}}_{p,A}.(\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega) \quad \text{by (5)}\\
&= \mathsf{put}^{k.\mathsf{req}}_{p,A}.\mathsf{put}^{\ell\mu.\mathsf{req}}_{qr,B}.\omega \quad \text{by (5)}\\
&= \mathsf{put}^{k;(\ell\mu)}_{p,A}.\omega \quad \text{by (5)}
\end{aligned} \tag{8}$$

Computing of $\mathsf{put}^{(\kappa\ell);\mu.\mathsf{upd}}$ is more involved (below a pair (x, y) will be denoted as either xy or $x\|y$ depending on the context).

(9)

$$\begin{array}{rcl}
\mathsf{put}^{(\kappa\ell);\mu.\mathsf{upd}}_{(pq)r,A}.\omega & = & \left(\mathsf{put}^{\kappa\ell.\mathsf{upd}}_{pq,A}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega\colon\ p\|q \to p'\|q'\right) \ \|\ \left(\mathsf{put}^{\mu.\mathsf{upd}}_{r,C}.\omega\colon\ r \to r'\right) \quad \text{by (6)} \\
& = & \left(\mathsf{put}^{\kappa.\mathsf{upd}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega \parallel \mathsf{put}^{\ell.\mathsf{upd}}_{q,B}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}\omega\right) \| \mathsf{put}^{\mu.\mathsf{upd}}_{r,C}.\omega \quad \text{by (6)} \quad \| \text{ same} \\
& = & \mathsf{put}^{\kappa.\mathsf{upd}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega \parallel \left(\mathsf{put}^{\ell.\mathsf{upd}}_{q,B}.\mathsf{put}^{\mu.\mathsf{req}}_{r,C}\omega \| \mathsf{put}^{\mu.\mathsf{upd}}_{r,C}.\omega\right) \quad \text{by assoc. of } \| \\
& = & \mathsf{put}^{\kappa.\mathsf{upd}}_{p,A}.\mathsf{put}^{\ell\mu.\mathsf{req}}_{qr,B}.\omega \parallel \mathsf{put}^{\ell\mu.\mathsf{upd}}_{qr,B}.\omega \qquad \text{by (5)} \parallel \text{ by (6)} \\
& = & \mathsf{put}^{\kappa;(\ell\mu).\mathsf{upd}}_{p(qr),A}.\omega \quad \text{again by (6)}
\end{array}$$

Associativity of $\mathsf{put}^{(\kappa;\ell;\mu).\mathsf{self}}_{pqr,A}$ can be proved in a similar manner using associativity of ; (see (7)) rather than associativity of $\|$ (see (6)) used above. Below w stands for $\mathsf{put}^{\mu.\mathsf{req}}_{r,C}.\omega$

(10)

$$\begin{array}{rcl}
\mathsf{put}^{(\kappa\ell).\mu.\mathsf{self}}_{(pq)r,A}.\omega & = & (\mathsf{put}^{\mu.\mathsf{self}}_{r,C}.\omega)\ ;\ \mathsf{get}^{\mu}_{r'}\left(\mathsf{put}^{\kappa\ell.\mathsf{self}}_{p,A}.w\right) \quad \text{by (7)} \\
& = & (\mathsf{put}^{\mu.\mathsf{self}}_{r,C}.\omega)\ ;\ \mathsf{get}^{\mu}_{r'}\left((\mathsf{put}^{\ell.\mathsf{self}}_{q,B}.w)\ ;\ \mathsf{get}^{\ell}_{q'}(\mathsf{put}^{\kappa.\mathsf{self}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.w)\right) \quad \text{by (7)} \\
& = & \left((\mathsf{put}^{\mu.\mathsf{self}}_{r,C}.\omega)\ ;\ \mathsf{get}^{\mu}_{r'}(\mathsf{put}^{\ell.\mathsf{self}}_{q,B}.w)\right)\ ;\ \mathsf{get}^{\ell\mu}_{q'r'}(\mathsf{put}^{\kappa.\mathsf{self}}_{p,A}.\mathsf{put}^{\ell.\mathsf{req}}_{q,B}.w) \quad \text{by funct. of } \mathsf{get}^{\mu}_{r'} \text{ and assoc. of ;} \\
& = & (\mathsf{put}^{\ell\mu.\mathsf{self}}_{qr,B}.\omega)\ ;\ \mathsf{get}^{\ell\mu}_{q'r'}(\mathsf{put}^{\kappa.\mathsf{self}}_{p,A}.\mathsf{put}^{\ell\mu.\mathsf{req}}_{qr,B}.\omega) \quad \text{by def. of } w \text{ and (8) applied twice} \\
& = & \mathsf{put}^{\kappa(\ell\mu).\mathsf{self}}_{p(qr)}.\omega \quad \text{again by (7)}
\end{array}$$