Controllable and decomposable multidirectional synchronizations

Gábor Bergmann^{1,2}

Received: 25 August 2020 / Revised: 22 February 2021 / Accepted: 23 February 2021 / Published online: 14 April 2021
© The Author(s) 2021

Abstract

Studying large-scale collaborative systems engineering projects across teams with differing intellectual property clearances, or healthcare solutions where sensitive patient data needs to be partially shared, or similar multi-user information systems over databases, all boils down to a common mathematical framework. Updateable views (lenses) and more generally bidirectional transformations are abstractions to study the challenge of exchanging information between participants with different read access privileges. The view provided to each participant must be different due to access control or other limitations, yet also consistent in a certain sense, to enable collaboration towards common goals. A collaboration system must apply bidirectional synchronization to ensure that after a participant modifies their view, the views of other participants are updated so that they are consistent again. While bidirectional transformations (synchronizations) have been extensively studied, there are new challenges that are unique to the multidirectional case. If complex consistency constraints have to be maintained, synchronizations that work fine in isolation may not compose well. We demonstrate and characterize a failure mode of the emergent behaviour, where a consistency restoration mechanism undoes the work of other participants. On the other end of the spectrum, we study the case where synchronizations work especially well together: we characterize very well-behaved multidirectional transformations, a non-trivial generalization from the bidirectional case. For the former challenge, we introduce a novel concept of controllability, while for the latter one, we propose a novel formal notion of faithful decomposition. Additionally, the paper proposes several novel properties of multidirectional transformations.

Keywords Bidirectional transformation · Very well behaved · Collaborative engineering

1 Introduction

1.1 Motivation: collaborative engineering with access control

The engineering of complex systems frequently requires multiple teams formed of a diverse group of specialists in order to meet a large set of requirements including concerns such as safety or security. In industries with deep and interwoven supply chains, such as the contemporary consumer electronics and automotive industries, these teams

will be employed by multiple organizations, forming complex webs of partnerships [33]. Such *collaboration networks* may include OEM system integrators, external development contractors, component vendors, consultant firms providing security or safety audits, testing laboratories, and even governmental or intergovernmental authorities that provide supervision.

Systems engineering promotes a co-engineering workflow, such as hardware–system co-design, where artefacts of multiple engineering disciplines have to be kept consistent with each other in an ongoing feedback loop. Even within a single engineering discipline, quality concerns necessitate a process where design proposals and change requests are reviewed by engineers with different qualifications and/or familiarity with different components of the system under design. And even in cases where a clearly identified responsible owner (team or person) is assigned to an artefact, this owner must be ready to review and potentially accept change proposals from teams dealing with overlapping concerns. Therefore, the collaboration of such diverse teams of

Communicated by Perdita Stevens.

✉ Gábor Bergmann
bergmann@mit.bme.hu

¹ MTA-BME Lendület Cyber-Physical Systems Research Group, Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary

² IncQuery Labs, Budapest, Hungary

engineers cannot be described by a simple unidirectional information flow, but rather as mutual and *multidirectional* exchange of information, knowledge, design decisions and proposals, with *consistency enforcement* among its goals.

Of course, there are many cases where companies that are partners in one project may be direct competitors in another market. Such participants want to share only a limited amount of information as required during their collaboration, hiding from each other the rest of their trade secrets and other forms of intellectual property. Therefore, ensuring *confidentiality* and *security* must be a top goal [33] of industrial collaboration networks; this is technically realized as (read) *access control*.

There are well-known centralized engineering repository technologies, such as Subversion [2] for files, CDO [34] for EMF objects or Teamwork Cloud [22] for MagicDraw projects. However, they offer access control with very specific limitations, at least by default. More sophisticated access control solutions exist [9], but the interactions of multiple views are poorly studied, despite the potential for unintended side effects. We believe that a significant reason for the lack of satisfying solutions is that the underlying mathematics is not sufficiently understood at this point, making it difficult to engineer collaboration platforms that work both correctly and predictably, while still keeping a large degree of freedom in the kind of access control policies that can be applied.

There is an ongoing effort in the research community to develop a better understanding into problems similar to that of collaboration systems. This approach is discussed in the following paragraphs.

1.2 On bi- and multidirectional synchronizations

Bidirectional synchronizations (BX, also called bidirectional transformations) are ubiquitous in computing, from editable database views to data binding, from cloud-based file synchronization (see e.g. Fig. 1) to domain-specific or participant-specific views in collaborative model-driven engineering. An important BX construct, the *lens* is now a common pattern in functional programming for “editable” immutable data structures.

This prevalence of industrial examples of BX is a testament to its general applicability, as there are many use cases with the same basic challenge: Mutually exchanging information between participants, so that each participant has a different view on the same combined knowledge. The view provided to each participant is required to be different (e.g. in order to satisfy access control needs), yet at the same time also consistent in a certain sense, in order to enable collaboration. The central goal of BX is to ensure that after a participant modifies their view, the views of other participants are updated so that they are consistent again.

There are well-known languages and techniques (e.g. [6,7,27,29]) that provide concrete tools for realizing a BX. However, due to inherent complexity, it is difficult to determine when a BX does the “right thing”. Thus to formalize and reason on requirements against BX, the literature has developed a language of discourse [15,30] that abstracts away from the above technological contexts, and is equally applicable to any concrete data representation (relational databases, graphs, file systems, etc.).

Unfortunately, synchronizing more than two models raises additional challenges. The theory in the relatively juvenile field of *multidirectional synchronization* (MX) is not very well developed yet, as exemplified by the vast number of open problems identified at Dagstuhl Seminar 18491 [8] and also [25]. There are problems unique to MX, i.e. there are ways in which an MX composed of BX may fail even if all its constituent BXs are fine in isolation. Such interactions among the BXs may render an MX unsuitable for practical applications. This paper specifically focuses on such emergent properties of MX, as described in the next paragraphs.

1.3 Challenges and main contributions

In particular, we will demonstrate through a case study the phenomenon of “whack-a-mole” behaviour. In case of a data source equipped with multiple updateable views, it is possible for a valid combination of view states (that can co-occur under the right choice of source state) to be in practice unreachable (or very difficult to reach) using a sequence of view updates. Such a failure mode is not prevented by each updateable view being independently correct, i.e. able to force their view states onto the source model: setting one view to its desired state may move the other views out of alignment. We argue that most practical applications only benefit from MXs that are *controllable*, i.e. free from “whack-a-mole” behaviour. We formally define the class of controllable spans of lenses and characterize this class in terms of a simple-to-check condition. The argument is supported by a formal proof. A more detailed illustration of the whack-a-mole failure mode and a more detailed rundown of related contributions will come in Sect. 3.

On the other end of the spectrum, there are BX that work so well together that the emergent MX shows especially predictable behaviour. In such cases, it is very easy to answer questions such as whether the given states of two or more views can occur together, or how updating one view may affect another. While the class of *very well-behaved* (VWB) BX is well known, this paper is the first to develop the non-trivial generalization to the multidirectional case, by formulating properties of *decomposability*. We additionally completely characterize the class of VWB (wide) spans of lenses in terms of four simple-to-check conditions. The argument is supported by formal proofs. Example 1 gives a first

illustration of an MX that is VWB while also showcasing its highly predictable behaviour; a more detailed motivation for decomposable and VWB MX and a more detailed rundown of related contributions will come in Sect. 4.

Example 1 Figure 1 illustrates an MX with very predictable behaviour: a central server S hosting 5 files and 3 clients each having a local replica of a few of them.

It is easy to see, just by considering which files are available in which views, that (i) updates to view V_C will never influence V_B and that (ii) V_B can always be computed from V_A . It is easy to determine, without consulting source model S , (iii) whether a given update to V_C will be visible on V_A and (iv) whether a given state of V_A is compatible with a given state of V_C (i.e. there is a source state simultaneously consistent with them). Also, (v) once such a pair of consistent view states is known, we can update an initial source state with them in any order and yield the same end state.

1.4 Structure of the paper

Presenting the challenges and contributions in more detail requires familiarity with existing concepts and some well-known results. Section 2 establishes the terminology we will use regarding models, consistency among models, BX/MX to maintain such consistency, and a few other topics, while also giving a short summary of relevant existing results in the field, none of which are novel contribution of this paper.

The fundamentals being in place, Sect. 3 will present a novel challenge related to the “whack-a-mole” failure mode of MX, as well as a quick preview of contributions that the paper will make (mostly in Sect. 6) in relation to this challenge.

Similarly, Sect. 4 will take a closer look at challenges associated with the behaviour of MX being difficult to predict; as well as a quick preview of contributions that the paper will make (primarily in Sect. 8) in understanding the case with very predictable behaviour as the class of VWB MX.

In order to support the results in Sects. 6 and 8, the paper has to introduce and examine novel properties of MX in Sect. 5. Afterwards, Sect. 6 is ready to present the results regarding *controllability*, and discuss their wider implications.

Furthermore, in order to be able to state the main result in Sect. 8, additional novel terminology has to be introduced by Sect. 7 regarding the *decomposability* of models and MX. This allows Sect. 8 to formally state, but not yet prove, the main theorem characterizing VWB spans of lenses, and to discuss its implications.

The main proof requires Sect. 9 to introduce novel mathematical tools and several important lemmas regarding operations performed on lenses. This enables Sect. 10 to complete the proof.

Section 11 discusses related research by other authors, and carefully outlines the novelty of results in this paper. Finally, Sect. 12 summarizes the contributions and their expected impact, and also gives pointers for future research.

While not required for the main arguments of the paper, the appendices include a few additional remarks that may allow the reader to better understand the relationship between some of the novel concepts presented in the paper. First, Appendix A discusses whether a certain theoretical non-existence result can be worked around in practice. Among three closely related novel properties of MX, the main body of the paper only introduced the one that plays a direct role in addressing the challenges; the other two are to be found in Appendix B. A definition that was formerly only given for a special case is treated in more generality in Appendix C. A nice duality lemma is included in Appendix D to show how the novel concepts introduced in this paper are deeply connected.

2 Theoretical foundations

2.1 Models and consistency

2.1.1 Models and model states

In this paper, “model” will refer to the identity of an artefact or knowledge base, irrespective of its extension (current contents/state). We will denote by $m::M$ to signify that m is a concrete model state of model M . To clarify, M might be the name of a file on a collaboration server, while m may be a concrete sequence of bits that may form the content of the file at a certain point in time. If a database schema or metamodel is associated with M , any $m::M$ is expected to conform to it. We denote by $\text{St}M = \{m \mid m::M\}$ the set of all potential model states that M may have; elsewhere in the literature, this notion is usually called the *domain* or *model space*.

Example 2 In the file synchronization example of Fig. 1, we will refer to the contents of the server as source model S , and the filtered views available at the devices of each user will be called view models V_A , V_B , and V_C . Each source model state $s::S$ is a quintuplet of five byte streams, one for each of the files 1, 2, 3, 4, and 5. Similarly, e.g. states $a::V_A$ are triples formed from the actual contents of files 1, 2, and 3. An example state of V_A at a given point in time could be $a = \langle \text{“one”}, \text{“two”}, \text{“three”} \rangle$. If $\mathbb{F}$ denotes the set of all possible concrete file contents, then the model space of view model V_A is the set of all possible file content triplets $\text{St}V_A = \mathbb{F} \times \mathbb{F} \times \mathbb{F} = \mathbb{F}^3$, while $\text{St}V_B = \text{St}V_C = \mathbb{F}^2$ and $\text{St}S = \mathbb{F}^5$.

Fig. 1 File synchronization as a very well-behaved span of lenses. The source model S and views V_A , V_B , V_C are kept consistent by lenses A , B , C , each providing an updateable view on a part of S

2.1.2 Model parts

Similarly to the formal framework in [32], a *part* of a model will just mean an arbitrary function that can be evaluated on the states of that model; in other words, a *view*. It may or may not correspond to a well-delineated physical component (e.g. field of an object, file in a file system).

Example 3 In the file synchronization example of Fig. 1, we can interpret “file 2” as a model part of the view model V_A , defined by the function $f_2 : \text{St}V_A \rightarrow \mathbb{F}$ defined as $\langle x, y, z \rangle \mapsto y$.

2.1.3 Consistency among models

We will be interested in whether two or more models are in states that are compatible with each other. We expect that normally, all models have states that are consistent with each other, i.e. all *consistency constraints* are satisfied. If changes to one model violate this notion of consistency, then the other model(s) have to be adjusted so that they are consistent again; this consistency restoration task will be discussed in Sects. 2.2 and 2.3, but here we introduce the standard formal treatment of the notion of consistency.

Definition 1 (*Consistency relation (constraint)*) A consistency constraint or consistency relation $R(M_1, M_2, \dots)$ among models $M_1, M_2, \dots$ is a relation over the state spaces of these models, i.e. a set of tuples of model states that are considered consistent with each other: $R \subseteq \text{St}M_1 \times \text{St}M_2 \times \dots$

Given a constraint R , it may be practically convenient to replace it with the conjunction of multiple simpler subconstraints $R = R_1 \cap R_2 \cap \dots \cap R_k$. Furthermore, the question whether a multimodel tuple satisfies a consistency constraint R (or one of the subconstraints R_i introduced above) can often be determined based on a reduced subset of the models; the other models are irrelevant. In these cases, instead of the original form $R(M_1, M_2, \dots)$ or (R_i) , we may consider a simpler consistency constraint $R'(M_{i_1}, M_{i_2}, \dots, M_{i_k})$ that connects fewer models. The special case of a unary consistency (sub)-constraint, which may come up during decompositions such as these, can be interpreted as a reduction in the state space of the affected model. Such simplifications of constraints have been studied extensively in [32]; specifically

consistency constraints that can be replaced entirely by (at most) binary constraints are called *binary definable*.

2.2 Bidirectional synchronization (BX)

Next, we discuss the fundamentals of bidirectional synchronizations and transformations. Here, we adopt a language of discussion frequently used in the literature of BX [10,30] that captures bidirectional synchronization between models in a generic way, without formalizing the actual contents of the model. This way, results are equally applicable to any concrete model representation (relational databases, graphs, file systems, etc.). This picture is mainly useful for discussing whether a transformation is behaving in the expected way, in contrast to other perspectives (e.g. TGG [29], Relational lenses [7], Boomerang [6], QVT [27]) that aim at giving concrete tools for realizing a BX.

2.2.1 Basics of bidirectional synchronization

First, as a standard notion in this field of study, we introduce the notion of *bidirectional synchronization (BX)* as a kind of transformation that enforces a consistency relation among two models, and propagates information from either model to the other in order to restore this consistency. The definition largely follows [30], with the exception that we will require the so-called well-behaved property for all BXs in the paper, and have thus integrated it into the definition of BX.

Definition 2 (*Bidirectional synchronization*) For models X, Y and a binary consistency relation $R(X, Y)$ on them, a (well-behaved) bidirectional synchronization transformation $\langle \overrightarrow{R}, \overleftarrow{R} \rangle : \overleftrightarrow{R}(X, Y)$ is a pair of consistency restoring functions (or *restorers*),

$$\begin{aligned} & - \overrightarrow{R} : \text{St}X \times \text{St}Y \rightarrow \text{St}Y \text{ and} \\ & - \overleftarrow{R} : \text{St}X \times \text{St}Y \rightarrow \text{St}X, \end{aligned}$$

such that the following two properties are satisfied for each $x :: X$ and $y :: Y$:

Correctness: $R(x, \overrightarrow{R}(x, y))$ and $R(\overleftarrow{R}(x, y), y)$, i.e. applying the restorer in one direction will ensure that the updated model is now R -consistent with the unchanged one;

Hippocraticness: $R(x, y)$ implies $\vec{R}(x, y) = y$ and $x = \overleftarrow{R}(x, y)$, i.e. applying the restorer to a pair of models that are already consistent is a no-op (does not change the state).

Remark 1 The definitions given above follow the so-called *state-based* tradition in the field of BX. There exist more powerful *delta-based* BX formalizations [12,13], which process and propagate changes of models, in contrast to the state-based approach that only considers the old and new states of models. For the sake of simplicity, we restrict ourselves to the state-based approach in the current paper; as it is a special case of the delta-based approach, all challenges and failure modes investigated in the paper can be reproduced with delta-based lenses as well.

Remark 2 There also exist multiple traditions of considering consistency restorers that can modify both models. In one kind of approach, there is no preselected direction of information propagation [35], and both models can be updated to bring them into consistency. Our framework will be able to reproduce something similar, as discussed later in Remark 19. In the approach of *amendment lenses* [11], there is a clearly identified model most recently updated by the user, but there is still a way for the consistency restorer mechanism to *amend* user-proposed changes to that model. These are primarily delta-based approaches, so reservation of Remark 1 still apply.

The (*asymmetric*) *lens* is an important special kind of BX (study of which actually began [15] before the generic case), characterized by the property that one of the models is wholly determined by the other. In other words, one of the models is an updateable view of the other. In this case, the consistency relation is *functional*, with exactly one state of the view model being consistent with any state of the source model. The consistency restoring function from source to view (called GET) does not need the old state of the view model as its argument, and thus will, in fact, coincide with this functional relation. There may still be degrees of freedom in implementing the other consistency restorer (PUT) though.

Definition 3 (Lens (asymmetric)) For models S, V a (well-behaved asymmetric) lens $\langle \text{GET}, \text{PUT} \rangle : S \rightsquigarrow V$ is a pair of consistency restorer functions,

- $\text{GET} : \text{St}S \rightarrow \text{St}V$ and
- $\text{PUT} : \text{St}S \times \text{St}V \rightarrow \text{St}S$,

such that the following two properties are satisfied for each $s \in \text{St}S$ and $v \in \text{St}V$:

- PutGet: $\text{GET}(\text{PUT}(s, v)) = v$, i.e. updating the source model according to the view will achieve consistency;
- GetPut: $\text{PUT}(s, \text{GET}(s)) = s$, i.e. updating the source model according to the view state with which it is already consistent is a no-op.

This notion of lens is, indeed, proved [30] to be a special case of BX.

Example 4 In the file synchronization example of Fig. 1, we have three lenses $A : S \rightsquigarrow V_A$, $B : S \rightsquigarrow V_B$, $C : S \rightsquigarrow V_C$ that extract given files from the source model into a view model, and upon a view update, partially overwrite the source model. For instance, $A = \langle \text{GET}_A, \text{PUT}_A \rangle$ with $\text{GET}_A : \text{St}S \rightarrow \text{St}V_A$ definable as $\langle x, y, z, v, w \rangle \mapsto \langle x, y, z \rangle$ and $\text{PUT}_A : \text{St}S \times \text{St}V_A \rightarrow \text{St}S$ definable as $\langle x, y, z, v, w \rangle, \langle x', y', z' \rangle \mapsto \langle x', y', z', v, w \rangle$.

2.2.2 A practical notation

Here, we call the attention of the reader to an unusual notation that will be used throughout the paper. In order to simplify formulae involving chained applications of the above restorer functions, we will depart from the usual denotation of function application to write $\text{GET}(s)$ as $s.\text{GET}$ and $\text{PUT}(s, v)$ as $s.\text{PUT}(v)$. Remember: it is always the source state that goes before the dot.

With this notation, the PutGet law can be rendered as:

$$s.\text{PUT}(v).\text{GET} = v$$

The GetPut law becomes:

$$s.\text{PUT}(s.\text{GET}) = s$$

The same notational shortcut also allows us to discuss partial applications of the PUT function: $s.\text{PUT}$ is a unary function on view model states, while $\text{PUT}(v)$ is a unary function on source states.

2.2.3 The degenerate case

Note that the existence of a bidirectional transformation does not imply that R must be a bijection between model states of A and B . A lens can be understood as a special case where one of the models only contains information that is also found in the other one, while a bijective consistency relation (with the obvious restorers) is a lens in both directions. As pointed out in [30], “bijective transformations are the exception rather than the rule: the fact that one model contains information not represented in the other is part of the reason for having separate models”. Nevertheless, we will make extensive use

of a very special form of bijective consistency in the sequel, for which the following definition introduces (non-standard) terminology.

Definition 4 (*Mirror and replicator*) For two distinct models X, Y with the same state space $U = \text{St}X = \text{St}Y$, if their consistency relation is $R(X, Y) = \{x, x \mid x \in U\}$ simply expressing the equality of X and Y , we say that this consistency constraint is a mirror constraint. The replicator $X \rightleftharpoons Y = \langle \overrightarrow{R}, \overleftarrow{R} \rangle : \overrightarrow{R}(X, Y)$ is the (only) BX that maintains a mirror consistency with the obvious associated consistency restorers:

- $\overrightarrow{R}(x, y) = x$ and symmetrically
- $\overleftarrow{R}(x, y) = y$.

It immediately follows from the definitions that a replicator is correct and Hippocratic, thus a BX. Furthermore, it is obviously a lens as well (in either direction), with $x.\text{GET} := x$ and $x.\text{PUT}(y) = y$.

Example 5 In the file synchronization example of Fig. 1, none of the lens are replicators, since each view state is consistent with more than one source state. However, were we to extend the set-up with a fourth view model, a backup server, onto which a lens $D : S \rightsquigarrow V_D$ would extract all five files from the main server, then D would be a replicator, since it would enforce a one-to-one correspondence between $\text{St}S$ and $\text{St}V_D$.

Remark 3 Not all bijective consistency relations are mirrors (and thus the BX that maintains such is not a replicator). For instance, with common state space $U = \mathbb{R}^+$, the consistency relation $\{x, y \mid y = 1/x \wedge x \in U\}$ is bijective, but not an equality relation. We make this important distinction, as we will later use the property that transitive composition of such equality relations is still an equality relation, whereas *cyclic* natural join compositions of bijective relations may end up unsatisfiable.

2.2.4 Source models in 2D

A notion of *backward equivalence classes* was introduced in [31]: Two source states of a lens A are in the same equivalence class if they react to PUT_A exactly the same way, in other words, if they only differ in ways that can be overwritten via A .

Definition 5 (*Backwards equivalence classes of a lens*) For lens $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$, we define the function $\text{BackEq}_A : \text{St}S \rightarrow 2^{\text{St}S}$ as $s \mapsto \{s' \mid \forall a :: V_A : s.\text{PUT}_A(a) = s'.\text{PUT}_A(a)\}$.

Thus for any lens A , we have a function BackEq_A interpreted on the source state space, that takes each source state

to its backward equivalence class. As introduced in Sect. 2.1, such a function can be regarded as a model part or view. The lens A , of course, provides another view on S . A key insight of [31] is that these two views together uniquely identify the source state: the GetPut law implies that no two source states can agree in both GET_A and BackEq_A .

Based on this observation, [31] proposed a two-dimensional visualization of the source state space, plotting each source state according to these two model parts. Fig. 2a depicts the source state space $\text{St}S$ in terms of parts GET_A and BackEq_A . (For reasons that will be made clear in Sect. 9.2.1, we label latter axis as $\overline{A}$.) Since these two views jointly determine a source state, each point in the yellow area corresponds to at most one source state. We will refer to points that do not correspond to any source state as *A-gaps*. Note that for purposes of this visualization, we require an arbitrary total order of both the view states and of the backwards equivalence sets, so that they can be fit on an axis.

2.2.5 History ignorant and very well-behaved BX

Now we introduce an important class of BX with particularly nice properties: with a *history ignorant* synchronization, executing a sequence of consistency restorations (propagations) all in the same direction is equivalent to just executing the last one.

Definition 6 (*History ignorant BX*) A bidirectional synchronization $\langle \overrightarrow{R}, \overleftarrow{R} \rangle : \overrightarrow{R}(X, Y)$ is history ignorant if the following two conditions hold:

- $\overrightarrow{R}(x, \overrightarrow{R}(x', y)) = \overrightarrow{R}(x, y)$, and symmetrically
- $\overleftarrow{R}(\overleftarrow{R}(x, y'), y) = \overleftarrow{R}(x, y)$

We consider lens $\langle \text{GET}, \text{PUT} \rangle : S \rightsquigarrow V$ history ignorant iff it is history ignorant as a BX; this can be rephrased as the *PutPut* law holding $\forall s :: S, v, v' :: V$:

$$s.\text{PUT}(v).\text{PUT}(v') = s.\text{PUT}(v')$$

Example 6 In the file synchronization example of Fig. 1, all three lenses are history ignorant. If, for instance, the user in control of View A repeatedly updates one or more of files 1, 2, and 3, the end result on the server (and on the views at all users) will be the same as if the same user performed a single update, overwriting each file with the latest content assigned to it.

If a BX is shown to be history ignorant, its behaviour becomes particularly easy to reason about. Informally, each of the two related models have a model part that is *shared* with the other, and a *complement* model part that is not, such that the consistency relation is satisfied by a pair of model

Fig. 2 Visualization of source state space S according to a decomposition into A and $\bar{A}$

states exactly if this shared part is the same in both, and consistency restoration merely copies the shared part over from one model to the other, without touching either complement. We remind the reader that in our terminology, “model part” is not required to be a syntactical feature of the original model in some canonical representation (such as algebraic data type); it is rather computed arbitrarily by a function. This can formally expressed as below:

Definition 7 (*Very well-behaved (VWB) BX*) The bidirectional synchronization $\langle \vec{R}, \overleftarrow{R} \rangle : \overleftarrow{R}(X, Y)$ is called *constant complement* or very well behaved, VWB for short, if there exist bijections $f_X : X_C \times Z \leftrightarrow \text{St}X$, $f_Y : Z \times Y_C \leftrightarrow \text{St}Y$ such that:

- $R(f_X(x_C, z_x), f_Y(z_y, y_C)) \Leftrightarrow z_x = z_y$,
- $\vec{R}(f_X(x'_C, z'_x), f_Y(z_y, y_C)) = f_Y(z'_x, y_C)$, and symmetrically
- $\overleftarrow{R}(f_X(x_C, z_x), f_Y(z'_y, y'_C)) = f_X(x_C, z'_y)$

Intuitively, $\langle \vec{R}, \overleftarrow{R} \rangle : \overleftarrow{R}(X, Y)$ is VWB if the models X, Y can be decomposed in a way that turns the BX into a replicator; we will make this notion formal in Sect. 7.

The above definition, when applied to lenses, gives the following form: the lens $\langle \text{GET}, \text{PUT} \rangle : S \rightsquigarrow V$ is VWB if there is a bijection $f_S : S_C \times \text{St}V \leftrightarrow \text{St}S$ such that:

- $f_S(s_C, v).\text{GET} = v$, and
- $f_S(s_C, v).\text{PUT}(v') = f_S(s_C, v')$

Example 7 In the file synchronization example of Fig. 1, all three lenses are VWB. For instance, from the point of view of lens A , the source model can be split into a shared part (files 1, 2, and 3) and a complement (files 4 and 5). View V_A is in synch (i.e. consistent) with the server exactly if the content of files 1, 2, and 3 are the same at both locations.

Synchronization in either direction simply involves copying over the content of all three shared files.

The set of BXs that have the VWB property (as defined above) have been completely characterized in the literature [31]:

Proposition 1 (History ignorance and VWB) *A BX is VWB (constant complement) if and only if it is history ignorant.*

To better explain how these two concepts coincide, Fig. 2b illustrates VWB lenses and their relationship with PUT operations. First, it is known [31] that a lens A is history ignorant if and only if there are no A -gaps; there is exactly one valid source state for each pair of valid view states and backward equivalence classes.

A further known consequence is that backward equivalence classes will play the role of the “complement” model part: the S_C value associated with a source state s will simply be $\text{BackEq}_A(s)$. The constant complement nature of a VWB lens implies that $\text{PUT}_A(a)$ will be *ineffectual* on the complement, only replacing the part of the source model shared with the view; so, it will visualize as a horizontal movement to coordinate a . This immediately explains how a sequence of PUT_A operations can be simplified to just the last operation, hence history ignorance.

This notion of ineffectual update will come up a few times in the paper, so it is best to extract it as a stand-alone definition (generalizing the notion of ineffectual lens in [17]):

Definition 8 (Ineffectual update) The update $\text{PUT}_A(a)$ is ineffectual on model part f if, for each source state s , it holds that $f(s.\text{PUT}_A(a)) = f(s)$. An entire lens A is ineffectual on f if its PUT_A component is ineffectual for every view state.

In the history ignorant case, we can also construct an operation $\text{PUT}_{\bar{A}}$ that is “orthogonal” to the lens A , overwriting

only the complement; its applications are depicted as vertical arrows. We will see in Sect. 9.2.1 how $\text{PUT}_{\bar{A}}$ is actually the PUT component of a separate lens $\bar{A}$, explaining the choice of notation.

Remark 4 Neither the language we use here nor the definition of VWB given above are standard. Other sources have used the term “very well behaved” to simply mean history ignorant [15], or discuss it using different concepts such as equivalence sets [31]. This is of little import, as VWB and history ignorance have been shown to be equivalent in case of BXs; we insist on giving separate definitions only because the two concepts will generalize differently for multidirectional synchronizations.

2.3 Multidirectional synchronization (MX)

Synchronizing more than two models raises additional challenges; the theory in this relatively juvenile field is not very well developed yet, as exemplified by the vast number of open problems identified at Dagstuhl Seminar 18491 [8].

Below we briefly review the current state of the art in two alternative perspectives: one that focuses on atomic synchronizations that connect more than two models and one that discusses networks of multiple connected synchronizations.

2.3.1 Low-level picture: multiary lenses

The state of the art includes a low-level formalization of the *multiary lens* [11], which is a single synchronization “hyper-edge” connecting multiple models (the *feet*), propagating an update of any *single* foot model to all affected models. The same paper also contains an alternative description of such a multiary lens in terms of a (wide) *span* of ordinary, binary (BX) lenses: a newly introduced central model holds the union of knowledge in all feet, and each foot is obtained from it via a simple binary lens. Below we include a simplified form of their definition. Additional concerns of [11], such as delta-based propagation or update reflection (amendment), will not be discussed here.

Definition 9 (*Span of lenses (wide)*) For the central source model S and foot models/views $V_1, V_2, \dots$, a wide span $\langle A_1, A_2, \dots \rangle : S \rightsquigarrow (V_1, V_2, \dots)$ is a collection of lenses with $A_i : S \rightsquigarrow V_i = \langle \text{GET}_i, \text{PUT}_i \rangle$. The lenses jointly act as a multiary synchronization that propagates an update of foot i both to S using PUT_i and to all other feet $j \neq i$ using a composition of PUT_i and GET_j .

Example 8 In the file synchronization example of Fig. 1, the lenses form span $\langle A, B, C \rangle : S \rightsquigarrow (V_A, V_B, V_C)$. If, for example, file 2 is overwritten at view model V_A , an application of PUT_A will propagate it to source model S , after which GET_B can update V_B accordingly.

Assume lenses A and B form a span; Fig. 2c illustrates source states in the usual plot by A and $\bar{A}$, grouped together based on their GET_B value. Associating exactly one B -value to each state, GET_B effectively partitions source state space $\text{St}S$ into “ B -blobs”. (Such blobs are visualized here as being contiguous, but this is a matter of how the view states and backwards equivalence classes are arbitrarily ordered for purposes of the visualization.) In case B is also history ignorant, PUT_B will establish a bijection between the individual states of each pair of such blobs (based on BackEq_B -congruence), demonstrating how such blobs are “of the same size”.

Some recent research [24] focuses on how such multiary lenses may be composed to form an entire network of synchronizers.

The categorical dual of the above is a much less general, but in practice also very important concept. A *co-span* of lenses [23] consists of two or more lenses each having their own source model, but sharing a single central view model. This is a schema for secure peer-to-peer data exchange in a collaborative environment with access control: both participants have their secrets, but there is a shared common subset of their knowledge on which they can collaborate.

2.3.2 High-level picture: BX in the large

The state of the art also provides a high-level picture [32] of the emergent behaviour of a *network of models* related by several simple local synchronizations. (Such local synchronizations are either binary/BX, or can be viewed as a span of BX.) Among other results, [32] investigates whether, after updates to *one or more* models, it can be guaranteed that there exists a sequence of invocations of individual consistency restorers of the local BXs that result in the entire network of models becoming consistent and also whether the outcome is unique (deterministic).

It is then a separate problem, discussed e.g. in [8,28], how to algorithmically find or manually specify such a *synchronization policy* (also referred to as *coordination mechanism* or *execution strategy*) that tells us which of the constituent synchronizers to invoke and in what order. Unfortunately, positive guarantees regarding these questions are restricted [32] to the case where the network is either acyclic (there are no “loops” among local synchronizations), or cyclic in a very limited sense.

In the context of this paper, however, a certain property called *non-interference* introduced in [32] (also called *independence* in [17]) is more important than the results themselves.

Definition 10 (*Non-interference*) The two bidirectional synchronizations $\langle \vec{R}, \overleftarrow{R} \rangle : \vec{R}(X, Z)$ and $\langle \vec{S}, \overleftarrow{S} \rangle : \vec{S}(Y, Z)$ are non-interfering at Z if for all $x::X, y::Y, z::Z$ we have $\vec{S}(y, \vec{R}(x, z)) = \vec{R}(x, \vec{S}(y, z))$.

2.4 Difunctional relations

We will also use an important notion [23] from the theory of mathematical (heterogeneous) relations. Informally, this property states that there is a clearly identified extent of *shared information* between two sets. In the formal sense, the property will be explored in multiple mathematically equivalent forms; the first definition is given below following [23] and is illustrated in Fig. 3.

Definition 11 (*ZX property*) A relation $R \subseteq X \times Y$ between sets X and Y has the ZX property if, for each $x, x' \in X$ and $y, y' \in Y$ with $\langle x, y \rangle \in R \wedge \langle x', y \rangle \in R \wedge \langle x', y' \rangle \in R$, the fourth combination $\langle x, y' \rangle \in R$ also holds.

Figure 3a shows members of set X in the left column, and members of set Y in the right column; existing tuples in the relation R are depicted with solid brown lines, while the dashed blue line completing the “Z”-shape indicates where another tuple is inferred by the ZX property. Figure 3b gives an alternative perspective, where tuples in R are plotted as brown boxes, with the X and Y values scattered along each axis; the blue dashed box indicates where another tuple can be inferred, as the ZX property enforces a “rectangular” layout.

In order to motivate this property, let us consider the case of a consistency relation R between models M_X, M_Y , with $\text{St}M_X = X, \text{St}M_Y = Y$. Intuitively, a violation of the ZX property in the consistency relation would imply that one model does not “know” how much information the other model “knows” about it. Concretely, if ZX is violated by $\langle x, y' \rangle \in R$ missing (despite $\langle x, y \rangle \in R \wedge \langle x', y \rangle \in R \wedge \langle x', y' \rangle \in R$), then (1) in a consistent state where M_X is in x , there is enough information in M_X to know that the current state of M_Y is not y' , (2) whereas in state $x'::M_X$, there is not enough information in M_X to know whether M_Y is in y or y' , and (3) $y::M_Y$ is consistent with both x, x' , so there is not enough information in M_Y to know how much M_X reveals about M_Y .

An alternative form of the above property follows:

Definition 12 (*Difunctional relation*) A relation $R \subseteq X \times Y$ is difunctional, if there exists a set Z and projection functions $f_X : X \rightarrow Z, f_Y : Y \rightarrow Z$ such that for each $x \in X$ and $y \in Y$, the relation is determined as $\langle x, y \rangle \in R \equiv (f_X(x) = f_Y(y))$.

Intuitively, this means that we can partition both domains X, Y so that the classes of X are in one-to-one mapping with the classes of Y , and the relation holds between two elements exactly if they belong to corresponding classes.

A well-known and easy-to-prove result is that these two notions are in fact the same:

Proposition 2 (Difunctionality and ZX) *A relation is difunctional iff it has the ZX property.*

Proof It follows immediately from the definitions that any difunctional relation is ZX. Conversely, if a relation $R \subseteq X \times Y$ is ZX, then choosing $Z := 2^X, f_Y := y \mapsto \{x \mid \langle x, y \rangle \in R\}$ and $f_X := x \mapsto \{x' \mid \exists y : \langle x, y \rangle \in R \wedge \langle x', y \rangle \in R\}$ demonstrate that R is difunctional. $\square$

3 Challenge: the whack-a-mole property

Here, we will discuss an important manner in which lenses of a span may fail to work together, even if each of them is individually flawless. The problem is first illustrated by a case study, then rigorously captured in a formal definition.

3.1 Case study: the water tank problem

Here, we include from [5] a cartoon co-engineering problem, followed by a series of unsatisfactory solution attempts, to demonstrate a specific kind of failure called “*whack-a-mole*”¹ behaviour.

Example 9 (Water tank problem) Three engineers collaborate on the design of a water tank (see Fig. 4). The tank is a rectangular block with dimensions x meters by y meters by z meters, and its volume is required to be exactly 1 cubic meters.

Each engineer is responsible for choosing the extent of the tank along a different axis. Thus engineer X is responsible for specifying the width $x::V_X$ in their view model V_X , engineer Y is responsible for specifying the depth $y::V_Y$ in their view model V_Y , while engineer Z is responsible for specifying the height $z::V_Z$ in their view model V_Z . For the purposes of this example, we ignore any other information the engineers have in their model.

Thus we have a source model S with $\text{St}S = \{x, y, z \in \mathbb{R}^+ \mid xyz = 1\}$ and three view models V_X, V_Y, V_Z with $\text{St}V_X = \text{St}V_Y = \text{St}V_Z = \mathbb{R}^+$. Note that the three models and the notion of consistency are symmetric to permutation of the axes.

The views are computed by the trivial projection functions (thus $\langle x, y, z \rangle.\text{GET}_X := x$ etc.); the task is to find appropriate PUT functions to accompany them. We will now review a series of failed (but very educational) attempts to do so.

¹ The colloquial meaning of this term, as defined by Wikipedia [1]: “each time a task is finished or a problem is dealt with, yet another task/problem appears elsewhere. In a military context, the term is used to refer to ostensibly inferior opposing troops who keep reappearing. In a law enforcement context, it refers to criminal activity popping up in another part of a city or region after increased enforcement in one area reduces it there. In a programming/debugging context, it refers to the fact that fixing a bug has a certain chance of creating a new bug which itself needs to be fixed”.

Fig. 3 Illustrating the ZX property **Fig. 4** Three engineers collaborate on the design of a water tank with unit volume

Example 10 (Water tank solution 1: SPLIT) Take the following definition for PUT_X , with PUT_Y , PUT_Z defined symmetrically:

$$\langle x, y, z \rangle . \text{PUT}_X(x') := \begin{cases} \langle x, y, z \rangle & \text{if } x = x' \\ \langle x', 1/\sqrt{x'}, 1/\sqrt{x'} \rangle & \text{otherwise} \end{cases}$$

Clearly GetPut and PutGet holds, so this indeed defines a lens, and there is nothing obviously wrong with it. However, a span formed from these lenses is practically unusable in practice, as we will show below. We expect a view-based co-engineering tool to not restrict the space of potential solutions: taking any consistent source model state as the *desirable state* and any other source model state as the *initial state*. It is prudent to require that any desirable state can be reached from the initial state by repeatedly invoking PUT on some foot of the span. This is not the case for SPLIT: any non-trivial PUT application will result in two of the axes being identical, so any desired end state of $\langle x, y, z \rangle$ where the extents along the three axis are all different is simply unreachable from any other state of the source model. No matter how many consecutive PUT applications we perform, foot models that have reached their desired state in the previous step will revert back to being different, like in a whack-a-mole game.

Example 11 (Water tank solution 2: PROPORTIONAL) Take the following definition for PUT_X , with PUT_Y , PUT_Z defined symmetrically:

$$\langle x, y, z \rangle . \text{PUT}_X(x') := \langle x', y/\sqrt{x'y/z}, z/\sqrt{x'y/z} \rangle$$

(By $xyz = 1$, it is possible to rewrite the updated Y -value as $y\sqrt{x/x'}$, similarly $z\sqrt{x/x'}$ for Z .) This lens proportionally scales back y and z , preserving their ratio; in addition to GetPut and PutGet , it even obeys history ignorance. Never-

theless, every non-trivial application of PUT will change the other two view models. Thus the span still shows whack-a-mole behaviour: after the first engineer sets their model to the desired state, it will be overwritten again when the second engineer does the same. In a mathematical sense, this solution offers more hope: with repeated interleaved updates of feet X and Y , the source model will at least converge to its desired state. In a practical sense, however, requiring infinite amount of steps in a collaboration workflow is unacceptable.

We have another option: from any initial state $\langle x_I, y_I, z_I \rangle$, two (carefully chosen) steps suffice to reach any desired state $\langle x_D, y_D, z_D \rangle$:

$$\langle x_I, y_I, z_I \rangle . \text{PUT}_X((\sqrt[3]{z_I/y_I})(x_D/z_D)^2) . \text{PUT}_Y(y_D) = \langle x_D, y_D, z_D \rangle$$

However, this is still not satisfactory: engineer X would have to deliberately update their model with a state *different* from their desired state x_D ; even worse, it would additionally require knowledge of z_D and the ratio z_I/y_I , both of which should be unknown to X .

Example 12 (Water tank solution 3: CLOCKWISE) Take the following *cyclically symmetrical* definitions for the PUT restorers:

$$\begin{aligned} \langle x, y, z \rangle . \text{PUT}_X(x') &:= \langle x', xy/x', z \rangle \\ \langle x, y, z \rangle . \text{PUT}_Y(y') &:= \langle x, y', zy/y' \rangle \\ \langle x, y, z \rangle . \text{PUT}_Z(z') &:= \langle xz/z', y, z' \rangle \end{aligned}$$

Here, we have three history ignorant lenses, with Y preserving X , Z preserving Y , and X preserving Z . We can now happily observe that:

$$\langle x_I, y_I, z_I \rangle . \text{PUT}_X(x_D) . \text{PUT}_Y(y_D) . \text{PUT}_Z(z_D) = \langle x_D, y_D, z_D \rangle$$

In this case, the last of the three steps can even be skipped. Unfortunately, this favorable behaviour is very fragile. If we do the updates out of order, the final state will not, in general, match the desired state:

$$\langle x_I, y_I, z_I \rangle . \text{PUT}_Z(z_D) . \text{PUT}_Y(y_D) . \text{PUT}_X(x_D) \\ = \left\langle x_D, y_D \frac{x_I z_I}{x_D z_D}, z_D \frac{y_I}{y_D} \right\rangle$$

With the right engineering workflow, the desired state can be achieved; but this requires very tight coordination between the collaborating participants, not suitable for independent or especially distributed co-engineering.

In actual co-engineering scenarios, it would be very onerous to impose a strict workflow that prescribes the exact order in which engineers are allowed to work in. A pragmatic collaborative engineering platform has to guarantee the absence of whack-a-mole behaviour without restricting the freedom of its users. As the above examples show, it is not easy to guarantee this.

3.2 Execution histories: a way out?

The bad news, which will be proved as Corollary 2, is that the water tank problem (Example 9) is unsolvable; in other words, if the feet of the span can be updated in any order, then the absence of whack-a-mole behaviour cannot be guaranteed. How general is this negative result? The proof that will be presented is valid at least in the state-based lens formalism used in the paper; likely also in the commonly used delta-based approaches [11–13].

Nevertheless, a practically feasible *partial* solution to the Water Tank problem does, in fact, exist. It is presented in [5], but it (a) requires us to relax one of our assumptions, by somehow storing extra information (regarding the history of past executions) and furthermore (b) still imposes certain, arguably minor, restrictions on the permitted behaviour of users. This solution is discussed in detail in Appendix A, along with multiple reasons why it is not entirely satisfactory.

3.3 Contributions on controllability

The paper makes the following general contributions in relation to the challenge outlined in this case study.

- We have demonstrated the whack-a-mole problem, as well as the difficulty of avoiding this problem without restricting the freedom of participants. We have argued that nevertheless, co-engineering platforms should guarantee the absence of this problem without forcing a specific workflow on the participants.

- We will present a rigorous treatment of the above requirement in Sect. 6.1, by formally defining the class of *controllable* MX.
- We will introduce a litmus test in Sect. 6.2 that implementors of collaboration platforms can use to easily verify controllability. The test will rely on a novel property of MX that will have been introduced in Sect. 5.1. We will prove the correctness of this technique and demonstrate its usefulness by applying it on the water tank problem.
- In Sect. 5.2.5, we will present other properties of MX that automatically guarantee controllability.

4 Challenge: decomposable, predictable MX

4.1 Motivating decomposable MX

As mentioned, the behaviour of an MX may be very difficult to understand or predict. In addition to the whack-a-mole problem discussed in Sect. 3, there are other questions that are difficult to answer in spans of lenses, ways in which their behaviour is hard to predict. This is true even if the constituent BX are known to work fine in isolation, simply because such questions pertain the interactions of the feet of the span, and thus standard BX properties do not address these concerns.

Examples of such challenges include the difficulty to tell whether the states of two or more models can be simultaneously consistent with the rest of the network, whether updates to two or more models can be applied in arbitrary order, whether modifications to one model are visible in another, whether one model can be computed from a set of other models (without consulting the rest of models), etc. The state of the art (see Sect. 2.3.2) can only provide rather weak guarantees, in very specific circumstances.

Example 13 The water tank solutions presented in Sect. 3.1 suffer from many issues. For instance, the effect of updates to multiple models depends on the order in which they were carried out. Also, the CLOCKWISE solution of Example 12 has the peculiar property that updates to view model V_X cannot affect view model V_Z ; the other two solutions do not have this guarantee, despite enforcing the exact same consistency constraint.

Nevertheless, simple file synchronization services, such as the one depicted in Fig. 1, already work reliably in industry. These systems show easily predictable behaviour, that makes it trivial to answer the above questions (as already demonstrated in Example 1). For instance, concurrent updates to different views can always be merged, unless contradictory (assigning different new values to the same file). Unfortunately, techniques in the state of the art (see Sect. 2.3.2) are unable to provide this guarantee.

Roughly speaking, the key enabling feature of file synchronization networks is that we can consider the models to be made up of parts (files in this case), that the consistency restorers simply copy around. This suggests that there is an important special case of VWB MX that “just work”—intuitively, this is when it solves a simple data synchronization problem. In the binary case, a BX is known to “work like a file synchronization service”, i.e. be VWB (see Definition 7), in the case where there is a well-defined shared model part that is common in the two models, while the BX ignores the complement part in each model. However, up to now no formal definition has been proposed for capturing the analogous case for non-binary synchronizations: synchronization networks that are somehow decomposable into a very simple structure and therefore must exert very regular behaviour.

Remark 5 Not all MX networks can be as simple and predictable as data synchronization services. But if certain parts of a larger MX network work in such a very regular way, then it is sufficient to focus the attention of the MX designers on the other parts, where giving guarantees of predictable behaviour may require a lot of effort. While out of scope for the current paper, we still foresee the challenges stated here (and the mathematical tools developed in response) as being relevant for MX that are less regular in some aspects.

4.2 Motivating criteria for decomposability

Furthermore, it is unclear how to determine whether an MX falls into this category. In the binary case, the VWB property was equivalent to history ignorance (see Proposition 1), which is easy to mechanically check. Unfortunately, in the MX case history ignorance alone does not guarantee that some sort of decomposition can be applied. This is demonstrated by the following synthetic example.

Example 14 (Available as [4]) Let $\text{St}V_A = \text{St}V_B = \{0, 1\}$ and $\text{St}S = \{0, 1\} \times \{0, 1\}$; denote as $\oplus$ the modulo 2 addition operator. Construct span $\langle A, B \rangle : S \rightsquigarrow (V_A, V_B)$, consisting of lens A with $\langle x, y \rangle.\text{GET}_A := x, \langle x, y \rangle.\text{PUT}_A(x') = \langle x', y \rangle$ and lens B with $\langle x, y \rangle.\text{GET}_B := x, \langle x, y \rangle.\text{PUT}_B(x') = \langle x', y \oplus x \oplus x' \rangle$. In other words, A reads and writes the first component of the source tuple, preserving the second; B reads and writes the first component, preserving the binary sum of the two components.

It is easy to check that A and B are in fact lenses, and both are history ignorant. Yet (as Theorem 2 will confirm later) it is not possible to decompose the source model into parts available to one or both lenses, and parts hidden from (and unaffected by) them. It is thus difficult to reason about the behaviour of this MX, despite each constituent lens being history ignorant.

4.3 Contributions on decomposability

The paper makes the following general contributions in relation to the challenge outlined above.

- We have argued that in an MX, updating a model has effects on the other models that may be difficult to predict, unless we have an understanding of the models as being made up of smaller, perhaps shared, parts.
- We will present a generalization of the VWB property for non-binary synchronizations in Sect. 8.1, by rigorously examining how such MX can be decomposed. MX that are VWB are guaranteed to be very predictable in the above sense.
- We will introduce a litmus test in Sect. 8.2 that implementors of collaboration platforms can use to easily verify the VWB property. The test will rely on novel properties of MX we define in Sect. 5 as well as a novel formalization of models and model synchronization networks being decomposed that we propose in Sect. 7. The correctness of the test will be proved in Sect. 10 based on the novel algebraic framework introduced in Sect. 9.

Although the challenge posed here is distinct from that of Sect. 3, it will turn out the answers given will be intricately connected.

5 Novel properties of MXs

Before being able to make our main contributions in Sects. 6 and 8, we have to lay some theoretical groundwork by introducing novel concepts. This section will introduce new properties that MXs may exhibit, which will be used both in Sects. 6 and 8. Similarly, Sect. 7 will introduce precise notions of how MX networks can be broken down into smaller parts, which will be used in Sect. 8 only. Later in Sect. 9, we will also present a novel perspective of a span of lenses as a Boolean algebra; however, this treatment is only needed for the proof in Sect. 10, and is therefore deferred until the main contributions are stated.

5.1 Basic cross-effects among feet of a span

Studying MX requires us to understand how lenses in a span interact with each other. First, we will examine the most straightforward properties that are simple generalizations of the conventional BX properties.

5.1.1 Cross-consistency

First, we have to study whether a foot of a span is consistent with another foot—regardless of the state of other feet and

the source model. In effect, we will obtain a projection of multiary consistency onto a pair of feet, identifying pairs of foot states that can appear together (as far as the span is concerned).

Definition 13 (*Projected cross-consistency relation*) For a span of lenses $\langle A_1, A_2, \dots \rangle : S \rightsquigarrow (V_1, V_2, \dots)$, where each foot A_i enforces a consistency relation $R_i(S, V_i)$, the projected cross-consistency relation between feet i, j is defined as $R_{i,j}^S(V_i, V_j) := \Pi_{V_i, V_j}(R_i \bowtie R_j) = \{ \langle a_i, a_j \rangle \mid a_i :: V_i \wedge a_j :: V_j \wedge \exists s :: S : a_i = s.\text{GET}_i \wedge a_j = s.\text{GET}_j \}$ relates those states of the two feet for which there is at least one state of the source model S that is consistent with both of them.

Example 15 In the file synchronization example of Fig. 1, the specific state of V_A where its copy of file 1 contains “one”, file 2 contains “two” and file 3 contains “three” is cross-consistent with the state of V_C that has “one” as the content of file 1 and “four” as the content of file 4. More generally, $R_{A,C}^S$ contains all pairs of valid states of views V_A and V_C where both views have the same content for file 1.

Example 16 In the water tank example of Example 9, all valid contents of view X are cross-consistent with all valid contents of view Y , since one can always find a state of Z that makes the product 1, making the source model formed from these three elements valid. In other words, $R_{X,Y}^S = \text{St}V_X \times \text{St}V_Y$

5.1.2 Cross-hippocraticness

Below, we will show a systematic way to discover a few interesting conditions that limit cross-interference between lenses in a span. One of these conditions, cross-hippocraticness, will be vital to understand not just controllability, but also the relationship between the axioms of regularity (which will be discussed in Sect. 5.2).

Note how Definition 13 expresses a notion of consistency between view states of a span of lenses. Note also that if we enlarge a span by an additional replicator lens $\top$ that simply (bijectively) extracts the source model, then a view state being cross-consistent with a state of this lens would be the same as saying that the view state is consistent with the source state. This suggests that our ordinary notion of consistency of a lens is just a degenerate case of cross-consistency, with $\top$ taking the role of the other lens. We will now try to apply this insight to find other properties, expressing cross-effects between feet of a span, that likewise degenerate to well-known properties of a single lens. We will attempt this trick at generalizing correctness, hippocraticness, and history ignorance. One of the resulting generalized forms will turn out to be especially interesting and will be discussed here; the other two are of lesser consequence and will only be explored for the sake of completeness in Appendix B.

In particular, we can generalize hippocraticness, so that updating foot i with a new view state will leave foot j alone, as long as the current view state of foot j is cross-consistent with the new state of foot i .

Definition 14 (*Cross-hippocraticness*) For a span of lenses $\langle A_1, A_2, \dots \rangle : S \rightsquigarrow (V_1, V_2, \dots)$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$, the foot A_i is cross-hippocratic towards foot A_j if, for each $s :: S$ and each $a_i :: V_i$ with $R_{i,j}^S(a_i, s.\text{GET}_j)$, we have that $s.\text{GET}_j = s.\text{PUT}_i(a_i).\text{GET}_j$. The span is cross-hippocratic if all pairs of its feet are so.

As promised, A_i being cross-hippocratic towards $\top$ is equivalent to ordinary hippocraticness; cross-hippocraticness towards any other A_j is an independent condition, however.

Example 17 In the file synchronization example of Fig. 1, A is cross-hippocratic towards C , since updating the view A with a new state will not affect the existing state of C as long as both have the same content in file 1 (and hence cross-consistent). The same holds for any other pair of lenses in the span.

Example 18 In the water tank solutions SPLIT of Example 10 and PROPORTIONAL of Example 11, none of the feet are cross-hippocratic towards each other. This is fairly obvious: since all foot state pairs are cross-consistent, any effect PUT_X has on Y is a violation of cross-hippocraticness. The solution CLOCKWISE of Example 12 has the peculiar property that PUT_X never modifies Z , only Y ; so X is cross-hippocratic towards Z , but not Y .

Figure 5a illustrates source states in the usual plot by A and $\bar{A}$, grouped together based on their GET_B value. If A is history ignorant and cross-hippocratic towards B , then “ B -blobs” will show up as “ B -bricks”, with “horizontal” and “vertical” boundaries. The reason for this is whenever we have a source state s_1 with $s_1.\text{GET}_A = a_1$ and another source state s_2 so that $s_1.\text{GET}_B = s_2.\text{GET}_B = b_1$, then by cross-hippocraticness $s_3 := s_2.\text{PUT}_A(a_1)$ also has B -value b_1 . This s_3 is a member of the same B -blob that has the same A -value as s_1 , and (due to A being VWB) the same $\bar{A}$ -value as s_2 and hence the rectangular-looking shape. Though, as before, the ordering imposed on the axes can be arbitrary permuted, there is no guarantee the B -bricks will form a single contiguous rectangle (as opposed to multiple smaller rectangles, still parallel to the axes).

5.2 The axioms of regularity

Beyond the simple cross-effects investigated above, we examine additional properties (collectively called the four *axioms of regularity*), that will characterize spans of lenses that are easier to reason about, but are not necessarily

Fig. 5 Visualization of advanced concepts (Fig. 5a and c additionally assume the history ignorance of A)

expected to hold for every practically relevant MX. In the context of BX, history ignorance was such a nice-to-have but not mandatory property. This will, in fact, be our first axiom of regularity:

Definition 15 (*History ignorant span of lenses*) A span of lenses is history ignorant if all its constituent lenses are so.

The other three axioms of regularity are novel concepts that will be introduced below. At first, it might appear like these properties are included at random; but later in the paper we will see that this is not the case: There are deep connections between these axioms (see Appendix D), and furthermore, the four of them combined will jointly guarantee especially predictable behaviour (see Sect. 8.2).

5.2.1 Projectional difunctionality

Our first new requirement is a well-defined notion of the knowledge (information) that the MX passes along between two involved models. This is captured by the following definition.

Definition 16 (*Projectional difunctionality*) For span of lenses $S \rightsquigarrow (V_1, V_2, \dots) = \langle A_1, A_2, \dots \rangle$, the feet i, j are *projectionally difunctional* if the projected cross-consistency relation $R_{i,j}^S(V_i, V_j)$ is difunctional. A span is projectionally difunctional if all pairs of its feet are so.

Example 19 In the file synchronization example of Fig. 1, A is projectionally difunctional with C , since the content of file 1 is the entirety of information shared between these two views. In other words, a view state of V_A is cross-consistent with a view state of V_C if and only if they hold the same content for file 1; the difunctionality is witnessed by the two functions that extract file 1 from V_A or V_C , respectively. The same holds for any other pair of lenses in the span.

Example 20 In the water tank example of Example 9, all valid contents of view X are cross-consistent with all valid contents of view Y , implying that they are also projectionally difunctional (regardless how consistency restorers are chosen). The difunctionality is witnessed by a pair of constant functions taking the same value on all states of both view models.

Figure 5b depicts B -blobs on the usual plot in a case where B is projectionally difunctional with A . Two B -blobs either have entirely disjoint A -values, or all the same A -values. This means that the individual B -blobs will form bands of tightly packed vertical “columns”. Let us now see how difunctionality guarantees that there is a well-defined notion of shared knowledge between the two lenses! For any A -value or B -value, we can tell which “column” it will fall into. Furthermore, if we do not know the source state but know its associated “column”, then we have some partial information in the value of B (only a subset of $\text{St}B$ is compatible with the “column”); and, interestingly, knowing the value of A does not add any more information on B (does not allow us to restrict this subset further). In sum, the identity of the column is exactly the knowledge that is shared between A and B . Later, Sect. 9.2 will investigate the case where this “intersection of knowledge” is not just a view, but in fact an updateable view itself: the lens $A \cap B$.

Remark 6 As a degenerate case of Definition 16, each lens in a span is considered projectionally difunctional with itself.

5.2.2 Freedom from leaks

Lenses perform an information hiding function, either for access control or just for separation of concerns/abstraction. While they achieve this goal statically, they might fail at keeping the same information hidden when we are allowed to observe an entire sequence of states. We are thus interested in ensuring that, if A_i hides from V_i some information stored

in S , then any subsequent update from any other foot A_j shall keep this information about the past state hidden from V_i . Hiding information from V_i means that there are states $s_1, s_2 :: S$ that are not possible to distinguish based on their V_i -projection. Therefore, PUT_j must map states of S in a way that preserves GET_i -congruence.

Definition 17 (*Leak-free span*) For a span of lenses $\langle A_1, A_2, \dots \rangle$ $S \rightsquigarrow (V_1, V_2, \dots)$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$, the foot i is free from leaks via j if, for any $s_1, s_2 :: S$ with $s_1.\text{GET}_i = s_2.\text{GET}_i$ and any $a_j :: V_j$, we have that $s_1.\text{PUT}_j(a_j).\text{GET}_i = s_2.\text{PUT}_j(a_j).\text{GET}_i$. A span is leak-free if all pairs of its feet are leak-free in both directions.

Example 21 In the file synchronization example of Fig. 1, C is free from leaks via A , since overwriting V_A will not reveal to V_C the old contents of files 2 and 3, or the unchanged content of file 5. If two source states are not possible to distinguish based on files 1 and 4 alone, then overwriting files 1, 2, and 3 (with the same content in both cases) will still not reveal which one was the original source state, since V_C will only show the unchanged value of file 4 and the common new value of file 1.

Example 22 In the water tank solution SPLIT of Example 10, engineer Y can learn x even though that model part is supposed to be hidden by lens Y . While each value $y :: V_Y$ is cross-compatible with all valid values $x :: V_X$, in practice if V_Y is updated by $\text{PUT}_X(x')$, engineer Y can deduce x' based on the new value of V_Y . Concretely, the new Y -value must be $1/\sqrt{x'}$ by the definition of SPLIT; thus, x' is $1/y^2$.

The alternative solution PROPORTIONAL of Example 11 also suffers from leaks: initial states $\langle 1, 1, 1 \rangle$ and $\langle 0.25, 1, 4 \rangle$ are indistinguishable based on Y , but $\text{PUT}_X(0.25)$ would take them, respectively, to $\langle 0.25, 2, 2 \rangle$ and $\langle 0.25, 1, 4 \rangle$, which Y can tell apart; so if engineer Y has a good guess on the new value of X , they have also learned something about its old value. Due to symmetry, any lenses in the span can leak supposedly hidden information towards any other.

CLOCKWISE works similarly: X can leak sensitive information to Y , as can Y to Z , and Z to X . There are no leaks in the opposite direction though, as e.g. PUT_X is ineffectual on Z .

Figure 5c depicts B -blobs on the usual plot in a case where B is free of leaks from A and where A is history ignorant. Now there will be non-overlapping rows formed of the B -blobs, for the following reasons. If two source states have the same B -value, freedom from leaks implies that they retain this congruence after a $\text{PUT}_A(a)$ for any a . Since A is VWB, the same operation is ineffectual on $\bar{A}$ and show up on the plot as a purely horizontal move. Thus, at any A -value, the backwards equivalence classes of these two states will be associated with the same B -value. All such backwards equivalence classes must therefore form rows within which the

B -value is entirely determined by the A -value. In sum, the identity of the row is exactly the information in B that is missing from A . Later, Sect. 9.2 will investigate the case where this “difference of knowledge” is not just a view, but in fact an updateable view itself: the lens $B \setminus A$.

Remark 7 Figure 5c shows B -bricks instead of mere blobs; the above argument explains why they must be rectangular. Such bricks heralded cross-hippocraticness before (in the history ignorant case); it is generally true that freedom from leaks implies cross-hippocraticness (even without assuming history ignorance), as will be proved in Lemma 1.

Remark 8 The columns in Fig. 5b and the rows in Fig. 5c appear very similar, just with the two axes swapped. We will formalize this as a (conditional) duality between projectional difunctionality and leak freedom in Appendix D.

Remark 9 As a degenerate case of Definition 17, each lens in a span is considered free from leaks via itself.

5.2.3 Minimal interference

The non-interference property (see Definition 10 and [32]) aims to ensure that updates from the various feet may come in arbitrary order and still retain the same overall effect; but as pointed out in [32], it is a very restrictive property that requires the feet to be entirely independent. In particular, it would require all pairs of states for the two feet to be cross-consistent; otherwise, correctness/PutGet would naturally require the two updates to overwrite each other. This motivates us to use a relaxation of the non-interference property: it will suffice that updates arriving from the various feet would not “overwrite” each other in those cases where they do not contradict each other (i.e. are cross-consistent).

Definition 18 (*Minimal interference*) For a span of lenses $\langle A_1, A_2, \dots \rangle = S \rightsquigarrow (V_1, V_2, \dots)$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$, the feet $i \neq j$ are minimally interfering if for all cross-consistent foot states $\langle a_i, a_j \rangle \in R_{i,j}^S(V_i, V_j)$ and source state $s :: S$ we have that $s.\text{PUT}_i(a_i).\text{PUT}_j(a_j) = s.\text{PUT}_j(a_j).\text{PUT}_i(a_i)$. A span is minimally interfering if all pairs of its feet are minimally interfering.

Example 23 In the file synchronization example of Fig. 1, A is minimally interfering with C , since overwriting both V_A and V_C can happen in any order, the end result will be the same (files 1, 2, 3, and 4 replaced in the source model) as long as the two updates are cross-consistent (agree in the content of file 1). The same is true for any other pair of lenses in the span.

Example 24 In the water tank problem of Example 9, all pairs of feet are cross-consistent, so minimal interference degenerates into non-interference and requires that any pairs

of updates (on two distinct feet) have the same end result regardless of order of execution. In the solutions SPLIT of Example 10 and PROPORTIONAL of Example 11, the order of updates matter, since only the most recently updated dimension is guaranteed to end up with its desired value. In case of CLOCKWISE (Example 12), both dimensions will end up in the desired state in one of the update orders, but not in the other one. All in all, none of the solutions guarantee minimal interference.

Remark 10 The latter observation is unsurprising, since these solutions to the water tank problem were already found to lack cross-hippocraticness, and (as we will prove in Lemma 2) minimal interference implies cross-hippocraticness in both directions.

Remark 11 As a degenerate case of Definition 18, each lens in a span is considered minimally interfering with itself.

5.2.4 Regularity

Based on the above definitions, we can now assemble the four axioms which we require in order to consider a span as having sufficiently regular behaviour.

Definition 19 (*Regular span of lenses*) A span of lenses is regular if all of the following four conditions hold:

- The span is history ignorant.
- The span is projectionally difunctional.
- The span is leak-free.
- The span is minimally interfering.

We have already mentioned (and will prove in Sect. 5.2.5) that freedom from leaks and minimal interference both individually imply cross-hippocraticness, so regularity is a stricter condition than cross-hippocraticness.

On the other hand, the axioms of regularity are *independent*, meaning none of them are redundant in the definition of a regular span, since none of the four axioms of regularity are implied by a combination of the other three. This can be easily shown by listing four counterexamples, each of which are regular save for one of the axioms. This is the purpose of Sect. 5.2.6.

5.2.5 Relationships with cross-hippocraticness

We will take advantage of the knowledge that two of the axioms of regularity both individually imply cross-hippocraticness, and hence, regular spans are always cross-hippocratic.

Lemma 1 *For a span of lenses $\langle A, B, \dots \rangle : S \rightsquigarrow (V_A, V_B, \dots)$, if B is free from leaks via A , then A is cross-hippocratic towards B .*

Proof Take states $s :: S, a :: V_A$ with $b := s. \text{GET}_B$ and $R_{A,B}^S(a, b)$ witnessed by $s' :: S$. We have to prove $s. \text{PUT}_A(a). \text{GET}_B = b$. Since s and s' are congruent in B which is leak-free from A :

$$s. \text{PUT}_A(a). \text{GET}_B = s'. \text{PUT}_A(a). \text{GET}_B$$

GetPut simplifies the right-hand side to $s'. \text{GET}_B = b$. $\square$

Lemma 2 *For a span of lenses $\langle A, B, \dots \rangle : S \rightsquigarrow (V_A, V_B, \dots)$, if A, B are minimally interfering, then B is cross-hippocratic towards A (and vice versa).*

Proof As minimal interference is symmetric in A, B , it suffices to prove one direction. Take states $s :: S, b :: V_B$ with $a := s. \text{GET}_A$ and $R_{A,B}^S(a, b)$. We have to prove $s. \text{PUT}_B(b). \text{GET}_A = a$. By minimal interference and the cross-consistency assumption, we have

$$s. \text{PUT}_A(a). \text{PUT}_B(b) = s. \text{PUT}_B(b). \text{PUT}_A(a)$$

Applying GET_A to both sides:

$$s. \text{PUT}_A(a). \text{PUT}_B(b). \text{GET}_A = s. \text{PUT}_B(b). \text{PUT}_A(a). \text{GET}_A$$

Simplifying the left-hand side with GetPut, the right-hand side with PutGet:

$$s. \text{PUT}_B(b). \text{GET}_A = a$$

$\square$

5.2.6 Independence of axioms

Lemma 3 *There is a span of lenses that is projectionally difunctional, minimally interfering, free from leaks but not history ignorant.*

Proof Any 1-wide span consisting of a single, history non-ignorant lens fits the criteria. (Or for any arbitrary span arity: an n -wide span consisting of n identical history non-ignorant lenses is also a counterexample.) $\square$

Lemma 4 *There is a span of lenses that is history ignorant, free from leaks, projectionally difunctional but not minimally interfering.*

Proof Such an example was already given as Example 14. It is easy to check that A and B are in fact lenses, and both are history ignorant and free from leaks via the other. Their cross-consistency relation is $\{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$, making them projectionally difunctional. However, they are not minimally interfering: starting from source state $\langle 0, 0 \rangle :: S$, updating both views with value 1 (which are thus cross-consistent) may end up in $\langle 1, 0 \rangle$ or $\langle 1, 1 \rangle$ depending on the order. $\square$

Lemma 5 *There is a span of lenses that is history ignorant, minimally interfering, free from leaks but not projectionally difunctional.*

Proof The following counterexample is available online at [3].

Let $\text{St}V_A = \mathbb{Z}$, $\text{St}V_B = \mathbb{Z} \times \mathbb{Z}$ and $\text{St}S = \mathbb{Z} \times \mathbb{Z}$. Construct span $\langle A, B \rangle : S \rightsquigarrow (V_A, V_B)$ as:

$$\begin{aligned} \langle a, c \rangle. \text{GET}_A &:= a \\ \langle a, c \rangle. \text{PUT}_A(a') &= \langle a', c \rangle \\ \langle a, c \rangle. \text{GET}_B &:= \lfloor (a + c)/2 \rfloor, c \\ \langle a, c \rangle. \text{PUT}_B(\langle b'_1, b'_2 \rangle) &= \langle 2b'_1 - b'_2 + ((b'_2 + a) \bmod 2), b'_2 \rangle \end{aligned}$$

In other words, A reads and writes the first component of the source tuple, preserving the second; B reads and writes all information in the source, except $a \bmod 2$.

It is easy to check that A and B are in fact lenses, and both are history ignorant. Since PUT_B depends on a but not on its complement c , A is leak-free from B . As GET_B -congruent states only differ in $a \bmod 2$, B is leak-free from A . Any given $\langle b_1, b_2 \rangle :: V_B$ is cross-compatible with two states of V_A , namely $a_0 := 2b_1 - b_2$ and $a_1 := 2b_1 - b_2 + 1$. It is easy to check that $\text{PUT}_B(\langle b_1, b_2 \rangle)$ commutes with both $\text{PUT}_A(a_0)$ and $\text{PUT}_A(a_1)$, making the two lenses minimally interfering.

Source states $\langle 0, 0 \rangle, \langle 1, 0 \rangle :: S$ witness that $\langle 0, 0 \rangle :: V_B$ is cross-consistent with $0, 1 :: V_A$. Also, $\langle 0, 1 \rangle :: S$ witnesses that $\langle 0, 1 \rangle :: V_B$ is cross-consistent with $0 :: V_A$. But it is not cross-consistent with $1 :: V_A$ (since $\langle 1, 1 \rangle. \text{GET}_B = \langle 1, 1 \rangle$), violating projectional difunctionality. $\square$

Lemma 6 *There is a span of lenses that is history ignorant, minimally interfering, projectionally difunctional but not free from leaks.*

Proof The following counterexample is available online at [3].

Consider the span given in the proof for Lemma 5, but instead of A take the lens C defined as:

$$\begin{aligned} \langle a, c \rangle. \text{GET}_C &:= c \\ \langle a, c \rangle. \text{PUT}_C(c') &= \langle a, c' \rangle \end{aligned}$$

Now C is obviously a history ignorant lens. Clearly $B \supseteq C$, in other words each value of B is cross-compatible with exactly one value of C . This alone implies that B and C are projectionally difunctional. Furthermore, it is easy to check that $\text{PUT}_B(\langle b_1, b_2 \rangle)$ commutes with $\text{PUT}_C(b_2)$, making the two lenses minimally interfering.

Source states $\langle 0, 0 \rangle, \langle 1, 0 \rangle :: S$ are congruent in B , both yielding $\langle 0, 0 \rangle :: V_B$. Applying $\text{PUT}_C(1)$ leads to source states $\langle 0, 1 \rangle, \langle 1, 1 \rangle :: S$, respectively, yielding $\langle 0, 1 \rangle, \langle 1, 1 \rangle :: V_B$; thus, B is not free from leaks via C . $\square$

Remark 12 The lens C in the Proof of Lemma 6 extracts exactly the complement of A in the Proof of Lemma 5. We will introduce a formal notion of lens complement in Sect. 9.2.1 under which $C = \bar{A}$ can be rigorously stated. Furthermore, the lack of projectional difunctionality in one case and the lack of freedom from leaks in the other are, in fact, equivalent; we have already discussed that there is a duality between these two properties, which will be formally treated in Appendix D.

Corollary 1 *None of the four axioms of regularity are implied by the conjunction of the other three.*

6 Results on controllability

In this section, we finally introduce our main contributions regarding controllability.

6.1 Formulating the requirement of controllability

MXs in general are applicable in many scenarios, for instance collaborative engineering. As argued in Sect. 3.1, at least for the latter purpose, they are only useful if they avoid whack-a-mole behaviour. We would like to be able to reach any desired state by updating, one by one and in any order, each foot lens to its desired value. We shall require that none of these steps would invalidate an earlier one; a foot that has already been set to its desired state will stay that way, without a mole popping up. In other words: if, following an arbitrary permutation of the feet, we have already set the first j feet to their desired values, then for $i \leq j$, the i th foot is still in this state. Borrowing the terminology of observability/controllability from control engineering, this is a *controllability* issue.

Definition 20 (*Controllable span of lenses*) We call the n -wide span of lenses $\langle A_1, A_2, \dots, A_n \rangle : S \rightsquigarrow (V_1, V_2, \dots, V_n)$ consisting of $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$ controllable if for any permutation $\pi \in \mathbb{S}_n$ of the feet $\{1 \dots n\}$, and each pair of initial and desired source states $s_I, s_D :: S$ with the denotation $a_D^k := s_D. \text{GET}_{\pi(k)}$, and any pair of indices $1 \leq i \leq j \leq n$, we have that

$$s_I. \text{PUT}_{\pi(1)}(a_D^1). \text{PUT}_{\pi(2)}(a_D^2) \dots \text{PUT}_{\pi(j)}(a_D^j). \text{GET}_{\pi(i)} = a_D^i$$

Remark 13 The above definition of controllability is not total, in the sense that it does not imply that s_D will be actually reached after n updates, merely that the final source state will be indistinguishable from s_D via this span of lenses. Referring back to Example 9, if the source model were to contain an additional field for the *color* of the water tank, without it being visible in (or writeable through) any of the three view models under study, then the color of the tank cannot be influenced through these views. This is clearly

a limit of *observability*, but it does not necessarily signify a failure in the way the various PUT restorers interact with each other, so we will not consider it a lack of controllability here. The interesting case is when the view models can jointly observe, but still not control, some “part” of the source model.

Remark 14 While the above definition concerns spans of lenses only, Appendix C includes a formal definition for the slightly more general case that has arbitrary BXs instead of lenses.

While controllability is formally defined above, it is a complex definition that requires us to consider many model states and permutations of feet. Fortunately, there is a very simple way to test whether a span is controllable, which will be presented in Sect. 6.2.

6.2 Characterizing controllable synchronizations

We now state our first main result, the relationship between cross-hippocraticness and controllability, and apply it to prove the negative result promised in Sect. 3.

Theorem 1 (Controllability equals cross-hippocraticness) *A span of lenses is controllable iff it is cross-hippocratic.*

Proof (Forward direction) Assume that the n -wide span $\langle A_1, A_2, \dots, A_n \rangle : S \rightsquigarrow (V_1, V_2, \dots, V_n)$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$ is controllable. To prove that A_j is cross-hippocratic towards A_i , we have to show $s.\text{PUT}_j(a_j).\text{GET}_i = s.\text{GET}_i$ for any states $s :: S, a_j :: V_j$ with $R_{i,j}^S(s.\text{GET}_i, a_j)$. Take a witness for the latter $s_D :: S$, such that $s_D.\text{GET}_i = s.\text{GET}_i$ and $s_D.\text{GET}_j = a_j$, also take any permutation π with $\pi(1) = i$ and $\pi(2) = j$. Controllability towards desired state s_D gives

$$s.\text{PUT}_i(s_D.\text{GET}_i).\text{PUT}_j(s_D.\text{GET}_j).\text{GET}_i = s_D.\text{GET}_i$$

Now we can substitute $s_D.\text{GET}_i$ and $s_D.\text{GET}_j$ from the definition of s_D :

$$s.\text{PUT}_i(s.\text{GET}_i).\text{PUT}_j(a_j).\text{GET}_i = s.\text{GET}_i$$

Finally, eliminate the first update via GetPut:

$$s.\text{PUT}_j(a_j).\text{GET}_i = s.\text{GET}_i \quad \square$$

Proof (Backward direction) Take a cross-hippocratic span $\langle A_1, A_2, \dots, A_n \rangle : S \rightsquigarrow (V_1, V_2, \dots, V_n)$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$. Take initial and desired source states $s_I, s_D :: S$, a permutation $\pi \in S_n$ with $a_D^k := s_D.\text{GET}_{\pi(k)}$ and index $1 \leq i \leq n$. By PutGet, clearly

$$s_I.\text{PUT}_{\pi(1)}(a_D^1).\text{PUT}_{\pi(2)}(a_D^2) \dots \text{PUT}_{\pi(i)}(a_D^i).\text{GET}_{\pi(i)} = a_D^i$$

Now by induction on subsequent index $j \in (i, n]$, we have to show that the congruence still stands after update step j , provided that it stands at step $j - 1$. Thus the induction hypothesis is that

$$s_I.\text{PUT}_{\pi(1)}(a_D^1).\text{PUT}_{\pi(2)}(a_D^2) \dots \text{PUT}_{\pi(j-1)}(a_D^{j-1}).\text{GET}_{\pi(i)} = a_D^i$$

By Definition 13, $R_{\pi(i), \pi(j)}^S(a_D^i, a_D^j)$. As A_j is cross-hippocratic towards A_i , so:

$$\begin{aligned} s_I.\text{PUT}_{\pi(1)}(a_D^1) \dots \text{PUT}_{\pi(j-1)}(a_D^{j-1}).\text{PUT}_{\pi(j)}(a_D^j).\text{GET}_{\pi(i)} &= \\ s_I.\text{PUT}_{\pi(1)}(a_D^1) \dots \text{PUT}_{\pi(j-1)}(a_D^{j-1}).\text{GET}_{\pi(i)} &= a_D^i \end{aligned}$$

□

The correspondence in the above theorem is especially useful, since cross-hippocraticness has a much simpler definition, and it is much more economical to manually check. We will immediately apply it to find a (negative) answer to the water tank case study.

Corollary 2 *There is no controllable solution to the water tank problem of Example 9 (in the state-based formalism, without extending the source model).*

Proof By Theorem 1, we only have to prove that there is no cross-hippocratic solution. By Definition 14, such a solution would have X being cross-hippocratic towards both Y and Z . However, all values of X are cross-compatible with all values of Y , as well as all values of Z (i.e. $R_{X,Y}^S = R_{X,Z}^S = \mathbb{R}^+ \times \mathbb{R}^+$). So in this case PUT_X would be forbidden from changing either of the other two components. Fixing Y and Z will also fix X , so PUT_X will not be able to meet PutGet. □

We may say that in this case, it is the required consistency relations themselves that are fundamentally *uncontrollable*.

6.3 Discussion

The verdict is that it does not seem like a good idea to implement automatic restoration of complex consistency rules in an MX setting, as it may lead to a loss of controllability. Instead, we should accept the compromise that such high-level constraints may be left broken for a prolonged time (also argued in e.g. [26]) until manually repaired.

This raises the importance of focusing on automated synchronization mechanisms that behave very predictably, even at the cost that they can only guarantee restricted forms of low-level consistency. This is the kind of result that is delivered by Sect. 8, but first some necessary groundwork in Sect. 7.

7 Decomposition of MXs

We have motivated in Sect. 4.1 the need for a formal notion of decomposing models and synchronization networks into smaller parts. The goal of a decomposition is to allow us to handle and reason on several smaller models instead of a single larger one. Hopefully, such simpler models may have simpler relationships to other models.

Here, we introduce a formal framework to reason about such decomposition relationships between models, as well as how decomposition affects consistency constraints and especially consistency restorers.

7.1 Decomposition of models

We have already introduced the notion of model part (as defined in [32]) in Sect. 2.1, but when do such model parts constitute a *decomposition* of the original model?

Intuitively, decomposing a model means that we derive multiple smaller components from the original model in a way that the states of the component models are jointly sufficient to reproduce the information content of the original model. Let us now introduce a formal definition for this notion of model decomposition, and then see how it can be used to replace the composite model with smaller component models in considerations such as consistency constraints.

Definition 21 (Model decomposition) Let M be a model, and $M_1, M_2, \dots$ likewise be models. A *decomposition* ψ of M into component models $M_1, M_2, \dots$ is an *injective* function $\psi : \text{St}M \rightarrow \text{St}M_1 \times \text{St}M_2 \times \dots$ that assigns to each model state $m :: M$ a *unique* tuple (the *multimodel tuple*) formed of states of the component models. The *component functions* extracting the individual components of the result tuple of the decomposition are denoted $\psi_k : \text{St}M \rightarrow \text{St}M_k$.

Remark 15 To avoid derailing the main argument, the paper will focus on the case with a finite number of components.

Example 25 In the file synchronization example of Fig. 1, view models V_A, V_B, V_C and the associated model parts $\text{GET}_A, \text{GET}_B, \text{GET}_C$ that compute them do not constitute a decomposition of S , since the entire state of the source model (specifically file 5) cannot be reconstructed from these parts.

Example 26 In the file synchronization example of Fig. 1, a valid decomposition of the source model S into components X, Y , where $\text{St}X = \text{St}Y = \mathbb{F}^3$, could be $\psi : \mathbb{F}^5 \rightarrow \mathbb{F}^3 \times \mathbb{F}^3$ defined as $\langle s_1, s_2, s_3, s_4, s_5 \rangle \mapsto \langle \langle s_1, s_2, s_3 \rangle, \langle s_3, s_4, s_5 \rangle \rangle$. In essence, X would contain files 1 through 3, file Y would contain files 3 through 5. Together, they can recover all the contents in the source model.

Example 27 In the file synchronization example of Fig. 1, an alternative decomposition of S into components

S^1, S^2, S^3, S^4, S^5 (all of them with state space $\mathbb{F}$) would be $\psi' : \mathbb{F}^5 \rightarrow \mathbb{F}^5$ defined as $\langle s_1, s_2, s_3, s_4, s_5 \rangle \mapsto \langle s_1, s_2, s_3, s_4, s_5 \rangle$; in essence, we extract each of the files 1–5 into their own model. Again, the component models together contain all information in the source model.

Example 28 In the file synchronization example of Fig. 1, a decomposition of V_A into file granularity components V_A^1, V_A^2, V_A^3 (all of them with state space $\mathbb{F}$), as depicted in Fig. 6a, would be $\psi'' : \mathbb{F}^3 \rightarrow \mathbb{F}^3$ defined as $\langle a_1, a_2, a_3 \rangle \mapsto \langle a_1, a_2, a_3 \rangle$. In essence, we extract each of the files 1–3 into their own model. Again, the component models together contain all information in the view model.

Example 29 Let us now see two more cases of model decomposition that also demonstrate decomposition in multiple steps.

In the water tank problem of Example 9, a valid decomposition of the source model S into components S', S'' , where $\text{St}S' = \text{St}S'' = \text{St}S$, could be $\psi : \text{St}S \rightarrow \text{St}S' \times \text{St}S''$, depicted in Fig. 7a, defined as $\langle x, y, z \rangle \mapsto \langle \langle x, y, z \rangle, \langle x, y, z \rangle \rangle$. In essence, S' and S'' would just be two identical copies of S . Obviously, they can (even individually) recover all the contents in the source model.

A further decomposition ψ' of S'' into V_X, V_Y, V_Z (as shown in Fig. 7b) can then be defined as $\langle x, y, z \rangle \mapsto \langle x, y, z \rangle$. This just cuts one of the identical copies of the original source model into three parts that can together recover the contents of that part.

7.2 Interconsistency of components

Definition 21 does not require ψ to be surjective; in other words, not all combination of model states in $\text{St}M_1, \text{St}M_2, \dots$ have to correspond to a model state of M . The *interconsistency constraint* associated with ψ is the constraint on $M_1, M_2, \dots$ that requires the tuple of component model states to belong to the actual range $\text{Im}(\psi)$ of the decomposition function ψ . When considering the collection of component models as opposed to the single original model, this interconsistency constraint can be treated as one of the consistency constraints acting on the multimodel tuple.

Definition 22 (Interconsistency constraint) Let ψ be a decomposition of composite model M into component models $M_1, M_2, \dots$. The interconsistency constraint $\text{Con}[\psi] := \text{Im}(\psi)$ of decomposition ψ is a consistency relation over $M_1, M_2, \dots$, containing exactly those multimodel state tuples in $\text{St}M_1 \times \text{St}M_2 \times \dots$ that are in the range $\text{Im}(\psi)$ of ψ , i.e. have a corresponding model state in the original composite model.

In sum, decomposition ψ turns one model M into multiple models $M_1, M_2, \dots$, plus a consistency constraint $\text{Con}[\psi]$ that will have to be maintained.

Fig. 6 Decomposing the file synchronizers

Fig. 7 Decomposing the water tank models

Example 30 In the file synchronization-related decomposition example of Example 26, the decomposition ψ of the source model S into components X, Y gives rise to interconsistency constraint $\text{Con}[\psi](X, Y)$ that pairs up those model states of X and Y that agree on file 3: $\text{Con}[\psi] = \{\langle x, y, z \rangle, \langle z, v, w \rangle \mid x, y, z, v, w \in \mathbb{F}\}$.

Example 31 The alternative file granularity decomposition ψ' of S into components F_1, F_2, F_3, F_4, F_5 does not require an interconsistency constraint, since all component file states are compatible with all other component file states. Formally, $\text{Con}[\psi'] = \mathbb{F}^5$. Similarly for the file granularity decomposition ψ'' of V_A .

Example 32 In the water tank-related decomposition example of Example 29, the decomposition ψ of S into components S', S'' induces interconsistency constraint $\text{Con}[\psi](S', S'')$ that checks whether the two component models have the same state: $\text{Con}[\psi] = \{\langle s, s \rangle \mid s \in \text{St}S\}$.

Note that this is a mirror constraint; thus, it can be maintained by a replicator BX : $S' \rightleftharpoons S''$, as shown in Fig. 7a.

The second decomposition ψ' of S'' into V_X, V_Y, V_Z gives rise to the ternary constraint $\text{Con}[\psi'](V_X, V_Y, V_Z)$ that checks if the three components correspond to a valid composite model (jointly span a unit volume): $\text{Con}[\psi'] = \{\langle x, y, z \rangle \mid x, y, z \in \mathbb{R}^+ \wedge xyz = 1\}$.

7.3 Rewriting of constraints along decompositions

Next, we can analyse the effect of model decomposition on the set of consistency constraints to be maintained. The original composite model M might have been involved in one or more BX or MX , maintaining various consistency relationships with other models. How are these constraints affected if ψ replaces M with $M_1, M_2, \dots$?

Each constraint on M will be replaced with a constraint on the component models; these rewritten constraints, alongside

the interconsistency constraint introduced by the decomposition itself, allow us to consider the component models only. Recall from Definition 1 that we can represent a consistency constraint $R(M, N_1, N_2, \dots)$ on M (to be decomposed by ψ) and *external* models $N_1, N_2, \dots$ (unaffected by the decomposition ψ) as a set of allowed multimodel state tuples. A decomposition ψ allows us to rewrite it onto a consistency constraint that affects component models $M_1, M_2, \dots$ instead of the composite model M . This is achieved by substituting the composite state $m::M$ with component states $\psi(m)$ in each multimodel state tuple.

Definition 23 (*Constraint rewritten along decomposition*) For consistency constraint $R(M, N_1, N_2, \dots)$ and decomposition ψ of model M into components $M_1, M_2, \dots$, a consistency constraint $R_\psi(M_1, M_2, \dots, N_1, N_2, \dots)$ (implying $R_\psi \subseteq \text{St}M_1 \times \text{St}M_2 \times \dots \times \text{St}N_1 \times \text{St}N_2 \times \dots$) is a rewriting of R along ψ if for any $m::M$ with $\psi(m) = \langle m_1, m_2, \dots \rangle$ and any $n_1::N_1, n_2::N_2, \dots$, we have $\langle m, n_1, n_2, \dots \rangle \in R \Leftrightarrow \langle m_1, m_2, \dots, n_1, n_2, \dots \rangle \in R_\psi$.

Remark 16 Note that this definition does not restrict whether or not R_ψ holds for any multimodel tuple that does not correspond to a model state $m::M$. This is not an issue, since the interconsistency constraint $\text{Con}[\psi]$ is guaranteed to be included in the conjunction of all constraints on the component models. Such degrees of freedom in R_ψ can be used to simplify its implementation, e.g. by exploiting the observations in Sect. 2.1.3, that suggest replacing the constraint with multiple simpler constraints that are jointly equivalent with it. At the same time, however, there is no slack in specifying R_ψ for any valid $\psi(m)$.

Example 33 Continuing the file synchronization example of Example 28, there is a consistency constraint $R_A(S, V_A)$ (enforced by lens A) that checks whether V_A agrees with S , in the form of $R_A = \{\langle s_1, s_2, s_3, s_4, s_5 \rangle, \langle s_1, s_2, s_3 \rangle \mid s_1, s_2, s_3, s_4, s_5 \in \mathbb{F}\}$. If we now apply the file granularity decomposition ψ'' of V_A into V_A^1, V_A^2 and V_A^3 , this naturally rewrites R_A into quaternary constraint $(R_A)_{\psi''}(S, V_A^1, V_A^2, V_A^3)$ defined as $\{\langle s_1, s_2, s_3, s_4, s_5 \rangle, s_1, s_2, s_3 \mid s_1, s_2, s_3, s_4, s_5 \in \mathbb{F}\}$.

Clearly, the quaternary constraint $(R_A)_{\psi''}$ can be replaced with the conjunction of three looser (but still quaternary) constraints that each only check the equality of one of the files 1, 2, and 3: $(R_A)_{\psi''} = R_A^1 \cap R_A^2 \cap R_A^3$ where $R_A^1(S, V_A^1, V_A^2, V_A^3)$ is the relation $R_A^1 = \{\langle s_1, s_2, s_3, s_4, s_5 \rangle, s_1, s'_2, s'_3 \mid s_1, s_2, s_3, s_4, s_5, s'_2, s'_3 \in \mathbb{F}\}$ and analogously for R_A^2 and R_A^3 . Then, each of these three subconstraints can be restricted to two models: e.g. as $R_A^1(S, V_A^1, V_A^2, V_A^3)$ actually ignores V_A^2 and V_A^3 , it can be replaced with binary constraint $Q_A^1(S, V_A^1)$ specified as the relation $Q_A^1 = \{\langle s_1, s_2, s_3, s_4, s_5 \rangle, s_1 \mid s_1, s_2, s_3, s_4, s_5 \in \mathbb{F}\}$; analogously for $Q_A^2(S, V_A^2)$ and $Q_A^3(S, V_A^3)$. This demonstrates that $(R_A)_{\psi''}$ is binary definable (see Sect. 2.1.3).

Since these binary subconstraints are functional relations, they can each be maintained by a lens. This is how they are depicted in Fig. 6a, e.g. A^2 maintains $Q_A^2(S, V_A^2)$.

Example 34 In the water tank-related decomposition example of Example 29 and Example 32, S is first decomposed along ψ into components S', S'' ; giving rise to interconsistency constraint $\text{Con}[\psi](S', S'')$ that checks whether the two copies have the same state. Then S'' is further decomposed along ψ' into V_X, V_Y, V_Z .

In order to fully replace S'' with its components V_X, V_Y, V_Z , the interconsistency constraint $\text{Con}[\psi](S', S'')$ has to be rewritten into constraint $\text{Con}[\psi]_{\psi'}(S', V_X, V_Y, V_Z)$. This constraint relates the states of V_X, V_Y, V_Z with the state of S' that has exactly the same components: $\text{Con}[\psi]_{\psi'} = \{\langle x, y, z \rangle, x, y, z \mid x, y, z \in \mathbb{R}^+\}$. Note that it is optional to include the restriction $xyz = 1$ in $\text{Con}[\psi]_{\psi'}$, since that is already expressed by $\text{Con}[\psi']$ as per Example 32, as well as $\text{St}S'$.

Now, $\text{Con}[\psi]_{\psi'}$ can be replaced with the conjunction of three looser (but still quaternary) constraints that each only check one of the three axes: $\text{Con}[\psi]_{\psi'} = R_X \cap R_Y \cap R_Z$ where $R_X(S', V_X, V_Y, V_Z)$ is the relation $R_X = \{\langle x, y, z \rangle, x, y', z' \mid x, y, z, y', z' \in \mathbb{R}^+\}$ and analogously for R_Y and R_Z . Then, each of these three subconstraints can be restricted to two models: e.g. as R_X actually ignores $y'::V_Y$ and $z'::V_Z$, it can be replaced with binary constraint $R'_X(S', V_X)$ specified as the relation $R'_X = \{\langle x, y, z \rangle, x \mid x, y, z \in \mathbb{R}^+\}$; similarly $R'_Y(S', V_Y)$ and $R'_Z(S', V_Z)$. This demonstrates that $\text{Con}[\psi]_{\psi'}$ is binary definable.

As all of the binary subconstraints $R'_X(S', V_X), R'_Y(S', V_Y)$ and $R'_Z(S', V_Z)$ are functional relations, they can be maintained by lenses; this is how they are depicted in Fig. 7b.

7.4 Decomposition of synchronizers

Now we are getting ready to introduce the key contribution of Sect. 7: a formal notion of *faithfully* decomposing synchronizers into component synchronizers.

Take a composite model M that is affected not just by a consistency constraint $R(M, N_1, N_2, \dots)$, but also an MX that maintains R . In this case, a decomposition ψ of M into $M_1, M_2, \dots$ will rewrite not just R (into R_ψ) but also the MX (to maintain R_ψ). Such a rewritten MX will operate on the models affected by R_ψ . It will have consistency restorers that react to changes in each N_i (assuming the original MX did); apart from the other N_j , they will also read and write $M_1, M_2, \dots$ (instead of M , as translated by ψ). It will also have a consistency restorer that reacts to (at least $\text{Con}[\psi]$ -conformant) changes in $M_1, M_2, \dots$, translate them back to M using ψ , and then read/write N_i as the original MX would have.

Such a direct rewriting is not what we are aiming for, though. As Sect. 7.3 mentioned and Example 34 illustrated, the rewritten constraint R_ψ is typically replaced with a collection of simpler subconstraints that are jointly equivalent with it. Each of those subconstraints can be maintained by its own MX (or BX in case of binary subconstraints). On the surface, it appears to be a prudent assumption that the collection of these MX will end up producing the same result as the original MX did, since they jointly maintain an equivalent consistency relation anyway. The most important point of Sect. 7 is that, unfortunately, this is not necessarily true.

Example 35 In the water tank-related decomposition example of Example 34, S is decomposed (in two steps) into components S' , V_X , V_Y , V_Z . In the second step, the binary interconsistency constraint $\text{Con}[\psi](S', S'')$ (which was a mirror constraint and hence maintainable by a replicator BX $S' \rightleftharpoons S''$) is rewritten as $\text{Con}[\psi]_{\psi'}(S', V_X, V_Y, V_Z)$. Instead of maintaining this latter quaternary relation, we are interested in maintaining subconstraints $R'_X(S', V_X)$, $R'_Y(S', V_Y)$ and $R'_Z(S', V_Z)$. As all three are binary relations, they will be maintained by BX; moreover, as the relations are functional, the BX maintaining them will in fact be lenses $X : S' \rightsquigarrow V_X$, $Y : S' \rightsquigarrow V_Y$, $Z : S' \rightsquigarrow V_Z$ (see Fig. 7b). The task is to find such a span of lenses; or actually just the PUT parts, since GET is trivially fixed by the consistency relations. Note that this is exactly the original water tank problem of Example 9 (except that S is now called S'). Several solutions (of varying quality) are presented in Sect. 3; are there any of them where the 3-span of lenses work just like the replicator $S' \rightleftharpoons S''$?

If the three engineers X , Y , Z jointly come up with a new state of model S'' , modifying dimensions x , y , z in a consistent way, the replicator $S' \rightleftharpoons S''$ will simply overwrite S' with it. If the same three engineers take the same $\langle x, y, z \rangle :: S''$, and (translating by ψ') update each of V_X , V_Y , V_Z in some order, will the span of lenses $\langle X, Y, Z \rangle : S' \rightsquigarrow (V_X, V_Y, V_Z)$ lead to the same final state in S' ?

No matter which solution of the water tank problem we prefer, the answer is negative: by Corollary 2, it will not be a controllable span, so there is no guarantee of reaching the desired final state. In this sense, the span $\langle X, Y, Z \rangle : S' \rightsquigarrow (V_X, V_Y, V_Z)$ is not a *faithful decomposition* of the replicator BX $S' \rightleftharpoons S''$ for any of the solutions.

The above example demonstrated the need for a concept of *faithfulness* in the decomposition of synchronizers. Intuitively, we need to be able to say that no matter in which order we choose to propagate the updates of component models, the invoked component synchronizers will not overwrite changes made by each other, at least provided that the performed updates are *internally consistent* (correspond to a valid composite model). There is another, more straightforward requirement that the final result of propagation in both directions must also work in accordance with the composite

synchronizer. (Example 35 did not highlight this latter clause, as composite constraint $S' \rightleftharpoons S''$ was a lens in both directions, and hence, its consistency automatically guaranteed this.)

We are ready to make this concept of faithfulness formal now. For the sake of simplicity, we are limiting the definition below to the case of binary constraints, as that will be all we need in the rest of the paper. Intuitively generalizing it to the multiary case is fairly straightforward, but an actual formal treatment is left for future work. Likewise for brevity, we will only give the definition for a single decomposition, not a sequence of decompositions affecting different models.

Definition 24 (*Faithful decomposition of BX*) Take models A, B and BX $\langle \overrightarrow{R}, \overleftarrow{R} \rangle : \overleftarrow{R}(A, B)$ maintaining $R(A, B)$; also take a decomposition ψ of A into $A_1, A_2, \dots, A_n$; assume that there is a collection of *component BXs* $\langle \overrightarrow{R}_k, \overleftarrow{R}_k \rangle : \overleftarrow{R}_k(A_k, B)$ for indices $k \in [1, n]$ maintaining *external* constraints $R_k(A_k, B)$ between each component model and the external model B , as well as a collection of additional *internal* consistency constraints $T_u(A_{v_u^1}, \dots, A_{v_u^n})$ (optionally with BX/MX to maintain them) between component models, such that the conjunction of all external and internal constraints R_k and T_u is equivalent to (jointly satisfied by the same multimodel tuples as) the conjunction of the rewritten original binary constraint R_ψ and the interconsistency constraint $\text{Con}[\psi]$. The component BXs plus the internal constraints are said to be a *faithful decomposition* of the original BX, if for each initial consistent composite state $\langle a_I, b_I \rangle \in R$, the following three laws hold:

In: For any updated external state $b_D :: B$ and any index $i \in [1, n]$, we have $\overleftarrow{R}_i(\psi_i(a_I), b_D) = \psi_i(\overleftarrow{R}(a_I, b_D))$
 Cross: For any permutation $\pi \in \mathbb{S}_n$ of the components $\{1 \dots n\}$ and desired composite state $a_D :: A$ with the denotation $a_D^k := \psi_{\pi(k)}(a_D)$ (similarly for a_I^k), and any pair of indices $1 \leq i \leq j \leq n$, we have $a_D^i = \overleftarrow{R}_{\pi(i)}(a_I^i, b_j)$ where

$$\begin{aligned} * - 6pt b_j &:= \overrightarrow{R}_{\pi(j)}(a_D^j, \overrightarrow{R}_{\pi(j-1)}(a_D^{j-1}, \\ &\dots \overrightarrow{R}_{\pi(2)}(a_D^2, \overrightarrow{R}_{\pi(1)}(a_D^1, b_I)))) \end{aligned}$$

Out: For any permutation $\pi \in \mathbb{S}_n$ of the components $\{1 \dots n\}$ and desired composite state $a_D :: A$, we have final state $b_n = \overrightarrow{R}(a_D, b_I)$ where b_n is defined as above (for the special case of $j = n$).

Remark 17 The restorers of the BX/MX maintaining the internal constraints T_u play no role in any of the three laws, apart from (jointly with the external constraints) implementing the original constraint and the interconsistency constraint

of the decomposition. This also means that for given internal constraints, arbitrary restorers can be chosen to maintain them, without any influence on whether or not the decomposition is faithful.

Remark 18 If the BX $\langle \vec{R}, \overleftarrow{R} \rangle : \overleftrightarrow{R} (A, B)$ is actually a lens $(\text{GET}, \text{PUT}) : B \rightsquigarrow A$, the “In” condition is redundant.

Example 36 Following up the file synchronization example of Fig. 6 and Example 33, V_A can be decomposed into file granularity components V_A^1, V_A^2, V_A^3 . The binary consistency relation R_A of lens A is rewritten and simplified into three binary consistency relations $Q_A^1(S, V_A^1)$ for file 1, $Q_A^2(S, V_A^2)$ for file 2, and $Q_A^3(S, V_A^3)$ for file 3. Each of these are functional relations, so they can be maintained by lenses. Figure 6a shows such lenses $\langle A^1, A^2, A^3 \rangle : S \rightsquigarrow (V_A^1, V_A^2, V_A^3)$ composed from pairs of restorers such as $\langle s_1, s_2, s_3, s_4, s_5 \rangle. \text{GET}_{A^1} := s_1$ with $\langle s_1, s_2, s_3, s_4, s_5 \rangle. \text{PUT}_{A^1}(s'_1) := \langle s'_1, s_2, s_3, s_4, s_5 \rangle$. Are A^1, A^2, A^3 faithfully decomposing the original lens A ?

The law “In” holds, since these component lenses project from any state of S the same final value onto V_A as would the composite A . As for “Cross”, if we pick a state of V_A , decompose it into states of V_A^1, V_A^2, V_A^3 (i.e. individual files), then PUT_{A^i} will restore the consistency of one component lens in a way that subsequent applications of the other two component lenses will not invalidate, regardless of order. Finally, “Out” is satisfied as, after restoring consistency with all three component states, we reach the same state of S as if we applied PUT_A directly.

Similarly, lens B can be faithfully decomposed into B^2 and B^3 , and C into C^1 and C^4 .

Finally, at this stage, the source model S can be decomposed into file granularity components S^1, S^2, S^3, S^4, S^5 (see Fig. 6b). The constraints enforced by the component lenses can be rewritten and simplified into mirror constraints, e.g. S^3 must have the same value as V_A^3 and V_B^3 . These are maintained by the replicators $S^3 \rightleftharpoons V_A^3$ and $S^3 \rightleftharpoons V_B^3$ etc. As above, we can easily check that the decomposition of A^1 into $S^1 \rightleftharpoons V_A^1$ id faithful (since other components of S are ignored), while $S^2 \rightleftharpoons V_A^2$, respectively, $S^3 \rightleftharpoons V_A^3$ faithfully decompose A^2 , respectively, A^3 , and similarly for the other component lenses.

In the end, the original span $\langle A, B, C \rangle : S \rightsquigarrow (V_A, V_B, V_C)$ was faithfully decomposed (in multiple steps) into nothing but replicators.

Remark 19 While the component BXs can reproduce all behaviour of a BX they faithfully decompose, they can even produce behaviour that the original composite BX was not capable of. For instance, it is possible in a given starting situation to execute some external consistency restorers in the inwards direction and some in the outwards direction. This allows the decomposed MX to resolve the situation where

both internal and external models have been updated, as long as the updates jointly satisfy the simplified component consistency constraints affecting the modified component models. In this way, our decomposition formalism can help merge concurrent updates (c.f. [35]).

For instance, in the decomposed file synchronization example of Example 36, the component replicators can propagate changes and restore consistency (while preserving user intent) after concurrent updates that modify A^2 and B^3 (but not e.g. A^2 and B^2) at the same time. This is not possible with the original 3-lens, where feet A and B being concurrently updated would pose a problem: choosing either PUT_A and then GET_B or PUT_B and then GET_A would undo one of the two changes. We can conclude that decomposing the MX allowed more fine-grained change tracking and better merging of concurrent updates.

Remark 20 Example 35 disproves faithfulness by lack of controllability; furthermore, the similarity between the definitions of faithfulness (Definition 24) and controllability (Definition 20, but especially the more general variant Definition 38 from Appendix C) is inescapable. It is left as an exercise for the reader to work out that controllability is implied by faithful decomposition. More precisely, the span of lenses $\langle A_1, A_2, \dots, A_n \rangle : S \rightsquigarrow (V_1, V_2, \dots, V_n)$ must be controllable in the case that there exists an “observable part” lens $O : S \rightsquigarrow V$ such that the span is a faithful decomposition of O along some decomposition of V into $V_1, V_2, \dots, V_n$. The In condition is redundant as per Remark 18, and the Cross law directly spells out controllability. The additional Out condition further requires that after executing PUT_O of all feet in the span, the final state of S must be independent of the order of the feet (and consistent with PUT_O); as controllability does not require a unique final state, faithfulness is stricter in this regard.

8 Results on very well-behaved MX

8.1 Defining very well-behaved (VWB) synchronizations

As discussed in Sect. 4, there are no existing formal techniques to characterize those MX that “work like a file synchronization service”, in terms of showing predictable behaviour. We introduce the following definition, generalizing the notion of VWB BX (see Definition 7), to formally capture exactly those MX that display such high degree of regularity. While the current paper mostly focuses on spans of lenses as the simplest interesting case of MX, the definition below is equally applicable to arbitrary MX networks.

Definition 25 (*Very well-behaved (VWB) MX*) An MX is very well behaved, VWB for short, if it is possible to faithfully

decompose it (in any number of steps) into a collection of synchronizers consisting exclusively of replicators. These latter are called the *underlying* replicators of the VWB network.

To unpack, this definition means that all original models can be decomposed into parts, some of which are shared with some of the other original models, and there are no other constraints or propagation mechanisms. If this property holds, it greatly simplifies reasoning about the behaviour of the MX.

Example 37 In the file synchronization example of Fig. 1, the 3-wide span $\langle A, B, C \rangle : S \rightsquigarrow (V_A, V_B, V_C)$ is clearly conducive to decomposing the source model and each view model into a tuple of files stored there, and simply connecting the local copies of each file to its server replica using replicator lenses (see Fig. 6b). The faithfulness of the decomposition is demonstrated in Example 36. Thus the original span is VWB. The questions discussed in Example 1 can be answered just by considering which model parts each model decomposes into.

Remark 21 Why are replicators required, instead of any bijective lens? In case of no more than two component models, there is no difference: any bijection can be reinterpreted as a replicator by relabelling the model states of one of the components and patching the decomposition function accordingly. In general, however, bijective lenses may form cycles of 3 or more models, where their respective bijections would not compose to identity. In these cases, there may be additional hidden constraints (components may be pairwise but not globally consistent), and no way to eliminate them via relabelling.

Remark 22 It is quite clear that for the special case of BX, the VWB property as formalized in Definition 25 coincides with the existing BX-specific concept of VWB (“constant complement”) as formalized in Definition 7.

A further advantage of a network being VWB is that it allows us to consider it in simpler terms, by treating all mirror copies of a component model as a single *logical model (part)*. Where distinction is necessary, we may call individual model fragments as *physical model (part)s* to distinguish them from a logical model, which is the equivalence set of all physical models transitively connected by replicators.

Definition 26 (*Logical model*) Logical models are the connected components of the graph induced by the underlying replicators of a VWB network.

In light of the notion of logical models, we can describe a VWB MX network as a bipartite “has access” relationship between participants (plus shared models) and logical models.

Example 38 In the file synchronization example of Fig. 1, the logical model parts are the files 1, 2, 3, 4, and 5. For instance, file 1 has physical copies at S , V_A and V_C .

8.2 Characterizing VWB synchronizations

While VWB MXs have advantages, it may be difficult to tell in practice whether a given MX is VWB or not, since there are infinitely many possible ways to decompose the models involved. Focusing on the case of spans of lenses, here we present a full characterization of VWB spans in terms of simpler properties that are typically much easier to check (or find counterexamples for).

For bidirectional synchronizations, by Proposition 1 the VWB property simply coincides with history ignorance. As demonstrated by Example 14, this is not the case for multidirectional transformations. The intuitive reason for the discrepancy is that history ignorance restricts the behaviour of each foot of a multiary lens, but says nothing about how the feet relate to each other. However, Sect. 5.2 introduces three additional properties to accompany history ignorance, which cover this gap exactly.

Using these axioms of regularity, we can now present the second main result of the paper (proof delegated to Sect. 10).

Theorem 2 (Characterization of VWB synchronizers) *A span of lenses is VWB iff it is regular.*

We note that the above characterization of VWB in terms of the four axioms of regularity cannot be simplified, since by Corollary 1 none of the four axioms of regularity are implied by the other three.

This theorem allows us to more easily determine whether or not a given span is VWB. First, it gives us a straightforward decision procedure of exhaustively checking each of the axioms of regularity (instead of quantifying over the infinite space of possible decompositions). Second, in the negative case, the theorem allows us to present a compact witness of VWB failure: e.g. history ignorance can be refuted by a sequence of two updates at a foot; minimal interference can be refuted by a pair of updates from two feet; a pair of source states and a single update from a foot would form a counterexample to freedom from leaks; and finally, three source states may demonstrate a violation of the ZX property (i.e. difunctionality). Third, one might be able to show that a certain MX is VWB using the above theorem, even when a decomposition into the underlying replicators is not immediately apparent. The theorem also gives us insight into the structure of VWB spans, potentially allowing us to understand why a given design may not be VWB and where it could be improved.

Remark 23 Theorem 2 has applicability beyond a single span of lenses. It immediately generalizes from a span of lenses

to a span of arbitrary BXs, since any history ignorant BX decomposes into a co-span of lenses as per [31]. The result is also useful for checking whether an entire MX network is VWB, since the spans at each participant making up the network can, one by one, be decomposed into underlying replicators by repeated applications of Theorem 2.

8.3 Discussion

We have formally generalized the notion of VWB from BX to MX and provided a simple characterization of VWB synchronizations in terms of the four axioms of regularity.

An optimistic reading of Theorem 2 is that once we ensure the four axioms of regularity, we receive guarantees that the span of lenses will behave extremely nicely, all of its effects being easily predictable.

The pessimistic reading of Theorem 2 is that it shows the futility of trying to automatically enforce non-trivial consistency rules in an MX setting: unless it is a simple “data synchronization” task of copying content back and forth, the MX is guaranteed to violate at least one of the axioms of regularity. No matter how carefully we implement the MX, it will either show history-dependent behaviour on one foot or across multiple feet (respectively, lack of history ignorance or lack of minimal interference), or else it will fail at accomplishing predictable information hiding statically or dynamically (respectively, lack of projectional difunctionality or freedom from leaks). It may even fail at being controllable.

As discussed in Sect. 6.3, a possible takeaway message is to accept that automatic MX mechanisms should only enforce the simplest, most low-level notions of consistency (all participants are up to date with shared knowledge) as permitted by the concept of VWB. If we insist on the axioms of regularity being satisfied, automatic enforcement of high-level consistency constraints (that does not simply decompose into replicators) cannot be expected.

A potentially redeeming quality of Theorem 2 is that it allows us localize and isolate effects that violate regularity. By using the formal framework introduced in Sect. 7, we can understand an original MX to be faithfully represented by multiple simpler MXs that operate on component models. Such insight might be useful even if not all of the resulting MX/BX are replicators. Such component MXs can be individually studied to learn which of the axioms of regularity they violate and how. It is then a matter of deciding whether these specific kinds of regularity violations are acceptable in context.

8.4 Towards the proof

Before proving Theorem 2, we first need Sect. 9 to introduce considerable mathematical machinery that the proof

will rely on. In particular, Sect. 9.1 will introduce basic concepts, including a normal form for lenses; Sect. 9.2 defines operations on the lenses forming a span; and finally, Sect. 9.3 proves that these operations span a mathematical structure called Boolean algebra [20].

Building upon the interim results of Sect. 9, in Sect. 10 we use well-known properties on Boolean algebras [19] to construct a decomposition of a regular span into replicators, thereby demonstrating the VWB property.

9 A Boolean algebra of lenses

The final tool we need to construct the Proof of Theorem 2 in Sect. 10 is to develop an understanding of a Boolean algebra naturally induced by lenses of a span.

In Sect. 9.1, we introduce a relationship between lenses in a span: $A \supseteq B$ expresses that the lens A extracts all of the *information* from the source that lens B extracts. This relationship will be used to capture an equivalence class of lenses extracting exact same amount of information, only differing by labelling the states of the view model differently. From each such class, we pick a single representative lens, constructed as the *normal form* of lenses in the class.

In Sect. 9.2, we introduce operations performed on lenses (or, rather, equivalence classes of lenses) that add further lenses to a span. Applying a technique with equivalence sets laid out in [31], we define the *complement* $\bar{A}$ of a lens A as another lens that extracts from the source exactly those pieces of information that A does not. For any pair of feet A, B , we also define $A \cup B$ as a lens that extracts all information that any of A or B extracts. Similarly, $A \cap B$ is a lens that extracts just the amount of information that is extracted by both A and B , while $A \setminus B$ is a lens that extracts all information that is extracted by A but not B . We show that assuming the original span satisfies the four axioms of regularity, the above constructions are well defined and themselves satisfy the four axioms.

In Sect. 9.3, we use Huntington’s theorem [20] to show that these operations form a Boolean algebra.

9.1 Subviews, equivalent lenses, and normal forms

As stated above, the goal is to define the binary relation $A \supseteq B$ among lenses A, B with the same source, in order to capture the case where the lens A extracts all of the *information* from the source that lens B extracts. In such a case, the output of B can be computed (and updated) from the output of A , without directly consulting the source S . The form of this computation is actually an additional lens, which yields B when postfixed to A . So we have to first introduce the notion of lens composition below (from e.g. [17]).

Definition 27 (*Lens composition*) For the lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : V_A \rightsquigarrow V_B$, we define the lens $B \circ A = \langle \text{GET}_\circ, \text{PUT}_\circ \rangle : S \rightsquigarrow V_B$ as $\text{GET}_\circ := s \mapsto s.\text{GET}_A.\text{GET}_B$, $\text{PUT}_\circ := s, b \mapsto s.\text{PUT}_A(s.\text{GET}_A.\text{PUT}_B(b))$.

It is easy to see that the above-defined $B \circ A$ is indeed a lens (i.e. PutGet and GetPut are satisfied). Moreover, composition is associative: $C \circ (B \circ A) = (C \circ B) \circ A$.

Example 39 In the file synchronization example of Fig. 1, we can construct a lens $L = \langle \text{GET}_L, \text{PUT}_L \rangle : V_A \rightsquigarrow V_B$ with restorers $\text{GET}_L = \langle x, y, z \rangle \mapsto \langle x, y \rangle$ and $\text{PUT}_L = \langle x, y, z \rangle, \langle x', y' \rangle \mapsto \langle x', y', z \rangle$. In this case, the composite lens $L \circ A = B$.

We can now use lens composition to formally define the relationship between a superview and a subview: we say that a view extracts all the information from the source that another view does, if and only if the subview can be expressed as an updateable view on the superview. (This is called *sublens* or *lens containment* in [17].)

Definition 28 (Superview–subview relationship) Take the lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$. We say that B is a subview of A , denoted as $A \supseteq B$, iff it is possible to construct an *interpolating* lens $L : V_A \rightsquigarrow V_B$ such that $L \circ A = B$.

Example 40 In the file synchronization example of Fig. 1, $A \supseteq B$ as witnessed by the interpolating lens L constructed for Example 39.

By observing the way lens composition works, one finds that an obvious necessary condition of $A \supseteq B$ is that the value of lens B can be computed as a function of the value of A . More necessary conditions follow:

Lemma 7 If $A \supseteq B$, then A is free from leaks via B , and B is free from leaks via A , and the two are projectionally difunctional.

Proof B is free from leaks via A as a trivial consequence of its value being a function of the value of A . So is projectional difunctionality.

To show that A is free from leaks via B , take any states $s, s' :: S$ with $s.\text{GET}_A = s'.\text{GET}_A = a$. We have to prove that for any $b :: V_B$, $s.\text{PUT}_B(b).\text{GET}_A = s'.\text{PUT}_B(b).\text{GET}_A$. By the assumption of the subview relationship, an interpolating lens L must exist such that:

$$\begin{aligned} s.\text{PUT}_B(b) &= s.\text{PUT}_A(s.\text{GET}_A.\text{PUT}_L(b)) \\ &= s.\text{PUT}_A(a.\text{PUT}_L(b)) \end{aligned}$$

If we substitute $a' := a.\text{PUT}_L(b)$, apply GET_A on both sides, and use PutGet, we find:

$$s.\text{PUT}_B(b).\text{GET}_A = s.\text{PUT}_A(a').\text{GET}_A = a'$$

The exact same holds for s' . $\square$

Does the conjunction of some of the necessary conditions above guarantee that a subview relationship holds? Not in general, as PUT_A and PUT_B may act on S in incompatible ways. However, we present a sufficient condition to guarantee that any interpolating view is an updateable view:

Lemma 8 If the lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$ are minimally interfering and A is free from leaks via B and there exists a function $f : \text{St}A \rightarrow \text{St}B$ such that $f \circ \text{GET}_A = \text{GET}_B$, then $A \supseteq B$.

Proof The subview relationship will be witnessed by an interpolating lens $\langle \text{GET}_L, \text{PUT}_L \rangle = L : V_A \rightsquigarrow V_B$, where $a.\text{GET}_L := f(a)$, and $a.\text{PUT}_L(b) := s.\text{PUT}_B(b).\text{GET}_A$ for an arbitrary $s :: S$ such that $s.\text{GET}_A = a$. Here, PUT_L is well defined due to the assumption that A is free from leaks via B . Given that GET_L indeed identifies the single B -value that is cross-consistent with a given value of A , the pair indeed satisfies GetPut and PutGet, and hence a lens. We only need to check whether $L \circ A = B$ holds.

It follows from definitions that GET restorers compose:

$$s.\text{GET}_A.\text{GET}_L = f(s.\text{GET}_A) = s.\text{GET}_B$$

So what needs to be proved is that PUTs also compose, for any initial source state $s :: S$ and any update $b' :: B$.

Let $a' := s.\text{PUT}_B(b').\text{GET}_A$. At the same time, PutGet makes $s.\text{PUT}_B(b').\text{GET}_B = b'$; thus, a', b' are cross-consistent; hence, $f(a') = b'$. This also means that $s.\text{PUT}_A(a').\text{GET}_B = b'$. By applying GetPut:

$$s.\text{PUT}_B(b') = s.\text{PUT}_B(b').\text{PUT}_A(s.\text{PUT}_B(b').\text{GET}_A)$$

Here, we first recognize a' then apply minimal interference:

$$s.\text{PUT}_B(b') = s.\text{PUT}_B(b').\text{PUT}_A(a') = s.\text{PUT}_A(a').\text{PUT}_B(b')$$

Remembering that $s.\text{PUT}_A(a').\text{GET}_B = b'$, GetPut allows us to simplify:

$$s.\text{PUT}_B(b') = s.\text{PUT}_A(a').\text{PUT}_B(b') = s.\text{PUT}_A(a')$$

Recognize also that $a' = s.\text{GET}_A.\text{PUT}_L(b')$, leading to:

$$s.\text{PUT}_B(b') = s.\text{PUT}_A(a') = s.\text{PUT}_A(s.\text{GET}_A.\text{PUT}_L(b'))$$

Which is exactly how composition of PUT is defined. $\square$

As any lens can be composed with a replicator, $\supseteq$ is reflexive. Due to the associativity of composition, $\supseteq$ is transitive. This makes $\supseteq$ a *preorder* among lenses from the same source.

It is not a *partial order* though, as it lacks the anti-symmetric property: there exist pairs of different lenses $A : S \rightsquigarrow V_A$ and $B : S \rightsquigarrow V_B$ with $A \supseteq B$ and $B \supseteq A$ both being true; we denote this case as $A \cong B$. Informally, $\cong$ expresses that two lenses extract the same amount of information from the source model (merely presenting it differently), and update the source model the same way if the extracted view is updated.

Such pairs with $A \cong B$ are easily constructed by taking any lens $A : S \rightsquigarrow V_A$, and (uniquely) relabelling the states of its target model V_A into a new model space V_B . On the other hand, $A \cong B$ implies a roundtrip of interpolating GET functions, which must therefore be the inverses of each other. In other words, $A \cong B$ iff a bijective (replicator) lens can interpolate among them. Replicator lenses reverse and compose to replicator lenses, making $\cong$ an equivalence relation (see also [17]).

We can naturally contract the preorder $\supseteq$ into a partial order among $\cong$ -equivalence classes. In the sequel, we will identify such equivalence classes by selecting a representative lens from each class. In fact, we will introduce a construction that assigns to each lens the representative of its class, and call it the *normal form* of the lens. This normal form of the lens will focus on the information extracted from the source model by identifying which states of S can be distinguished through lenses in the equivalence class. In particular, the normal form lens will relabel the output states of the original lens to the partition of source model states that are all mapped to the same value by the original lens (the “forward equivalence set” of [31]).

Definition 29 (*Normal form of lens*) For lens $A = \langle \text{GET}_A, \text{PUT}_A : S \rightsquigarrow V_A \rangle$, we define its normal form lens as $[A] = \langle \text{GET}_{[A]}, \text{PUT}_{[A]} \rangle : S \rightsquigarrow S/\text{GET}_A$ with restorers $\text{GET}_{[A]} := s \mapsto \{s' \mid s.\text{GET}_A = s'.\text{GET}_A\}$ and $\text{PUT}_{[A]} := s, \{s', \dots\} \mapsto s.\text{PUT}_A(s'.\text{GET}_A)$.

In the above definition, the target model space S/GET_A consists of the GET_A -equivalent partitions of $\text{St}S$.

It is easy to check that (a) $\text{PUT}_{[A]}$ is well defined, since the second argument (the new value of the view) is a non-empty set of states of S , all with the same GET_A -value; (b) $[A]$ is indeed a lens (i.e. PutGet and GetPut are met); (c) $A \cong [A]$ since GET_A can trivially interpolate between the two lenses; and finally (d) $A \cong B$ implies $[A] = [B]$. In sum, the normal form is truly a unique representative lens of the $\cong$ -equivalence class of lenses.

Example 41 In the file synchronization example of Fig. 1, the normal form $[A]$ will map a source state to all source states that agree with it on the contents of those files that are extracted by lens A , i.e. files 1, 2 and 3. This makes $\langle x, y, z, v, w \rangle.\text{GET}_{[A]} = \{\langle x, y, z, v', w' \rangle \mid v', w' \in \mathbb{F}\}$. At the same time, $\text{PUT}_{[A]}$ just takes the contents of the files 1, 2,

and 3 (on which all source states returned by $\text{GET}_{[A]}$ agree) and executes PUT_A with them.

Example 42 In the water tank problem of Example 9, the normal form $[X]$ will map a source state to all source states that agree with it on dimension X , i.e. $\langle x, y, z \rangle.\text{GET}_{[X]} = \{\langle x, y', 1/xy' \rangle \mid y' \in V_Y\}$. At the same time, $\text{PUT}_{[X]}$ just takes the extent in dimension X (on which all source states returned by $\text{GET}_{[X]}$ agree) and executes PUT_X with it.

The operations of the Boolean algebra, to be introduced in Sect. 9.2, will be directly formulated on individual lenses from a single source S , but will conceptually map the above introduced $\cong$ -equivalence classes to $\cong$ -equivalence classes. This congruence will be made manifest by defining the operations in a way that (a) they always yield a lens in its normal form, and (b) lens target models are uninterpreted save for equality, or in other words, the operations will always be expressible using only the normal forms of the input lenses.

9.2 Operations on lenses

Here, we are going to define a collection of operations on lenses from the same source (more precisely, on their $\cong$ -equivalence sets). We will also state and prove some important lemmas on the behaviour of these constructions. Most importantly, that for a span of lenses that is regular, these operations produce new lenses that likewise exhibit these properties.

9.2.1 Complementation

Before formally defining our first operation, the complement of a lens, we have to first consider the notion of *orbits*. While defined separately, we will soon see that in case of history ignorant lens, the orbits coincide with the backwards equivalence sets of Sect. 2.2.4 and [31]. In the context of a history ignorant lens, the orbit of a state of the source model is the set of source model states that are reachable from the original state by updating the source model using the lens.

Definition 30 (*Orbit*) The orbit of a source state $s :: S$, with respect to a history ignorant lens $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$, is $\text{Orb}_A(s) := \{s.\text{PUT}_A(a) \mid a :: V_A\}$, or in a different but equivalent form, $\text{Orb}_A(s) := \{s.\text{PUT}_A(s'.\text{GET}_A) \mid s' :: S\}$.

We make the following immediate observations:

- Each source state is contained by its own orbit. (By the GetPut law, updating with the result of GET_A leaves them fixed.)
- Due to history ignorance, the source model stays within the same orbit after any number of updates using A . In other words, orbits are *closure sets*, and the value of the orbit function $\text{Orb}_A()$ is invariant under PUT_A .

- Orbits form a partitioning of $\mathbf{St}S$: if source states $s_1, s_2 :: S$ can be moved to the same state s by PUT_A , then by history ignorance $s.\text{PUT}_A(s_1.\text{GET}_A) = s_1$ and the same for s_2 , demonstrating that both s_1 and s_2 are within the orbit of s as well as the orbit of s_1 and s_2 .
- As a consequence of the above, an orbit contains exactly those source states that have it as their orbit.
- All orbits are of the same size, since the elements of the orbit correspond one-to-one to $\mathbf{St}V_A$ (easily distinguishable using GET_A).
- $s.\text{PUT}_A(a)$ is thus entirely determined by a and $\text{Orb}_A(s)$, since there is exactly one element in the orbit that is generated by (and thus projecting to) a .
- Consequently, source states that have the same orbit must also belong to the same backwards equivalence class BackEq_A . But by definition of the latter, source states in the same backwards equivalence class trivially have the same orbit. In sum, orbits and backwards equivalence classes are the same.

We can now formally define the complement of a lens, obtained by erasing from the source model all information that is extracted by the lens. This erasure is performed by overwriting this view by all possible values, and taking the set of resulting source model states—in other words, the orbit. If two different states of the source model agree on the complement (i.e. the orbit), then they only differ in ways that are overwriteable via the lens.

Definition 31 (*Complement of lens*) For history ignorant lens $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$, we define its complement lens $\bar{A} = \langle \text{GET}_{\bar{A}}, \text{PUT}_{\bar{A}} \rangle : S \rightsquigarrow S/\text{PUT}_A$ using the restorers $\text{GET}_{\bar{A}} := s \mapsto \text{Orb}_A(s) = \{s.\text{PUT}_A(a) \mid a :: V_A\}$, and $\text{PUT}_{\bar{A}} := s, \{s', \dots\} \mapsto s'.\text{PUT}_A(s.\text{GET}_A)$.

Here, the state space of the target model S/PUT_A consists of the A -orbits in $\mathbf{St}S$. It is easy to check the following:

- $\text{PUT}_{\bar{A}}$ is well defined because its second argument is such an orbit $\text{Orb}_A(s')$, a non-empty set of source states that react identically to $\text{PUT}_A(a)$ for any $a :: V_A$. Therefore, the meaning of the definition does not depend on which s' is picked as representative from the same orbit.
- The complement is indeed a lens, i.e. meets PutGet and GetPut .
- The complement lens is invariant under relabelling of A , is in normal form, and thus can be considered to map a $\cong$ class to a $\cong$ class.
- Due to the properties of orbits, $\text{GET}_{\bar{A}}$ is invariant under PUT_A .
- Due to PutGet on A , GET_A is invariant under $\text{PUT}_{\bar{A}}$.

Example 43 In the file synchronization example of Fig. 1, the complement $\bar{A}$ will map a source state to all source states

that agree with it on the contents of all files except those writable through lens A , i.e. agree on files 4 and 5. Thus $\langle x, y, z, v, w \rangle.\text{GET}_{\bar{A}} = \{\langle x', y', z', v, w \rangle \mid x', y', z' :: \mathbb{F}\}$. At the same time, $\text{PUT}_{\bar{A}}$ just takes any single source state returned by $\text{GET}_{\bar{A}}$ (all of which agree on the contents of the files 4 and 5) and injects the GET_A -projection of the current source state via PUT_A to ensure that files 1, 2, and 3 keep their current value. This is just the normal form of a lens that would extract files 4 and 5 from S , which is exactly the complement of files visible to A .

Example 44 The PROPORTIONAL solution of Example 11 is history ignorant, so it has complements. $\bar{X}$ will map a source state to all source states reachable from it via PUT_X , i.e. $\langle x, y, z \rangle.\text{GET}_{\bar{X}} = \{\langle x', y\sqrt{x/x'}, z\sqrt{x/x'} \rangle \mid x' :: V_X\}$. At the same time, $\text{PUT}_{\bar{X}}$ just takes any single source state returned by $\text{GET}_{\bar{X}}$ (all of which agree on the ratio y/z) and injects the X -projection of the current source state via PUT_X (which likewise preserves the ratio y/z) to ensure that x keeps its current value. This is just the normal form of a lens that would extract the ratio y/z from S . While this form of the complement lens might not be immediately obvious, it is not a surprise, since X is independent from (non-interfering with) y/z and together they uniquely determine the source state.

Example 45 On the other hand, in the context of the likewise history ignorant CLOCKWISE solution of Example 12, complementation will give bizarrely interesting results. Since PUT_X just updates Y to match the old Z and new X , one might expect that $\bar{X} = Z$. This is not true, however. While $\text{GET}_{\bar{X}}$ will indeed be a relabelling of GET_Z , $\text{PUT}_{\bar{X}}$ will work differently from PUT_Z : it will preserve X instead of Y . In other words, $\bar{X}$ is ($\cong$ with) the ANTI-CLOCKWISE twin of lens Z .

Does a complement lens satisfy the axioms of regularity? It directly follows that the lens $\bar{A}$ is (a) history ignorant, (b) minimally interfering (in fact, non-interfering) with A , and (c) mutually leak-free with A . However, we can prove even more than that: we can show that enlarging a span with the complement of one of its lenses will preserve regularity. (We will make use of the fact that regular spans are also cross-hippocratic, proved in Sect. 5.2.5.)

Lemma 9 $\bar{A}$ is free from leaks via any B if A and B are minimally interfering.

Proof Take any $b :: V_B$ and assume $s_1, s_2 :: S$ are in the same A -orbit:

$$\text{Orb}_A(s_1) = s_1.\text{GET}_{\bar{A}} = s_2.\text{GET}_{\bar{A}} = \text{Orb}_A(s_2)$$

The goal is to prove $s_1.\text{PUT}_B(b)$ and $s_2.\text{PUT}_B(b)$ also share A -orbits. Let $a_* :: A$ be an A -value cross-consistent with b .

$$\text{Orb}_A(s_1.\text{PUT}_B(b)) = \{s_1.\text{PUT}_B(b).\text{PUT}_A(a) \mid a :: V_A\}$$

This orbit includes, for the case $a = a_*$, the value $s_1.PUT_B(b).PUT_A(a_*)$ which by minimal interference equals $s_1.PUT_A(a_*).PUT_B(b)$. Similarly, the orbit of $s_2.PUT_B(b)$ includes the state $s_2.PUT_A(a_*).PUT_B(b)$. These two must be the same state, since s_1 and s_2 being in the same orbit implies that $s_1.PUT_A(a_*) = s_2.PUT_A(a_*)$. Thus, on this state, the orbits of $s_1.PUT_B(b)$ and $s_2.PUT_B(b)$ intersect, meaning they are in fact the same orbit. $\square$

Lemma 10 *Any B is free from leaks via $\bar{A}$ if A and B are projectionally difunctional and A is cross-hippocratic towards B .*

Proof Assume s_1 and s_2 with $b := s_1.GET_B = s_2.GET_B$. For any $\bar{a}$ with $s'_1 := s_1.PUT_{\bar{A}}(\bar{a})$ and $s'_2 := s_2.PUT_{\bar{A}}(\bar{a})$, we have to show $s'_1.GET_B = s'_2.GET_B$.

As witnessed by s_1 and s_2 , both $a_1 := s_1.GET_A$ and $a_2 := s_2.GET_A$ are consistent with b . Since $PUT_{\bar{A}}$ is ineffectual on A , $a_1 = s'_1.GET_A$ and $a_2 = s'_2.GET_A$. Since A and B are projectionally difunctional, and $b'_1 := s'_1.GET_B$ is apparently consistent with a_1 , it must also be consistent with a_2 (see the application of the ZX rule illustrated in Fig. 8a).

Thus by cross-hippocraticness,

$$s'_1.PUT_A(a_2).GET_B = s'_1.GET_B = b'_1$$

By definition, s'_1 and s'_2 are in the same A -orbit; therefore, $s'_1.PUT_A(a_2) = s'_2.PUT_A(a_2)$; by GetPut, the latter further equals s'_2 . We thus get $s'_2.GET_B = b'_1$, which we set out to prove. $\square$

Lemma 11 *$\bar{A}$ is minimally interfering with any B if A and B are minimally interfering and A is free from leaks via B .*

Proof Take $s :: S$ with $a := s.GET_A$, $\bar{a} := s.GET_{\bar{A}}$, $b := s.GET_B$. Assume that updates $\bar{a}'$, b' are cross-consistent as witnessed by $s' :: S$ such that $a' := s'.GET_A$, $\bar{a}' := s'.GET_{\bar{A}}$, $b' := s'.GET_B$. Since an A -orbit and the value of GET_A together uniquely identify a state of S , it is sufficient to prove that the two final states $s_1 := s.PUT_{\bar{A}}(\bar{a}').PUT_B(b')$ and $s_2 := s.PUT_B(b').PUT_{\bar{A}}(\bar{a}')$ agree both on GET_A and $GET_{\bar{A}}$.

Recall that $PUT_{\bar{A}}$ preserves the value of GET_A . Applying on s , we have the following:

$$s.PUT_{\bar{A}}(\bar{a}').GET_A = s.GET_A$$

Since A is free from leaks via B , this GET_A -congruence persists after applying a B -update to both sides:

$$s.PUT_{\bar{A}}(\bar{a}').PUT_B(b').GET_A = s.PUT_B(b').GET_A$$

Recognize s_1 on the left-hand side. As before, applying $PUT_{\bar{A}}$ on the right-hand side preserves GET_A :

$$s_1.GET_A = s.PUT_B(b').PUT_{\bar{A}}(\bar{a}').GET_A$$

Recognizing s_2 on the left-hand side, we get $s_1.GET_A = s_2.GET_A$

Since $s.PUT_{\bar{A}}(\bar{a}')$ is $GET_{\bar{A}}$ -congruent with s' , and this relationship (by Lemma 9) is preserved by PUT_B , we have $s.PUT_{\bar{A}}(\bar{a}').PUT_B(b') = s_1$, in turn, being $GET_{\bar{A}}$ -congruent with $s'.PUT_B(b')$. The latter is of course just s' due to GetPut. So we have the following congruence:

$$s_1.GET_{\bar{A}} = s'.GET_{\bar{A}} = \bar{a}'$$

The latter further equals $s.PUT_B(b').PUT_{\bar{A}}(\bar{a}').GET_{\bar{A}}$ due to PutGet, which is just $s_2.GET_{\bar{A}}$. This concludes the proof. $\square$

Lemma 12 *$\bar{A}$ is projectionally difunctional with any B if A is cross-hippocratic towards B .*

Proof Assume states s_1, s_2, s_3 such that $s_1.GET_{\bar{A}} = s_2.GET_{\bar{A}}$ and also $s_2.GET_B = s_3.GET_B$. As illustrated in Fig. 8b, we have to show the existence of a state s with $s_1.GET_B = s.GET_B$ and $s.GET_{\bar{A}} = s_3.GET_{\bar{A}}$.

We will show that this holds for $s := s_3.PUT_A(s_1.GET_A)$. First, PUT_A is ineffectual on $\bar{A}$, so the latter goal is already met. Second, $s.GET_A = s_1.GET_A$ demonstrates cross-consistency between $s_1.GET_A$ and $s.GET_B$; therefore, A being cross-hippocratic towards B dictates the following:

$$s.GET_B = s_2.PUT_A(s_1.GET_A).GET_B$$

Fig. 8 Visual aid in applying the ZX property in proofs

By the assumption, here $s_2.\text{PUT}_A(s_1.\text{GET}_A) = s_1$, since s_1 and s_2 only differ in A and not $\bar{A}$. This completes the proof. $\square$

Lemma 13 $\bar{A}$ is projectionally difunctional with any B if B is free from leaks via A .

Proof Recalling that freedom from leaks implies cross-hippocraticness as per Lemma 1, the result directly follows from Lemma 12. $\square$

After dealing with the axioms of regularity, we introduce two more important lemmas.

Lemma 14 $\bar{A}$ is anti-monotonic: if $A \supseteq B$ for history ignorant and minimally interfering $A : S \rightsquigarrow V_A$ and $B : S \rightsquigarrow V_B$, then $\bar{B} \supseteq \bar{A}$ also holds.

Proof To prove the subview relationship, we are going to apply the sufficient condition of Lemma 8: show that (a) $\bar{B}$ is free from leaks via $\bar{A}$, (b) $\bar{A}$ and $\bar{B}$ are minimally interfering, and (c) that there is a function f that computes the value of $\bar{A}$ from the value of $\bar{B}$.

Since $A \supseteq B$, by definition there must exist an interpolating lens $I : V_A \rightsquigarrow V_B$ with $I \circ A = B$. Also by Lemma 7 A and B are mutually free from leaks via each other and projectionally difunctional. At the same time, we can apply Lemma 11 to learn that $\bar{A}$ is minimally interfering with B . Symmetrically for $\bar{B}$, it is minimally interfering with A . Thus by Lemma 9, $\bar{B}$ must be free from leaks via $\bar{A}$ since B and $\bar{A}$ are minimally interfering. This was condition (a).

From Lemma 1, we know that A and B are also mutually cross-hippocratic. Lemma 10 thus implies that B is free from leaks via $\bar{A}$. Since we have also found these two to be minimally interfering, applying Lemma 11 again finally tells us that $\bar{A}$ and $\bar{B}$ are minimally interfering. This was condition (b).

Condition (c) requires an interpolating function $f : V_{\bar{B}} \rightarrow V_{\bar{A}}$ such that $f \circ \text{GET}_{\bar{B}} = \text{GET}_{\bar{A}}$. Such a function f , by definition, must be able to identify $\text{Orb}_A(s)$ using $\text{Orb}_B(s)$ only, without knowing s . For each $s :: S$ and $b :: V_B$, the lens composition $I \circ A = B$ implies that the effect of $s.\text{PUT}_B(b)$ can be achieved via $s.\text{PUT}_A$ for some A -value. Therefore, $\text{Orb}_B(s)$ is simply a subset of $\text{Orb}_A(s)$. Since exactly one A -orbit can contain the non-empty $\text{Orb}_B(s)$, it is possible for f to identify $\text{Orb}_A(s)$ based on $\text{Orb}_B(s)$. $\square$

Lemma 15 Complementation is its own inverse: $\lceil A \rceil = \overline{\bar{A}}$.

Proof Follows immediately by expanding the definition. $\square$

9.2.2 Union

It is very straightforward to define a view (model part) that combines all information extracted by two other views.

Under the right conditions, such a view is updateable (hence a lens): a union lens that combines all information extracted by two other lenses (as long as they are minimally interfering with each other)². A possible way to implement such a lens would be to extract pairs consisting of the views of the two original lenses. The following definition is slightly different, as it is the normal form of the above construction.

Definition 32 (Union of lenses) For minimally interfering lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$, we define their union $A \cup B = \langle \text{GET}_{A \cup B}, \text{PUT}_{A \cup B} \rangle : S \rightsquigarrow S/(\text{GET}_A \times \text{GET}_B)$ using the restorers $\text{GET}_{A \cup B} := s \mapsto \{s' \mid s' :: S \wedge s'.\text{GET}_A = s.\text{GET}_A \wedge s'.\text{GET}_B = s.\text{GET}_B\}$, and $\text{PUT}_{A \cup B} := s, \{s', \dots\} \mapsto s.\text{PUT}_A(s'.\text{GET}_A).\text{PUT}_B(s'.\text{GET}_B)$.

Here, the state space of the target model $S/(\text{GET}_A \times \text{GET}_B)$ consists of the partitions in $\text{St}S$ with equivalent GET_A and GET_B . It is easy to check the following:

- Due to the assumption of minimal interference, $\text{PUT}_{A \cup B}$ and hence $A \cup B$ is symmetric in A, B .
- $A \cup B$ meets PutGet and GetPut, making it a lens.
- $A \cup B$ is in normal form and is invariant under relabelling of A or B .

Example 46 In the file synchronization example of Fig. 1, the union lens $A \cup C$ will map a source state to all source states that agree with it on the contents of all files available through either A or C , i.e. agree on files 1 through 4. Thus $\langle x, y, z, v, w \rangle.\text{GET}_{A \cup C} = \{\langle x, y, z, v, w' \rangle \mid w' :: \mathbb{F}\}$. At the same time, $\text{PUT}_{A \cup C}$ just overwrites these files by a combination of PUT_A and PUT_C . This is just the normal form of a lens that would extract files 1 through 4 from S , which is exactly the union of files visible to A and C .

Do union lenses satisfy the axioms of regularity? It immediately follows from definitions that (a) $A \cup B$ is minimally interfering with any lens $C : S \rightsquigarrow V_C$ that is minimally interfering with both A and B ; likewise (b) $A \cup B$ is free from leaks via any lens C via which A and B are free from leaks; and finally any lens C is free from leaks via $A \cup B$ if it is also free from leaks via A and B . In order to show that the union operation also preserves the regularity of the span, we will present similar results on history ignorance and projectional difunctionality, requiring slightly longer proofs:

Lemma 16 $A \cup B$ is history ignorant if A and B are both history ignorant.

² This operation was called *lens sum* in [17]; here, we extend the definition to any pair of minimally interfering lenses (instead of requiring non-interference).

Proof History ignorance holds iff the first term can be eliminated from:

$$s.\text{PUT}_{A \cup B}(\{s'_1, \dots\}).\text{PUT}_{A \cup B}(\{s'_2, \dots\})$$

Substituting $a_1 := s'_1.\text{GET}_A$, $a_2 := s'_2.\text{GET}_A$, $b_1 := s'_1.\text{GET}_B$, and $b_2 := s'_2.\text{GET}_B$, the above formula expands to:

$$s.\text{PUT}_A(a_1).\text{PUT}_B(b_1).\text{PUT}_A(a_2).\text{PUT}_B(b_2)$$

By the minimal interference of A and B at $\langle a_2, b_2 \rangle$ (which are cross-consistent as evidenced by s'_2), this equals:

$$s.\text{PUT}_A(a_1).\text{PUT}_B(b_1).\text{PUT}_B(b_2).\text{PUT}_A(a_2)$$

By the history ignorance of B , this further equals:

$$s.\text{PUT}_A(a_1).\text{PUT}_B(b_2).\text{PUT}_A(a_2)$$

Applying again minimal interference at $\langle a_2, b_2 \rangle$ yields:

$$s.\text{PUT}_A(a_1).\text{PUT}_A(a_2).\text{PUT}_B(b_2)$$

By the history ignorance of A , this gives

$$s.\text{PUT}_A(a_2).\text{PUT}_B(b_2)$$

By definition, this is $s.\text{PUT}_{A \cup B}(\{s'_2, \dots\})$. $\square$

Lemma 17 $A \cup B$ is projectionally difunctional with any C that is projectionally difunctional with both A and B , if A and B are both free from leaks via C .

Proof Assume states s_1, s_2, s_3 with $s_1.\text{GET}_C = s_2.\text{GET}_C$ and $s_2.\text{GET}_{A \cup B} = s_3.\text{GET}_{A \cup B}$. The latter is equivalent to $s_2.\text{GET}_A = s_3.\text{GET}_A$ and $s_2.\text{GET}_B = s_3.\text{GET}_B$. The goal, as illustrated by Fig. 8c, is to show the existence of a state s with $s_3.\text{GET}_C = s.\text{GET}_C$ and $s.\text{GET}_{A \cup B} = s_1.\text{GET}_{A \cup B}$. Again, the latter can be shown on A and B separately.

Projectional difunctionality between C and A (which, when depicting the ZX property on a scatter plot, would look very similar to Fig. 8c except having A instead of $A \cup B$) implies the existence of an s_A with $s_3.\text{GET}_C = s_A.\text{GET}_C$ and $s_A.\text{GET}_A = s_1.\text{GET}_A$. For B , projectional difunctionality implies the existence of s_B with $s_3.\text{GET}_C = s_B.\text{GET}_C$ and $s_B.\text{GET}_B = s_1.\text{GET}_B$, respectively. As A and B are both leak-free via C we have:

$$\begin{aligned} s_A.\text{PUT}_C(s_A.\text{GET}_C).\text{GET}_A &= s_1.\text{PUT}_C(s_A.\text{GET}_C).\text{GET}_A \\ s_B.\text{PUT}_C(s_B.\text{GET}_C).\text{GET}_B &= s_1.\text{PUT}_C(s_B.\text{GET}_C).\text{GET}_B \end{aligned}$$

Recognizing that $s_A.\text{GET}_C = s_B.\text{GET}_C = s_3.\text{GET}_C$, we can substitute $s := s_1.\text{PUT}_C(s_3.\text{GET}_C)$ into both right sides. On

the other hand, by GetPut, the left sides are just $s_A.\text{GET}_A = s_1.\text{GET}_A$ and $s_B.\text{GET}_B = s_1.\text{GET}_B$. Finally, we obtain:

$$s_1.\text{GET}_A = s.\text{GET}_A$$

$$s_1.\text{GET}_B = s.\text{GET}_B$$

In other words,

$$s_1.\text{GET}_{A \cup B} = s.\text{GET}_{A \cup B}$$

At the same time, by PutGet and by the definition of s , we have:

$$s_3.\text{GET}_C = s.\text{GET}_C$$

Thus s so constructed demonstrates the ZX property between $A \cup B$ and C . $\square$

After addressing regularity, we introduce three additional important lemmas on unions.

Lemma 18 *Unions are inflationary:* $A \cup B \supseteq A$.

Proof By the properties of the union operator discussed above, $A \cup B$ is free from leaks via A and minimally interfering with A . So to apply Lemma 8, we only have to construct f with $f \circ \text{GET}_{A \cup B} = \text{GET}_A$ to ensure subview relationship. This is trivially satisfied by choosing $f := \{s', \dots\} \mapsto s'.\text{GET}_A$. $\square$

Lemma 19 *For pairwise minimally interfering A, B, C with $C \supseteq A$ and $C \supseteq B$, we also have $C \supseteq A \cup B$.*

Proof By Lemma 8, C is free from leaks via A and B . By the properties of the union operator discussed above, C is free from leaks via, and minimally interfering with, $A \cup B$. So to apply Lemma 8, we only have to construct f with $f \circ \text{GET}_C = \text{GET}_{A \cup B}$. This is trivially met by $f := c \mapsto \{s \mid s::S \wedge s.\text{GET}_A = c.\text{GET}_{I_A} \wedge s.\text{GET}_B = c.\text{GET}_{I_B}\}$. $\square$

As a corollary of these results, we can show that the sub-lens partial order $\subseteq$ exactly matches the union operator.

Lemma 20 *For minimally interfering lenses A, B , we have $A \subseteq B$ iff there is a lens C with $B \cong A \cup C$.*

Proof If $A \subseteq B$, then $B \cong A \cup B$ because (i) $B \subseteq A \cup B$ due to Lemma 18, and (ii) $B \supseteq A \cup B$ because Lemma 19 applies as $B \supseteq A$ and $B \supseteq B$.

The reverse direction is just Lemma 18, already proved. $\square$

9.2.3 Composite operations

Finally, we quickly introduce a pair of composite operations as notational shorthands:

Definition 33 (*Difference of lenses*) For suitable lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$, we define their difference lens $A \setminus B := \overline{(\overline{A} \cup B)}$ if it exists.

Definition 34 (*Intersection of lenses*) For suitable lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$, we define their intersection lens $A \cap B := A \setminus \overline{B} = \overline{(\overline{A} \cup \overline{B})}$ if it exists.

Note that by Lemma 15, it also holds that $A \setminus B = A \cap \overline{B}$.

Example 47 In the file synchronization example of Fig. 1, $A \cap C = \overline{(\overline{A} \cup \overline{C})}$ is a lens that extract only file 1 from the source model, while $A \setminus C = \overline{(\overline{A} \cup C)} = [B]$ is a lens that extracts only files 2 and 3 from the source model, and is therefore $\cong$ with B .

Remark 24 By expanding the definitions of these nested operations (at least GET), we learn how they actually work; below we display simplified forms after eliminating repeated updates using history ignorance.

$$\begin{aligned} \text{GET}_{A \cap B} &= s \mapsto \{s''.\text{PUT}_B(s'.\text{PUT}_A(s.\text{GET}_A).\text{GET}_B) \mid s', s'' :: S\} \\ \text{GET}_{A \setminus B} &= s \mapsto \{s'.\text{PUT}_A(s.\text{GET}_A).\text{PUT}_B(s''.\text{GET}_B) \mid s', s'' :: S\} \end{aligned}$$

A closer look reveals $\text{GET}_{A \cap B}$ to identify all B -values compatible with $s.\text{GET}_A$, and $\text{GET}_{A \setminus B}$ to identify all $\overline{B}$ -values compatible with $s.\text{GET}_A$.

The above expanded definitions of $\text{GET}_{A \cap B}$ and $\text{GET}_{A \setminus B}$ both manifestly depend on s only through $s.\text{GET}_A$. This gives the hint that both of these composite lenses are subviews of A ; we can formalize this as the following lemma, which will be important later.

Lemma 21 *Differences and intersections are deflationary: for minimally interfering and history ignorant lenses $A = \langle \text{GET}_A, \text{PUT}_A \rangle : S \rightsquigarrow V_A$ and $B = \langle \text{GET}_B, \text{PUT}_B \rangle : S \rightsquigarrow V_B$, we have $A \supseteq A \setminus B$. (As the intersection with B is just the difference with $\overline{B}$, the same goes for intersections.)*

Proof By Lemma 18, $\overline{A} \subseteq \overline{A} \cup B$. By applying Lemma 14, Lemma 15, Lemma 16: $[A] = \overline{(\overline{A})} \supseteq \overline{(\overline{A} \cup B)} = A \setminus B$. Since $A \cong [A]$, this concludes the proof. $\square$

9.3 Booleanness

We have established that regular spans are closed under the above lens operations, see Lemmas 9, 10, 11, 13, 16, 17

as well as immediate observations under Definitions 31 and 32. This also implies that the lens operations (including the composite ones) are always applicable on feet of a regular span.

However, we have still not shown that they obey any algebraic rules. Boolean algebra has a large number of axioms [19], so individually proving them would be tedious. We instead take a shortcut enabled by the following theorem by Huntington [20,21]:

Proposition 3 (Huntington's theorem) *A set closed under unary operation $\overline{X}$ and binary operation $X \cup Y$ is a Boolean algebra if the following three independent conditions are met for all elements of the set A, B, C :*

1. $A \cup B = B \cup A$
2. $A \cup (B \cup C) = (A \cup B) \cup C$
3. $(\overline{A} \cup \overline{B}) \cup (\overline{A} \cup B) = A$ (or, using our notational shorthand: $(A \cap B) \cup (A \setminus B) = A$)

Theorem 3 *The $\cong$ -classes of lenses, which are obtained by repeated applications of the operations of complementation and union, on an initial set of generator lenses that constitute a regular span, form a Boolean algebra.*

Proof The first two Huntington axioms follow immediately from the definition of $\cup$, so we focus our attention on the third and main axiom. We will separately prove $(A \cap B) \cup (A \setminus B) \subseteq A$ and $(A \cap B) \cup (A \setminus B) \supseteq A$, which will together show $\cong$, thus the equality of equivalence classes.

The $\subseteq$ case follows immediately from Lemma 21 and Lemma 19.

For the $\supseteq$ case, by Lemma 8 and preservation of regularity we only have to prove that an interpolating function exists. In essence, we have to show that the values $s.\text{GET}_{A \cap B}$ and $s.\text{GET}_{A \setminus B}$ uniquely determine $s.\text{GET}_A$.

As discussed before, $s.\text{GET}_{A \cap B}$ and $s.\text{GET}_{A \setminus B}$ extract the B -values, respectively, the $\overline{B}$ -values compatible with $s.\text{GET}_A$. In other words, any source state that is GET_A -congruent with s must have a B -value in $s.\text{GET}_{A \cap B}$ and a $\overline{B}$ -value in $s.\text{GET}_{A \setminus B}$. A B -value together with a $\overline{B}$ -value will of course uniquely determine a source state. All we need to show is that all such combinations of B -values and $\overline{B}$ -values selected from $s.\text{GET}_{A \cap B}$ and $s.\text{GET}_{A \setminus B}$ will, in fact, identify a source state that is GET_A -congruent with s , as that would imply $s.\text{GET}_{[A]}$ and thus $s.\text{GET}_A$ is computable using the results of $s.\text{GET}_{A \cap B}$ and $s.\text{GET}_{A \setminus B}$ alone.

Take a source state $s :: S$. Take any $b :: V_B$ from $s.\text{GET}_{A \cap B}$, witnessed by $s_1 :: S$ such that $s_1.\text{GET}_A = s.\text{GET}_A$ and also $s_1.\text{GET}_B = b$. Likewise, take any $\overline{b} :: \overline{V_B}$ from $s.\text{GET}_{A \setminus B}$, witnessed by $s_2 :: S$ such that $s_2.\text{GET}_A = s.\text{GET}_A$ and $s_2.\text{GET}_{\overline{B}} = \overline{b}$. Note that s_1 and s_2 are GET_A -congruent; as A is assumed

leak-free via B , it follows that:

$$s_1.\text{PUT}_B(b).\text{GET}_A = s_2.\text{PUT}_B(b).\text{GET}_A$$

Recognize that by GetPut , $s_1.\text{PUT}_B(b) = s_1$, and so the left-hand side becomes $s_1.\text{GET}_A = s.\text{GET}_A$. On the other hand, $s_2.\text{PUT}_B(b)$ is exactly the unique source state characterized by b and $\bar{b}$, which must thus be A -congruent with s . This concludes the proof. $\square$

Remark 25 We can work out this above described interpolating function $f : V_{(A \cap B) \cup (A \setminus B)} \rightarrow V_A$ using, as always, the normal forms of the lenses as follows:

$$\{s, \dots\} \mapsto \{s'.\text{PUT}_B(s''.\text{GET}_B) \mid s' \in s.\text{GET}_{A \setminus B}, s'' \in s.\text{GET}_{A \cap B}\}$$

As a quick sanity check, we can plug in the formulas from Remark 24 and eliminate a redundant PUT_B using history ignorance to arrive at the following form:

$$\{s'.\text{PUT}_A(s.\text{GET}_A).\text{PUT}_B(s''.\text{PUT}_A(s.\text{GET}_A).\text{GET}_B) \mid s', s'' :: S\}$$

By cross-hippocraticness of B towards A , this latter is of course just $[s.\text{GET}_A]$.

Now that we understand our lens operators as a Boolean algebra, we can explore the structure of Boolean algebras to gain insight into a regular span.

First, we note that like all Boolean algebras, the algebra of lenses also has a zero element: $\perp$ is the trivial lens that maps all source states to a singleton set (StS itself in normal form). This zero element arises quite naturally: for any lens A in the regular span, $A \cap \bar{A} = \perp$. More generally, if $A \cap B = \perp$, we can draw the conclusion that all values of A are cross-compatible with all values of B . (The two lenses are non-interfering in the terminology of [30], independent in the terminology of [17].)

Second, Boolean algebras induce a “implies”/“contains” partial order along with the union operator; as already seen in Lemma 20, this coincides with the sublens relation $\subseteq$ of Definition 28.

10 Proof of Theorem 2 (VWB iff regular)

10.1 Proving the forward direction

Let us start with the more difficult direction of the theorem, that regularity implies VWB.

The $\cap$ and $\setminus$ operators can be used to decompose lenses in the span into smaller and smaller parts, but not infinitely small: due to the well-known conjunctive absorption law of Boolean algebras, the so-called *minterms* [19] are the smallest (non-trivial) lenses expressible using these operations. Let us see this in more detail.

Definition 35 (Minterm) For a Boolean algebra spanned by a set of initial generator objects, a *minterm* is an intersection of one object per each generator, where this object is either the generator itself or its complement.

Example 48 For instance, if we start applying our lens operations on an initial regular span $\langle A, B, C \rangle : S \rightsquigarrow (V_A, V_B, V_C)$, the eight minterms would be $A \cap B \cap C$, $A \cap B \cap \bar{C}$, $A \cap \bar{B} \cap C$, $A \cap \bar{B} \cap \bar{C}$, $\bar{A} \cap B \cap C$, $\bar{A} \cap B \cap \bar{C}$, $\bar{A} \cap \bar{B} \cap C$, and finally $\bar{A} \cap \bar{B} \cap \bar{C}$. As mentioned, intersecting a minterm with another lens does not lead to smaller (non-trivial) fragments: $A \cap \bar{B} \cap C \cap A = A \cap \bar{B} \cap C$ due to the absorption law, while $A \cap \bar{B} \cap C \cap B$ involves $\bar{B} \cap B = \perp$ and is hence $\perp$ itself.

Two remarkable properties of minterms [19] are the following:

- the intersection of any two (or more) different minterms is $\perp$;
- any value in the Boolean algebra is equal to the union of those minterms that are $\supseteq$ -contained in it.

The above observations have the following implications in the case of the Boolean algebra of lenses. First, due to the intersections being empty, the values of different minterm views can be independently chosen (they are non-interfering). Second, all ($\cong$ -equivalence classes of) lenses extract a view model that can be decomposed, without interconsistency constraint, into one component model per minterm contained in the original lens.

Example 49 For instance, in the context of the regular 3-span of Example 48, $[A] = (A \cap B \cap C) \cup (A \cap B \cap \bar{C}) \cup (A \cap \bar{B} \cap C) \cup (A \cap \bar{B} \cap \bar{C})$. This means that the values of these minterm views jointly determine the value of A ; in fact, they jointly have the same information content as A . And as any two of these minterm lenses will have an empty intersection $\perp$, their values can be chosen independently and still lead to a valid A -value; there is no non-trivial interconsistency constraint when V_A is decomposed into these components.

Further note that regularity of the span is known to be closed under lens intersections, so these minterm component lenses will themselves be minimally interfering (in fact, due to their intersection being $\perp$, non-interfering) with each other. Therefore, not only can we independently choose the values for each component model they extract, we can use the component lenses in any order to propagate the chosen values back into the source model, without having to worry about cross-effects.

These observations allow us to conclude the main proof effort, closely following the argument in Example 36 of the file synchronization case study being VWB.

Proof Take a regular n -wide span $\langle A, B, C, \dots \rangle : S \rightsquigarrow (V_A, V_B, V_C, \dots)$. The two above-mentioned properties of minterms together imply that any generator lens can be decomposed into 2^{n-1} minterms. For lens $A : S \rightsquigarrow V_A$, this involves the splitting of V_A into 2^{n-1} model parts, one per minterm $\mu \subseteq A$, each produced by its own (minterm) lens $A^\mu : S \rightsquigarrow V_{A^\mu}$ (with $\lceil A^\mu \rceil = \mu$). This actually is a model decomposition, as the contained minterms together contain all information extracted by the original generator lens. Furthermore, this decomposition of A into component lenses A^μ is faithful, as the component lenses are non-interfering (and hence would not overwrite each other).

Now the replicator $\top$ (obtained e.g. as $\overline{(\perp)}$ the complement of the zero element) decomposes into a union of all 2^n minterms, as $\mu \subseteq \top$ for all minterms μ . Since the values of $\top$ are in one-to-one correspondence with S , this involves a decomposition of S into 2^n component models S^μ , one for each minterm. Along this decomposition of the source model, we can rewrite and simplify the (minterm) component lenses A^μ of each original lens A of the span. Since different minterms have an intersection of $\perp$, only one of the components S^μ of the source model will share any information with a given component lens A^μ , the one that corresponds to the same minterm μ and is thus constrained to be equal with it. Thus each component lens $A^\mu : S \rightsquigarrow V_{A^\mu}$ is replaced by a single replicator $S^\mu \Leftarrow V_{A^\mu}$, from the single source model component S^μ that corresponds to the same minterm μ (is in the same $\cong$ -class) as the component lens A^μ . This decomposition of the component lens into the replicator again is faithful, since all components of the source model except one are ignored.

Altogether, we have a decomposition of each involved model into a finite number of components. The initial lenses will faithfully decompose (in two steps) into several replicator lenses connecting the model components that correspond to the same minterm (logical model). Since different minterms have an intersection of $\perp$, no interconsistency relation is imposed between them. This is therefore a VWB decomposition. $\square$

10.2 Proof of the reverse direction

Finally, we wrap up by proving the implication in the less interesting reverse direction: VWB implies regularity.

Proof First examine the decomposed form of the VWB span, consisting of isolated logical models, each made up of component models connected by the underlying replicators. The source model and each of the view models of the original VWB span can be understood as sets of such component models. The source model has the peculiar property of containing a component model from each logical model, while each view model may be associated with a subset of logi-

cal models. A source model state is consistent with a view model state if they agree on the state of each logical model that the view model is associated with. Two view models are cross-consistent if they agree on the states of all those logical models that they are both associated with. A GET of the original span, up to faithful decomposition, just extracts from the source model the states of the logical models associated with one view, and overwrites the state of that view with it. A PUT of the original span, up to faithful decomposition, just overwrites in the source model the states of the logical models associated with one lens, with the states of the same logical model in the view model of that lens, without affecting anything else.

From this point of view, it is apparent that all four axioms of regularity are satisfied in the original composite span:

- Projectional difunctionality between any two lenses is satisfied, as witnessed by the pair of model part functions that extract the state of the shared logical models from each view model state.
- History ignorance of any lens is satisfied, because its underlying replicators are history ignorant.
- Minimal interference between any two lenses is satisfied, as two view states being cross-consistent means that they agree on the shared logical models, and hence, the two PUT will not overwrite each others effects.
- Freedom from leaks of any lens A via any other lens B is satisfied, as the only effect of PUT_B visible in V_A is the overwriting of the logical models shared between A and B , which depends only on the new state of V_B and not the previous source state.

As all four axioms of regularity are satisfied, the original span is regular. $\square$

11 Related work

11.1 Identifying challenges in MX

One of the earliest works to raise issues specific to MXs was [32]. It examined basic questions, such as whether or not a consistent state can be reached at all. It did not consider whether the consistent state thus achieved is the desired state or not; the notion of controllability presented in this paper is therefore novel. The benefits of the VWB property are also beyond the scope of [32].

The Dagstuhl seminar [8] collected several concerns and challenges related to MX, such as the need for a synchronization policy, or that an MC might be required to guarantee “some notion of reachability of states, or progress”. The latter can be considered an informal precursor to the controllability requirement formalized in this paper.

Additional concerns were raised in [25], but that paper primarily focused on the methodology of (and sources of errors in) developing a specification for MX, while the current paper investigates properties of a concrete MX itself.

Further difficulties were addressed in [24], but the focus there was likewise different from our own. The paper examined whether the category theoretical objects describing multiary lenses (essentially, spans of lenses) compose to similar kinds of category theoretical objects and whether axioms of well behaving are preserved at all in such compositions. The current paper, on the other hand, is more concerned with the cross-effects among feet of a span.

11.2 Related solution techniques

11.2.1 Decompositions

The concept of *model parts* was introduced in [32]. Formally, no general notion of model *decomposition* was given in that paper (where the parts can jointly recreate the original model); so, in this sense our definition of decomposition is novel. But this novelty is trivial and rather unimportant, as the word “decomposition” was used by [32] in this sense. Specifically, two concepts *A/B/rest decomposition* and *tagged decomposition* were defined, which are both also (specific kinds of) decompositions in our terminology.

The relationship of the above notions of decomposition with the non-interference property can be seen as a precursor to Theorem 2. Whether or not multiary consistency constraints can be replaced with binary ones was also studied in [32].

Decomposing synchronizers (including their consistency restoration behaviour) is a new result to the best of my knowledge, at least when universally discussing models from first principles. The approach in [18] also uses decomposition of models (or rather model updates) to combine lenses in a way that is very similar to our union operation, but it is (a) less general in the sense that it applies only to specific modeling formalisms with richer structure (containment relationship between model states, three-way merge algebra), while at the same time (b) more general in the sense that it does not require the existence of a static decomposition of model states into a multimodel state tuple only that there is a well-defined three-way merge that combines update deltas to the various parts.

The notion of faithful decompositions is novel in the current paper.

11.2.2 The equivalence sets technique and 2D plotting

The current paper owes many ideas to [31], which introduced the notions of backwards and forward equivalence sets (BackEq, respectively, “blobs”), the visualization technique

of plotting the state space in two dimensions (see e.g. Fig. 2) based on these two attributes, and several important lemmas on history ignorant BX. As [31] focuses exclusively in BX, the application of similar proof techniques (and visualization approach) to MX is novel in the current paper.

11.2.3 Properties of spans

The property called “lens independence” in [17] and “non-interference” in [32] was already known in the literature. (Beyond the BX literature, many similar concepts of order independence have been floating around, e.g. sequential independence [14] in graph transformation.) The novelty in this paper is the more general property minimal interference. The two properties coincide in the case where all view states of one lens are cross-consistent with all view states of the other one. However, the concept of regularity required the more relaxed definition presented in this paper.

While projectional difunctionality is novel in this paper, difunctional sets have been applied before [23] in the context of MX, specifically co-spans of lenses.

To the best of our knowledge, cross-hippocraticness and freedom from leaks are entirely novel in this paper.

History ignorance has been studied for decades, and its generalization to MX is trivial. The VWB property (or “constant complement”) is similarly well known and equivalent to history ignorance in the BX case; however, its generalization to MX as faithful decomposition to replicators is a novel contribution of this paper.

11.2.4 Operations on lenses

Lens composition is of course not new. The sublens relationship $\supseteq$ and the induced equality classes $\cong$ have also been used before [17]. The unary operations (constant lenses) $\top$ and $\perp$ have likewise been introduced in [17], under the names “identity lens”, respectively, “unit lens”. The normal form $[\cdot]$ is novel in this paper, however.

It has been long known that history ignorant BX are “constant complement”, see e.g. [31] for a discussion using the equivalence sets method. However, introducing complementation as an operation on history ignorant lenses is, to the best of our knowledge, novel in this paper.

The “lens sum” operation of [17] (and the unnamed operation on BXs in Lemma 2 of [32]) is basically the same as the lens union operation used in this paper, except for not being in normal form. However, we have extended the applicability of the operation from non-interfering lenses (“independent” in the terminology of [17]) to minimally interfering lenses; this weakened precondition is crucial for the axioms of regularity being closed under the union operation. Unlike [32], we did not consider the case of generic BXs, only lenses.

The composite operations $\setminus$ and $\cap$ are novel contributions in this paper, as are the connections between the operations and the axioms of regularity, and the recognition of the Booleanness (in case of regularity) of the algebra spanned by the operations.

11.2.5 Synchronization policies in networks of MX

Synchronization policies (see Sect. 2.3.2 or [8,28,32]), alternatively known as execution strategies, are essentially coordination mechanisms in an MX network: algorithms that take as input some user-proposed modifications in a multimodel state tuple and generate a sequence of invocations of consistency restorers (belonging to the BXs or MXs making up the network) that hopefully lead to an entirely consistent multimodel state, hopefully in a small number of steps, hopefully while preserving user intent. At this point, they terminate.

As explained in Sect. 3.2 (in the context of the LRU solution to the water tank problem), such execution histories are typically expected to schedule consistency restoration mechanisms, not user actions, and are usually expected to terminate when a consistent state is reached, without saving any extra state (beyond that of the models). Thus customary forms of synchronization policies cannot solve the water tank problem either. However, it is possible as future work to generalize synchronization policies so that they are allowed to persist some extra information. Alternatively, this persisted information may be represented as a model, with the special limitations that no user modification of this model is allowed, and the collaboration mechanism manipulating this model is exempted from certain requirements (e.g. history ignorance).

11.3 Alternative BX/MX formalisms

11.3.1 Delta-based approaches

Instead of the state-based formalism followed in this paper, the delta-based BX or MX formalisms [10,11] represent models not as a set of discrete states, but as a category of composable *deltas* (“modification trajectories”) between states. The state-based formalism is a special case, where there is exactly one such delta between each ordered pair of states. Therefore, the delta-based formalisms, in general, may show examples of at least those difficulties/challenges/failure modes that are present in the state-based formalism. Thus one should not expect the elimination of e.g. the whack-a-mole behaviour in a given solution solely by reformulating the same span in a delta-based formalism. Likewise, theorems that generally apply to all delta-based BX/MX are also more difficult to develop than those that apply only to state-based ones (see e.g. the multicategorical framework of [24]). For these reasons, the current paper stays within the bounds of

the state-based formalism; it is left as future work to see how much of the results will generalize to delta-based formalisms.

11.3.2 Concurrent changes and amendment lenses

The BX and MX studied in this paper will take a modification proposed on one *authoritative* model and propagate it to other models; the proposed modification can be applied on the authoritative model as is. There exist alternatives both in the state-based [35] and delta-based [11] paradigms that accept modifications to multiple models and furthermore allow the BX or MX to *amend* the originally proposed modifications into a different modification (of the same model). There seems to be two main motivations behind such amendments. First, it is possible that the original proposal would take the model into an inconsistent state, but somehow this inconsistency is only detected when it is compared against other models; and the best choice seems to be amending the original change, rather than adjusting the other models accordingly. Second, if multiple models are modified concurrently, then amendment-based solutions allow these changes to be merged.

For the sake of simplicity, the formalism in this paper is restricted to non-concurrent updates and no amendments. However, as explained and illustrated in Remark 19, our notion of model decomposition allows more fine-grained localization of change. (Instead of saying that a given composite model has been updated, we can say which of its components were modified.) Concurrent changes that are contradictory on the composite model level may turn out to decompose into componentwise modifications that can be propagated freely (especially in a VWB network) without fear of overwriting each other. This may in many cases eliminate the need for explicit, lens-level handling of concurrent updates and amending the modifications to a composite model.

11.3.3 Write access control

A frequent use case of lenses is to provide read access control: make sure that a view only shows some of the information available in the source. The advanced formalism of [16] goes one step further and applies non-trivial write access control on top of it: unauthorized changes to the source via a given view are avoided, as not all valid/representable modifications of the view are acceptable to be performed on the view side. The current paper focused exclusively on read access control, while [16] does not discuss the MX case. Extending the results presented in this paper to a framework enriched with write access is left for future work.

12 Conclusions and future work

Lenses in a span (and thus MX in general) may operate in a completely unsatisfactory way, even if all individual restorers are fine in isolation. Through a case study, we have amply demonstrated one such kind of failure, the whack-a-mole behaviour, and formulated its absence as the requirement of controllability. Another novel property, cross-hippocraticness, was introduced and shown to provide a simple, compact characterization of the controllable case.

While controllability may be considered indispensable, we have also formulated three additional properties of predictable behaviour in a span that are merely “nice to have”: projectional difunctionality, minimal interference, and freedom from leaks. These restrictions are more difficult to satisfy (the latter two are strictly stronger than cross-hippocraticness) and hence imply a great degree of predictability in the behaviour of the MX. And in the case where all three of these properties hold for a span, in addition to the well-known property of history ignorance, then the span is considered regular, as it has very straightforward behaviour in every sense. Informally, a regular span works “just like a data synchronization service”, simply copying pieces of data back and forth. To formally capture this case, we have given a non-trivial generalization of the well-known concept of very well-behaved BX to the MX domain. We have also charted out how the results on spans of lenses apply more broadly to other kinds of MX networks.

Thus we have learnt that complex multiary consistency rules are challenging to automatically maintain, and we either have to settle for very well-behaved MX that only enforce low-level consistency, or tolerate that the MX will not give entirely satisfactory behaviour guarantees.

Apart from the main conclusions, the paper also developed interesting techniques, such as the notion of faithful decomposition of MX, or Boolean operations on lenses, that we believe could find their use in other contexts as well.

It will require future applications of the theoretical results in this paper to learn its practical implications. Do industrially relevant MXs typically satisfy the novel properties proposed in this paper? Hopefully, they are at least controllable. On the other hand, by Theorem 2, they cannot be VWB, unless they are uninteresting data synchronization services. However, it would be very interesting to learn in each case study which of the axioms of regularity are violated, where, and how, and especially whether the violations are reasonably tolerable.

An obvious future direction would be to see whether any of these results generalize to broader BX formalisms, such as delta-based [12] or amendment lenses [11]. Perhaps even more importantly, there is much practical demand for there to be limits on how a view is allowed to be updated; use cases range from unidirectional derivations (e.g. in build systems) to non-trivial write access restrictions [16] as parts of

larger MX networks. Furthermore, the LRU [5] solution of the case study hints at an even more general, history-based formalization, where the state of the synchronization policy is preserved across executions.

But there is plenty of work to be done even in the basic BX formalism we have adopted. Since controllability is a very basic assumption while VWB is quite a restrictive case, there might be much leeway in between; we know little of the behaviour of controllable MX that are not quite as regular as to be VWB. Finally, we have not proposed any concrete languages or analysis techniques that would aid the developer in creating spans of lenses that are controllable or regular.

Acknowledgements The author thanks Dániel Varró for his insightful comments; additionally Csaba Debreceni and István Ráth for our fruitful collaboration in fine-grained access control and collaborative modeling, resulting in [9]. The author is grateful towards Schloss Dagstuhl and specifically the organizers and participants of Dagstuhl Seminar 18491 [8] for providing much insight into multidirectional transformations.

Acknowledgements

PROJECT
FINANCED FROM
THE NRDI FUND

Author Contributions See Sects. 3.3, 4.3 and 11.

Funding Open access funding provided by Budapest University of Technology and Economics. The author was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences as well as the ÚNKP-18-4 New National Excellence Program of the Ministry of Human Capacities and the ÚNKP-19-4 New National Excellence Program of the Ministry For Innovation and Technology. The research reported in this paper and carried out at the BME has been partially supported by the NRDI Fund based on the charter of bolster issued by the NRDI Office under the auspices of the Ministry for Innovation and Technology.

Availability of data and material Not applicable.

Declarations

Competing interests Not applicable.

Code availability Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

Appendix

A Discussing the LRU solution

Here, we present one important solution [5] of the water tank problem that was only superficially discussed in Sect. 3.2. The core idea is to remember which of the three feet was the *least recently updated* (LRU), and is hence the least authoritative.

Example 50 (Water tank solution 3: LRU) When constructing PUT_X make the assumption, without loss of generality, that Y was more (or equally) recently updated than Z , i.e. X, Y, Z are ordered “most authoritative” to “least authoritative”.

$$\langle x, y, z \rangle. \text{PUT}_X(x') := \langle x', y, 1/x'y \rangle$$

LRU is both well behaved and refreshingly free of whack-a-mole. To reach a given desired source state, we have to update all three foot models to their desired state in succession, and then we are done. (It is left as an exercise for the reader to prove this.) However, there are two important drawbacks outlined below; it therefore remains true that the originally specified models with their originally specified notion of consistency do not admit a solution that avoid whack-a-mole behaviour without restricting participants.

The first weakness of the LRU solution is that it is a history-based approach that is not expressible in either state-based or delta-based formalism without expanding the source model space to incorporate more state information.

It is tempting to consider the LRU solution an “synchronization policy” (see Sect. 2.3.2), but such execution strategies are invoked to restore consistency after one or more models have been concurrently updated; they terminate with a hopefully consistent final state as their output. Whereas in the case of the Water Tank problem, whack-a-mole behaviour is manifest even if the participant engineers produce their updates several days apart, with ample time for restoring consistency in between. Thus it is not sufficient for an execution strategy to have an internal algorithmic state, but it must persist this information even after its termination (when consistency is achieved). In other words, as said above, the amount of state information stored for the models needs to be extended; it is merely a matter of perspective whether we consider the new state variables part of the models or part of the persisted state of the execution strategy.

This is how to extend the source model to preserve history: Instead of a triplet $\langle x, y, z \rangle$, the entirety of the source state would actually be $\langle x, y, z, h_1, h_2 \rangle$, where $h_1, h_2 \in \{X, Y, Z\}$ encode the identity of the foot that was most recently and second most recently updated. These two new source model parts have to be appropriately maintained.

The second weakness is that absence of whack-a-mole is only guaranteed if all three feet are updated. If one of the engineers is on vacation, the other two might experience whack-a-mole behaviour where the first of them is overwritten by the second one. For instance, starting in source state $\langle 1, 0.1, 10, Y, Z \rangle$, intending to change Y to 2 while keeping X as 1 (resulting in a Z state of 0.5) will actually fail if only PUT_X and PUT_Y are invoked, even though it should not. The reason for this is $\text{PUT}_X(1)$ will be ineffectual due to hippocraticness, and $\text{PUT}_Y(2)$ will therefore find that X was the least recently updated foot, changing it to 0.05. It is possible to fix this problem by making sure that invoking PUT_X always sets X as the most recently changed foot model, even if X did not change in value. Unfortunately, this fix makes the lenses lose their hippocraticness; while allowing such limited violations of hippocraticness is an interesting direction to pursue in the future, we consider it to be out of scope for this paper, retaining hippocraticness as one of our core assumptions.

B Other cross-effects

We have seen in Sect. 5.1 that relationships of consistency, correctness, hippocraticness, and history ignorance of single lenses can be systematically generalized into cross-effect relationships between multiple lenses of a single span. Of these new concepts, cross-consistency and cross-hippocraticness are of great importance and were discussed in the main body of text. For the sake of completeness, here we include close analogues of the cross-hippocraticness property for correctness and history ignorance. They will not have any further use in this paper.

Definition 36 (Cross-correctness) For a span of lenses $\langle A_1, A_2, \dots : S \rightsquigarrow (V_1, V_2, \dots) \rangle$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$, the foot A_i is cross-correct towards foot A_j if, for each $s :: S$ and each $a_i :: V_i$, we have that $R_{i,j}^S(a_i, a'_j)$ where $a'_j := s.\text{PUT}_i(a_i).\text{GET}_j$.

Trivially, A_i being a correct BX (i.e. cross-correctness towards $\top$) implies cross-correctness towards any other A_j . Thus this notion is of limited usefulness, as it is relevant only among non-correct BXs.

Definition 37 (History cross-ignorance) For a span of lenses $\langle A_1, A_2, \dots : S \rightsquigarrow (V_1, V_2, \dots) \rangle$ with $A_k = \langle \text{GET}_k, \text{PUT}_k \rangle$, the foot A_i is history-cross-ignorant towards foot A_j if, for each $s :: S$ and each $a_i, a'_i :: V_i$, we have that:

$$s.\text{PUT}_i(a_i).\text{PUT}_i(a'_i).\text{GET}_j = s.\text{PUT}_i(a'_i).\text{GET}_j$$

Again, ordinary history ignorance of A_j is equivalent to it being history-cross-ignorant towards $\top$. That also implies cross-ignorance towards any other foot A_j , but cross-ignorance may still be a useful distinction for otherwise history non-ignorant lenses.

C Generalizing controllability for arbitrary BX

The concept of controllability was motivated and introduced in Sect. 6.1. However, Definition 20 formally defined it only for the case of a span of lenses. Here, we present direct generalization of that definition, for the slightly more general case of a collection of BXs sharing a single model.

Definition 38 (Controllable span of BXs) Take models $A_1, A_2, \dots, A_n$, model B and a collection of BXs $\langle \overrightarrow{R_k}, \overleftarrow{R_k} \rangle : \overrightarrow{R_k}(A_k, B)$ for indices $k \in [1, n]$ maintaining constraints $R_k(A_k, B)$. The model B is said to be *controllable* through the BXs, if for any consistent initial state $b_I :: B$ plus $a_I^k :: A_k$ with $\langle a_I^k, b_I \rangle \in R_k$, and any consistent desired state $b_D :: B$ plus $a_D^k :: A_k$ with $\langle a_D^k, b_D \rangle \in R_k$, and for any permutation $\pi \in \mathbb{S}_n$ of the indexes $\{1 \dots n\}$ of A -models, and any pair of indices $1 \leq i \leq j \leq n$, we have $\langle a_D^{\pi(i)}, b_j \rangle \in R_{\pi(i)}$ where

$$b_j := \overrightarrow{R_{\pi(j)}}(a_D^{\pi(j)}, \overrightarrow{R_{\pi(j-1)}}(a_D^{\pi(j-1)}, \dots \overrightarrow{R_{\pi(2)}}(a_D^{\pi(2)}, \overrightarrow{R_{\pi(1)}}(a_D^{\pi(1)}, b_I))))$$

It is clear that Definition 20 is truly a restriction of the above definition for the special case of lenses.

D Duality through complementation

We will show how cross-hippocraticness and complementation establish a duality between two of the axioms of regularity. For a span containing lenses A, B where A is history ignorant (and hence has a complement lens $\overline{A}$) and cross-hippocratic towards B , the condition of A, B being projectionally difunctional is equivalent to B being free from leaks via $\overline{A}$.

More precisely, from Lemma 13 we know that history ignorant A is projectionally difunctional with any B if B is free from leaks via $\overline{A}$. A small deviation from the originally stated form is that here we have exchanged the roles of A and $\overline{A}$ with each other; we are free to make this cosmetic change since true complementation is its own inverse (see Lemma 15) up to “relabelling” equivalence $\cong$, which is sufficient to determine the relevant properties such as freedom from leaks.

Likewise in the context of a history ignorant A , we also know from Lemma 10 that any B is free from leaks via $\overline{A}$ if A and B are projectionally difunctional and A is cross-hippocratic towards B .

In sum, we get the following:

Corollary 3 *For history ignorant A cross-hippocratic towards B , we have that B is free from leaks via $\overline{A}$ if and only if A is projectionally difunctional with B .*

This result nicely explains how after a swap of coordinate axes, Fig. 5b and Fig. 5c look remarkably similar.

Remark 26 Projectional difunctionality is symmetric; thus if, additionally, B is also history ignorant and cross-hippocratic towards A , then the two above conditions are further equivalent to A being free from leaks via $\overline{B}$.

References

- Whac-a-mole, colloquial usage. https://en.wikipedia.org/wiki/Whac-A-Mole#Colloquial_usage. Retrieved: 26 Feb 2020
- Apache: Subversion (2018). <https://subversion.apache.org/>. Retrieved 8 Apr 2020
- Bergmann, G.: Non-difunctional / non-leak-free span of lenses v0.1 in Bx Examples Repository. <http://bx-community.wikidot.com/examples:spannondifuncleakfree>. Retrieved: 21 Feb 2020
- Bergmann, G.: Non-minimally interfering span of lenses v0.1 in Bx Examples Repository. <http://bx-community.wikidot.com/examples:spannonminint>. Retrieved: 21 Feb 2020
- Bergmann, G.: Water tank example v0.1 in Bx Examples Repository. <http://bx-community.wikidot.com/examples:watertank>. Retrieved: 3 Mar 2020
- Bohannon, A., Foster, J.N., Pierce, B.C., Pilkiewicz, A., Schmitt, A.: Boomerang: Resourceful lenses for string data. In: Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '08, pp. 407–419. ACM, New York, NY, USA (2008). <https://doi.org/10.1145/1328438.1328487>
- Bohannon, A., Pierce, B.C., Vaughan, J.A.: Relational lenses: A language for updatable views. In: Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '06, pp. 338–347. ACM, New York, NY, USA (2006). <https://doi.org/10.1145/1142351.1142399>
- Cleve, A., Kindler, E., Stevens, P., Zaytsev, V.: Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491). Dagstuhl Rep. 8(12), 1–48 (2019). <https://doi.org/10.4230/DagRep.8.12.1>
- Debreceeni, C., Bergmann, G., Ráth, I., Varró, D.: Enforcing fine-grained access control for secure collaborative modelling using bidirectional transformations. Softw. Syst. Model. (2017). <https://doi.org/10.1007/s10270-017-0631-8>
- Diskin, Z.: Algebraic models for bidirectional model synchronization. In: MoDELS, pp. 21–36 (2008). https://doi.org/10.1007/978-3-540-87875-9_2
- Diskin, Z., König, H., Lawford, M.: Multiple model synchronization with multiary delta lenses. In: Fundamental Approaches to Software Engineering, 21st International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14–20, 2018, Proceedings., pp. 21–37 (2018). https://doi.org/10.1007/978-3-319-89363-1_2
- Diskin, Z., Xiong, Y., Czarnecki, K.: From state- to delta-based bidirectional model transformations: the asymmetric case. J. Object Technol. 10(6), 1–25 (2011). <https://doi.org/10.5381/jot.2011.10.1.a6>
- Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: The symmetric case. In: Whittle, J., Clark, T. Kühne, T. (eds.) Model Driven Engineering Languages and Systems, pp. 304–318. Springer Berlin Heidelberg, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-24485-8_22
- Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Monographs in Theoretical Computer Science. An EATCS Series. Springer (2006). <https://doi.org/10.1007/3-540-31188-2>

15. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bi-directional tree transformations: A linguistic approach to the view update problem. In: Proceedings of the 32Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '05, pp. 233–246. ACM, New York, NY, USA (2005). <https://doi.org/10.1145/1040305.1040325>
16. Foster, J.N., Pierce, B.C., Zdancewic, S.: Updatable security views. In: 2009 22nd IEEE Computer Security Foundations Symposium, pp. 60–74 (2009). <https://doi.org/10.1109/CSF.2009.25>
17. Foster, S., Zeyda, F., Woodcock, J.: Unifying heterogeneous state-spaces with lenses. In: Sampaio, A., Wang, F. (eds.) Theoretical Aspects of Computing – ICTAC 2016, pp. 295–314. Springer International Publishing, Cham (2016). https://doi.org/10.1007/978-3-319-46750-4_17
18. He, X., Hu, Z.: Putback-based bidirectional model transformations. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2018, p. 434–444. Association for Computing Machinery, New York, NY, USA (2018). <https://doi.org/10.1145/3236024.3236070>
19. Hill, F.J., Peterson, G.R.: Introduction to Switching Theory and Logical Design, 3rd edn. Wiley, USA (1981)
20. Huntington, E.V.: A new set of independent postulates for the algebra of logic with special reference to whitehead and russell's principia mathematica. *Proc. National Academy Sci.* **18**(2), 179–180 (1932). <https://doi.org/10.1073/pnas.18.2.179>
21. Huntington, E.V.: Boolean algebra. A correction to: “new sets of independent postulates for the algebra of logic, with special reference to whitehead and russell's principia mathematica”. *Transactions of the American Mathematical Society* **35**(2), 557–558 (1933). <https://doi.org/10.1090/S0002-9947-1933-1501702-9>
22. Inc, N.M.: Teamwork cloud. <https://www.nomagic.com/products/teamwork-cloud>
23. Johnson, M., Rosebrugh, R.: Cospans and symmetric lenses. In: Conference Companion of the 2Nd International Conference on Art, Science, and Engineering of Programming, Programming'18 Companion, pp. 21–29. ACM, New York, NY, USA (2018). <https://doi.org/10.1145/3191697.3191717>
24. Johnson, M., Rosebrugh, R.D.: Multicategories of multiary lenses. In: Proceedings of the 8th International Workshop on Bidirectional Transformations co-located with the Philadelphia Logic Week, Bx@PLW 2019, Philadelphia, PA, USA, June 4, 2019, pp. 30–44 (2019). <http://ceur-ws.org/Vol-2355/paper3.pdf>
25. Klare, H., Syma, T., Burger, E., Reussner, R.: A categorization of interoperability issues in networks of transformations. *Journal of Object Technology* **18**(3), 4:1–20 (2019). <https://doi.org/10.5381/jot.2019.18.3.a4>. The 12th International Conference on Model Transformations
26. Nuseibeh, B., Easterbrook, S., Russo, A.: Making inconsistency respectable in software development. *J. Syst. Softw.* **58**(2), 171–180 (2001). [https://doi.org/10.1016/S0164-1212\(01\)00036-X](https://doi.org/10.1016/S0164-1212(01)00036-X)
27. Object Management Group: MOF Query/View/Transformation (QVT) (2008). <https://www.omg.org/spec/QVT>
28. Rocco, J.D., Ruscio, D.D., Heinz, M., Iovino, L., Lämmel, R., Pierantonio, A.: Consistency recovery in interactive modeling. In: Proceedings of MODELS 2017 Satellite Event, CEUR Workshop Proceedings, vol. 2019. CEUR (2017)
29. Schürr, A.: Specification of graph translators with triple graph grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) Graph-Theoretic Concepts in Computer Science, pp. 151–163. Springer, Berlin Heidelberg (1995)
30. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. In: Model Driven Engineering Languages and Systems, 10th International Conference, MoDELS 2007, Nashville, USA, September 30 - October 5, 2007, Proceedings, pp. 1–15 (2007). https://doi.org/10.1007/978-3-540-75209-7_1
31. Stevens, P.: Observations relating to the equivalences induced on model sets by bidirectional transformations. *ECEASST* **49**, 714 (2012)
32. Stevens, P.: Maintaining consistency in networks of models: bidirectional transformations in the large. *Softw. Syst. Model.* **19**(1), 39–65 (2020). <https://doi.org/10.1007/s10270-019-00736-x>
33. Systems Engineering Research Center: Research roadmaps 2019–2020 (2020). https://sercuarc.org/wp-content/uploads/2020/06/ROADMAPS_2.5.pdf
34. The Eclipse Foundation: CDO. <http://www.eclipse.org/cdo>
35. Xiong, Y., Song, H., Hu, Z., Takeichi, M.: Synchronizing concurrent model updates based on bidirectional transformation. *Softw. Syst. Model.* **12**, 89–104 (2013). <https://doi.org/10.1007/s10270-010-0187-3>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.