Least Change View-Updating for Functorial View-Gets with Left Adjoints

By

Ori Livson

A thesis submitted to Macquarie University
for the degree of Master of Research
Department of Mathematics
April 2018

Acknowledgements

I would like to foremostly express my gratitude to my supervisor Prof. Michael Johnson for his continual confidence and patience in me, as well as his outward passion for Category Theory, teaching and writing itself that inspire me fully. I also appreciate him indulging my work in progress whenever he could, amidst his busy schedule, constant travel and the many meetings afterwards he was surely made late for!

I would also like to thank my parents Ayala and Ben Livson for their belief in me and risks I took undertaking this project, along with my brother Matan for his editing suggestions and additionally my sister in law Diana for all together keeping me well fed and loved.

Finally, I would like to thank Prof. Steve Lack, Dr. Richard Garner and Dr. Gerry Myerson, all of which warmly helping me out whenever they could.

Abstract

Given a database, a view of it is a simplified version of that database, derived from some of its data, possibly through the output of query language expressions. This thesis is concerned with a category theoretic treatment of the View Update Problem, which is the problem of how to propagate a view-update to an update of the original database. The basic Category Theoretic setting of interest to this thesis is that a database has an associated category $\mathcal{S}$ whose objects are valid states that the database can be in and some choice of valid updates as morphisms. A view of the database corresponding to $\mathcal{S}$ has its own state space $\mathcal{V}$ and a functor $G : \mathcal{S} \to \mathcal{V}$, referred to as the view-get. So called *least change* view-update propagations have been studied in the categorical setting in papers stemming from [17], requiring that view-update propagations satisfy certain universal properties. This thesis studies the use of cartesian or opcartesian lifts as least change solutions to view-update problems. Moreover, the main contribution of this thesis are a pair of theorems pertaining to the existence of cartesian and opcartesian lifts respectively. The setting of these theorems involves G having a left adjoint $L \dashv G$ such that $GL = id_{\mathcal{V}}$.

Contents

Chapter. 1

Introduction

A database is typically interacted with via a simplified version of that database, whose logical structure and states are derived from those of the original database. The original database is referred to as the *source database* and the simplified database is called a *view* of it. Views are often constructed for users when only certain parts of a database are of interest to them or if they want to interact with data expressed differently, such as through the output of query language expressions. Moreover, the limited 'relevance' of a database to a user may be a matter of data security. In that case, matching a view to a user's security clearance hides the data that user shouldn't see. From a database designer's perspective, defining views for different uses of a database can modularise the logical structure of that database. Similarly, views can also make query language expressions more tractable as macros do for programming languages; an analogy between the two is drawn by Date in [8, p.298].

Two key problems can arise when using views. The first is the problem of discerning whether changes to the logical structure of the source database affect how its views function. The second is deciding, in the event a view state is updated (that is, the data contained in the view at a moment of time is updated), if and how one can update the original source state to restore synchronisation of the source with the view. This thesis studies second problem, known as the *View Update Problem* and assumes the logical structure of the source database remains fixed.

However, the view update problem isn't really a single problem because one must firstly decide what constitutes a propagation of a view-update that solves a view update problem, as a function of the application in question and how realistic finding such as solution is! For the first part of the problem, we take interest in so called *least change* propagations to view updates, more specifically, Category Theoretic analogues of least change propagations. Indeed, Category Theory has been used since the 90's to provide formal specifications, analyses and visualisations of database design. An example of an approach to these tasks is given by the use of *(mixed) sketches* from Categorical Universal Algebra; Johnson and Rosebrugh discuss its historical state of the art in [17]. Other topics in Category Theory have been drawn from and applied more generally to the study of bidirectional transformations, for which view-definitions are included; Johnson, Rosebrugh and Wood demonstrate this inclusion in [18].

We can unpack the relevant issues of least change view-updating in this thesis by firstly abstractly describing least change propagations for two kinds of view-updates:

View-Inserts: By view-insert, we mean a view-update that only inserts new data into the view state, that is, without deleting any data nor editing any existing data. In this case, a least change propagation of a view-insert is an update of the original source state that inserts the *minimal* amount of new source data required to be synchronised with the updated view state. Note, this least change propagation may have to additionally edit or delete data.

View-Deletes: By view-delete, we mean a view-update that only deletes existing data into the view state, that is, without inserting any new data nor editing any existing data. In this case, a least change propagation of a view-insert is an update of the original source state that deletes the *minimal* amount of existing source data required to be synchronised with the updated view state. Analogously, this least change propagation may have to additionally edit or insert data.

We run into our first concern when asking what a least-change view-edit propagation ought to do. This is because there are two distinct choices of how to define what an edit is. The first sense of the term "edit" we can consider is the sense the reader likely assumed: a single update of a database that modifies data values without inserting new data nor deleting existing data. We call such updates *in-place edits*. The other sense of edit involves simulating an in-place edit by: deleting the data that is to be modified then inserting the modified version of that data, which we call *simulated edits*.

A canonical choice of what a least-change propagation of a simulated edit ought to be is the composite of the least change view-delete propagation followed by the least change view-insert propagation. However, we will find that it is not obvious whether this represents a *minimal* edit to the source database so as to restore consistency to the view-edit. For example, we may expect a least change view-edit propagation to just require that some source data be modified, yet the sequence of least change delete and insert propagations may yield additional deletions or insertions of data.

Even so, reconciling the issues with simulated edits is a worthy pursuit because insert and delete updates have desirable properties, especially in the Category Theory analogues of these things. Conversely, forming a Category Theoretic account of in-place edits is difficult; a pithy summary of the issue is: if a database state X can be in-place edited to Y and visa-versa, the states typically end up isomorphic and suddenly Category Theory can't meaningfully distinguish the two states.

To summarise, when designing databases and views, attaining a means of least change propagations of view-updates is desirable so that the consequences of view-updating are as contained as possible and more predictable. However, when bringing Category Theory to bear on discerning the existence of least change view-update propagations, one has to be careful about how one defines edits of data and how one chooses to propagate view-edits.

This main contributions of this thesis are results identifying conditions on database view definitions, for which there are least change propagations of view-inserts, view-deletes and in-place edits. These results are statements of general Category Theory, so do not involve Category Theoretic constructions indicative of anything related to databasing. As such, we devote much discussion to determining which kinds of view definitions admit cases of these general results of Category Theory.

Accompanying these results pertaining to the existence of least change propagations, are results describing when and how these least change propagations can be computed. These results will suggest that discerning when and how least change view-updating can be done is generally much more tractable when we only deal with simulated edits rather than in-place edits. Moreover, we find our Category Theoretic analogue of least change propagations can break down for in-place edits.

We also find an interesting overlap between our approach to view-updating and the approach of *constant complement updating*, as categorically formulated by Johnson and Rosebrugh in [19] in reference to the original database theoretic approach of Bancilhon and Spyratos in [1]; and also propose a new approach to Categorically describing nulls, inspired by Diskin in [9].

Additionally, our main results can be ported to the theory of Category Theoretic lenses, used to study bidirectional transformations. Although, the bridge between our main results and database theory also suggest paths further studying existing Category Theoretic database specifications including the Sketch Data Model introduced in [17] and the Simplicial Data Model of Spivak in [31].

1.1 Technical Introduction

To begin to describe the above outline more technically, we introduce a shorthand that will be used throughout the thesis: Firstly, we will use the term *categorical state space* to refer to a category that can be understood as having database states as objects and updates between database states as morphisms. Secondly, we will use the term *categorical data model* to refer to a method of associating to a database state space, a categorical state space.

To understand the motivating Category Theoretic analogues of least change view-updating, we need a more detailed account of how a categorical data model should assign to a database, a categorical state space. We begin: Given a database with a fixed logical structure, we associate with it, a category $\mathcal{S}$ with objects as database states we deem valid on that logical structure. Then, for any two states $X, Y \in \mathcal{S}$, a morphism from X to Y consists of a valid update from state X to state Y.

For more specific updates: an insertion update in $\mathcal{S}$ from a state T to a state U should be given by a monomorphism $T \rightarrowtail U$. Then, a deletion update in $\mathcal{S}$ from a state T to a state S should be given by a monomorphism $T \leftarrowtail S$ as S contains the data of T not deleted by our update, so there should be a corresponding inclusion monomorphism. Hence, a simulated edit in $\mathcal{S}$ from some T' to U' is given by a span of monomorphisms $T' \leftarrowtail S' \rightarrowtail U'$, where S' is the data of T' that is not to be modified.

If a view of the database represented by $\mathcal{S}$ is updatable, that view has an associated state space category $\mathcal{V}$. A view-definition, which includes a means of mapping source states and updates to the appropriate view-states and view-updates is to be given by a functor, say $G : \mathcal{S} \to \mathcal{V}$. We refer to functors such as G as *view-gets*.

We then find there are two dual categorical statements of the view update problem, first introduced by Johnson and Rosebrugh in [17] – one for sole insertions and one for sole deletions. Given a view definition $G : \mathcal{S} \to \mathcal{V}$ and $S \in \mathcal{S}$, the former is:

Given a view-insertion $\alpha : GS \rightarrowtail T$ in $Hom_{\mathcal{V}}(GS, T)$, is there an appropriate propagation $\overline{\alpha} : S \rightarrowtail \overline{S}$ such that $G\overline{\alpha} = \alpha$?

And the latter:

Given a view-deletion $\beta : U \rightarrowtail GS$ in $Hom_{\mathcal{V}}(U, GS)$, is there an appropriate propagation $\underline{\beta} : \underline{S} \rightarrowtail S$ such that $G\underline{\beta} = \beta$?

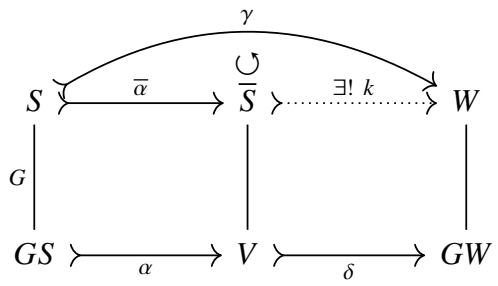
Note, that when a view is constructed out of a query that isn't monotonic (a term we will define in Chapter 3), it may not be possible to propagate a monic view-update to a monic source update at all, let alone "appropriately"[1].

For view-insertions, an answer of interest as to what "appropriate" should mean is given by the following universal mapping property:

For every other source insertion $\gamma : S \rightarrowtail W$ such that $G\gamma = \delta \circ \alpha$, for some further insertion $\delta : V \rightarrowtail GW$, there exists a unique monomorphism $k : \overline{S} \rightarrowtail W$ such that $k \circ \overline{\alpha} = \gamma$ and $Gk = \beta$

[1]For readers versed in relational database terminology, if a query is regarded as a function, that query is monotonic if it a monotonic function with respect to the inclusion of relations. Conjunctive queries (the Select-Project-Join and Equality constraints fragment of the relational model) are monotonic.

Or diagrammatically:



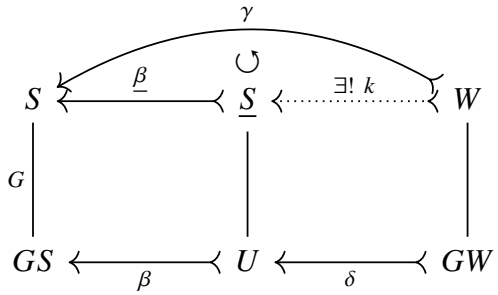
(1.1)

When one takes $GW = V$ and $\delta = id_V$, the identity on V, the statement corresponds to saying:

$\overline{\alpha}$ is the least change propagation of the view-insert α because for any other view-insert propagation γ, the existence of k tells us that W has the data of $\overline{S}$ and possibly more.

This condition is known as α being *insert-propagable*. Category Theorists may recognise the universal mapping property as that of being an *opcartesian lift of* α but with monomorphisms in place of all involved morphisms. This thesis will take interest in opcartesian propagations more generally.

The dual notion for view-deletions is that $\underline{\beta}$ is a given diagrammatically by the following (noting the reversed direction of arrows with respect to (1.1)):



(1.2)

And again taking $GW = U$ and $\beta = id_U$, we recognise $\underline{\beta}$ as being the least change propagation of the view-delete β. This condition is known as α being *delete-propagable* and has a universal property analogous to cartesian lifts. Furthermore, general view-updates $GS \leftarrowtail T \rightarrowtail V$, can be propagated to a span with a cartesian lift and an opcartesian lift as its left and legs respectively.

There is an issue of compositionality we're ignoring in this thesis. That is, for a composite of say, view-inserts $\alpha' \circ \alpha$, the question is does $\overline{\alpha' \circ \alpha} = \overline{\alpha'} \circ \overline{\alpha}$? This is an important property to strive for because otherwise the solutions to view update problems are contingent on the 'timing' of when view-updates are propagated. Least change propagations typically respect compositionality for view-inserts and view-deletes but can easily fail to respect compositionality for view-edits, Diskin studies this phenomena in [10]. Johnson and Rosebrugh study the problem of least change propagations of simulated edits in [15] and [21]; keeping in mind simulated edits are not composable in their respective categorical state spaces.

The main contribution of this thesis are general results of Category Theory, namely Theorem (4.4.2) and Theorem (4.5.5). They pertain to when opcartesian lifts and cartesian lifts exist (respectively) for functors $G : \mathcal{S} \to \mathcal{V}$ in the setting where G has a left adjoint $L \dashv G$ such that $GL = id_{\mathcal{V}}$. A common example of view-get with a left adjoint of this form are initial object preserving view-gets, with left adjoints (recall left adjoints are defined up to isomorphism). These are common since states with no user-defined data are typically initial and thus because there is no view-user defined data to derive from those states, we should expect G to be initial object preserving.

While the main Category Theoretic component of this thesis can be understood in isolation, we largely focus on the workings of a new, rudimentary categorical data model. This categorical data model is rudimentary compared to the Sketch Data Model of Chapter 3, the Simplicial Data Model of [31], the Indexed Categorical Data Model of [29], etc.

The focus on our new categorical data model is important for forming examples of view-update problems that motivate this thesis' main Category Theoretic results. Additionally, we show how the conditions for our main results can be checked explicitly in our new categorical data model and as a consequence, in existing categorical data models. Unlike the Sketch, Simplicial and Indexed Data Models, our new categorical data model allows in-place editing and as a consequence, we find examples where opcartesian and cartesian lifts don't provide as satisfying (least change) solutions to view-update problems as was described on the previous page. Checking if the conditions of Theorem (4.4.2) and Theorem (4.5.5) is also simpler when only monic updates are involved.

1.2 Thesis Structure & Assumed Knowledge

Chapter 2 introduces some preliminary concepts of the Relational Data Model, the theoretical basis for most contemporary database management systems and query languages. The concepts of particular importance we will survey are attributes, keys, schemas, ERA diagrams, Nulls and how they relate to the view update problem. However, we will not discuss any relational algebra in detail, that is, how new tables of data are typically constructed from existing tables. No prior knowledge of database theory or practice is assumed.

Chapter 3 introduces the Sketch Data Model, a categorical data model first defined in full by Johnson and Rosebrugh in [17], where the original definitions of insert and delete-propagability are given. This categorical data model will provide us with a host of examples of view-update problems. Furthermore, the Sketch Data Model's categorical state spaces disallow in-place editing. As such, the least change solutions to view-update problems we find provide important comparison for the view-updating done Chapter 4, where we allow in-place editing. It is assumed the reader knows the definitions of categories, functors, natural transformations, monomorphisms and some basic limits and colimits.

Chapter 4 derives the aforementioned main Category Theory results, along with the accompanying, new categorical data model. Here, the reader only needs to know the definitions of Hom-Set adjunctions, unit-counit adjunctions, products, pushouts, initial objects and also some basic categorical correspondences between preorders and thin categories.

Chapter 5 extends the results of chapter 4 to bear on some special topics in view-updating. The first, is *constant complement updating*; a condition on view-definitions that admits very satisfying solutions to the view-update problem – introduced by Bancilhon and Spyratos in [1] and first categorically studied by Johnson and Rosebrugh in [19]. The second, is to explore a method of refining categorical state spaces with a more general sorts of null-symbols (uncertain information), to remedy possible shortcomings of using opcartesian lifts to view-update. The approach of this topic is inspired by Diskin's broadening of uncertain data in [9], in the field of bidirectional transformations.

Chapter 6 discusses some further topics this thesis' research could be applied to. Particularly, applying our results to the study of delta lenses and also to refine the categorical data model we introduce to incorporate more topological features of database schemata, inspired by Spivak in [31].

Chapter. 2

Preliminary Concepts in Relational Database Theory

2.1 Entities, Attributes, Keys and the Relational Model

The Relational Model of data, introduced by E.F. Codd in [6] serves as the foundation for most contemporary database management systems and query languages built for them them such as SQL. A database state in the Relational Model is given by a collection of tables of data and although mathematically defining what that means is relatively simple, unpacking the terminology requires some patience. The seeming gap between the mathematics and terminology we encounter is largely a consequence of many database theoretic definitions and abstractions being motivated by concerns of implementation and human-centred interpretations of databasing that are redundant or implicitly dealt with when one's implementation is in Set Theory rather than a digital computer.

We begin by regarding a table of data as a set of rows representing a set of examples of some *Entity* such as "Persons". That table's columns refer to *Attributes* of that entity such as "FirstName", "Surname", "EyeColour", "Age", etc. Formally, each attribute is associated with a set of possible values called its *Attribute Domain*. For example, "FirstName" and "Surname" may be associated the set **Name** consisting of say, strings of *up to 50 alphabetical letters, beginning with an upper-case letter and lower-case letters everywhere else*. For "EyeColour" it may be the set $\{Blue, Brown, Green, Grey, Other\}$, $\mathbb{N}$ for "Age" and so on. Because distinct Attributes may have the same Domain such as **Name** we often distinguish *attribute headers* such as "FirstName" and "Surname" for clarity. In any case, a table of data on attribute domains $A_1, A_2, ..., A_n$ is a relation on those attributes in the set theoretic sense, that is, a subset $R \subset A_1 \times ... \times A_n$.[1] In our example thus far, a state of the $Persons$ entity is a subset $\mathbf{Persons}_0 \subset \mathbf{Name} \times \mathbf{Name} \times \{Blue, Brown, Green, Grey, Other\} \times \mathbb{N} \times ...$.

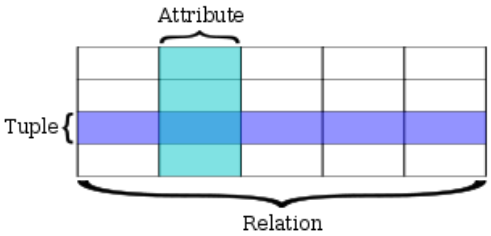


Figure 2.1: A Table Defined As A Relation From [35].

Before defining databases as somehow a collection of inter-related Entities, we need to address a confusion readers may face. This confusion is that the term "database" has been used to mean:

1. A particular state e.g. *the database consisting of:*
   $\boldsymbol{Persons}_0 \subset \boldsymbol{Name} \times \boldsymbol{Name} \times \{Blue, Brown, Green, Grey, Other\} \times \mathbb{N} \times ....$
2. The state space itself e.g. *the database that consists of an entity Persons with attributes "FirstName", "Surname", "EyeColour", "Age", etc.*
3. The data model itself e.g. *the Relational Model as our definition of a database.*

[1] In a more rigorous Category Theoretic setting, tables are jointly monic spans, whereas relations are equivalence classes of said spans. [11] discusses this in more detail.

A compromise we will follow, although unconventional, will be to use the term "*Entity-State*" to refer to the data of a state $\mathbf{Persons}_0$; therein a "*Database-State*" refers to a collection of Entity-States that satisfy the database's logical structure as in 1. We use the terms Entity and Database purely, to refer to state spaces in the sense of 2. Finally, we will use "Data Model" to refer to databases in the sense of 3.

Now we can describe inter-table or Entity-Entity relations as follows: given two entity-states, that is, relational tables R and S, an *inter-table* relation f from R to S is simply a relation $f \subset R \times S$. Extending our example, we may form an inter-table relation, $\mathbf{BirthCertificates}_0 \subset \mathbf{Persons}_0 \times \mathbf{Birth}_0$, where $\mathbf{Birth}_0 \subset \mathbf{City} \times \mathbf{Country} \times \mathbf{Date} \times \mathbf{Weight} \times \ldots$. Inter-table relations can both be thought of as entities in their own right as is natural for $\mathbf{BirthCertificates}_0$ and they can also be thought of as a procedure of sorts, for example $\mathbf{has.a}_0 \subset \mathbf{Persons}_0 \times \mathbf{InvestmentProperties}_0$ is a procedure that tells us which properties are in a person's name and which names are associated to each property.

So, we find that inter-table relations and tables aren't formally distinguished in the relational model, which is why we can define a database state as a collection of relations. In fact, attribute domains can be regarded as tables with one column – sometimes called *value tables*. However, distinguishing the abstractions: Entity, Attribute and Inter-Table Relation linguistically, can serve to make the logical structure of some databases clearer. However, not making these distinctions has a certain mathematical elegance to it. This elegance can be exploited when one defines a host of operations on existing relations that produce new relations. These operations form what is known as a *relational algebra* or *relational calculus*, for which the expressions of query languages are based on.

We now proceed to discuss the concept of *keys* in the relational model, which is another abstraction that seems to depart from the mathematical elegance of the relational model in favour of clarity and implementation in practice. The basic idea begins as follows: we note that we have no problem considering tuples in an entity-state such as $\mathbf{Persons}_0$ holistically, that is, elements of $\mathbf{Persons}_0$ and the data of whole tuples are one and the same. However, juggling tuples at a large scale in practice, one needs a low complexity encoding of tuples as a matter of space and time efficiency. Codd's lasting solution to this problem was to require that every entity has at least one attribute that uniquely identifies tuples, this can be as simple as a row-number attribute.

Formally, R needs to have an attribute K such that R is an injective, functional relation $R \subset K \times A$, where A is the product of the rest of the attribute domains. Equivalently, we require the projection $\pi : R \to K$ is injective. A distinguished choice of such a K is called a *Primary Key* of the table. Familiar examples in practice include any attribute header that ends with "ID" such as "StudentID", "PatientID" as well as "Order Number","username" etc. Primary keys can also be comprised of multiple attributes that aren't primary keys such as the concatenation of "first-name", "middle-name" and "surname" attributes. Likewise, for inter-table relations: if two entities R and S have primary keys K_R and K_S then there is a bijection between relations $f \subset R \times S$ and relations $K_f \subset K_R \times K_S$. Moreover, inter-table relations can be represented by a table having as one of its attributes, the primary key of another table; that attribute is known as a *foreign key* in the original table. Another practical motivation for keys are to efficiently prevent duplicate entries of entity-state instances; this is a concern invisible to the pure set-theoretic formulation. That is, a subset by definition can't contain two indistinguishable elements.

For the moment, we can define database updates by a collection of functions between the entity states comprising the source and target database states. For example, consider a database consisting of just the $BirthCertificates$ relation. An update from the database state $\mathbf{BirthCertificates}_0 \subset$

$\mathbf{Persons}_0 \times \mathbf{Birth}_0$ to $\mathbf{BirthCertificates}_1 \subset \mathbf{Persons}_1 \times \mathbf{Birth}_1$ can be given by a span of injections:

$$\mathbf{Persons}_0 \xleftarrow{d_p} X_p \xrightarrow{i_p} \mathbf{Persons}_1 \ , \qquad \mathbf{Births}_0 \xleftarrow{d_b} X_b \xrightarrow{i_b} \mathbf{Births}_1$$

$$\mathbf{BirthCertificates}_0 \xleftarrow{d_c} X_c \xrightarrow{i_c} \mathbf{BirthCertificates}_1$$

For $X_{(-)}$ consisting of tuples of the respective instance in the source entity-state neither edited nor deleted; so that edits of data are done by simulated data, where $i_{(-)}$ contains all additional insertions of entity instances and $d_{(-)}$ handles all additional deletions of entity-instances.

Now that we have a working definition of relational databases and updates we can proceed to discuss Views in more detail and explore an example of the View Update Problem. Some immediate concerns we will face with trying to propagate view-updating will motivate discussing Nulls and Constraints, which are features contemporary databases typically support on top of the relational model.

2.2 Views, Nulls and Constraints

We begin by recalling our understanding of a Relational Database as a collection of Entities, each associated with a set of Attribute Domains that Entity-states are relations of. A View of a Relational Database is a Relational Database whose Entity-states are relations comprising *some of the attributes of some of the tuples of some of the entity-states* of the source Database; possibly further acted on by relational algebraic operations.

We should remind ourselves that Views are still motivated by the desire in practice to form a simplified database, serving some users' needs, that only requires *some* of the logical structure of the source database and its states' data. Furthermore, that user may be interested in interacting with the data under the result of some query rather than all the raw data itself.

We then recall the first problem we described with engaging with database using views in this way. That is, discerning whether changes to the logical structure of a database affects how its views function. This problem is known as the problem of *Logical Data Independence*. Again, while not the focus of this thesis, briefly discussing Logical Database Independence may make the uses and concerns of Views more intuitive and fundamental in database theory for some readers.

Logical Database Independence is as Date describes it "*the raison d'être for views"* and *that "if we're to achieve it in practice and not just in principle, then it's clear that views have to be updatable*" [7, p. xi]. That is, the problem of Logical Data Independence and The View Update problem bear on each other in the sense that understanding what kind of Views ensure "*the immunity of users and user programs to changes in the logical structure of the database*" [8, p.299] as Date defines logical database independence, bears on the what kinds of Views solve certain View Update Problems.

Date segregates the problem of logical data independence into the *growth* and *restructuring* problems. The growth problem refers to the case that in a database's lifetime, if new entities or new attributes to entities are added, that database's views should remain intact and their data unchanged. The restructuring problem refers to the requirement that if the logical structure of a database changes over its lifetime, for example new inter-table relations are added or the entities are split up/combined together in various ways; the views remain intact in the same manner [8, p.297].

We proceed to give an example of a view-definition and analyse propagations of various view updates, in naive terms. This will prove to be less suggestive of how to attain least change view update propagations than when we begin working with Category Theory.

Example 2.2.1. (The $ContactsList$ Database):

Consider attribute domains **Name**, **Address**, **Email**, **Phone** consisting of sets of names, mailing addresses, email addresses and phone-number strings respectively. We form a database consisting of two Entities: $AddressBook$ of $Name$, $Address$ and $Email$; and $PhoneBook$ of $Name$ and $Phone$ respectively. We also include an inter-table relation $contacts$ of $AddressBook$ and $PhoneBook$ consisting of say, all pairs of tuples in $AddressBook$ and $PhoneBook$ with matching names.

Recall database states then consist of a choice of two Entity-states $\mathbf{AddressBook}_0 \subset \mathbf{Name} \times \mathbf{Address} \times \mathbf{Email}$ and $\mathbf{PhoneBook}_0 \subset \mathbf{Name} \times \mathbf{Phone}$ and a choice of relation: $\mathbf{contacts}_0 \subset \mathbf{AddressBook}_0 \times \mathbf{PhoneBook}_0$ of tuples with matching names.

It may seem strange to construct this $ContactsList$ database in the above manner instead of as a subset of $\mathbf{Name} \times \mathbf{Address} \times \mathbf{Email} \times \mathbf{Phone}$, indeed it is more intuitive to think of $AddressBook$ and $PhoneBook$ as *views* of that database. Such a reformulation of the original $contacts$ relation is a classic example of Restructuring, for which we would desire Logical Data Independence. However, a scenario yielding our original database design that isn't so dubious could be that that the relational table $contacts$ refers to the data of a mobile phone's contacts-list and that its data is constructed out of two distinct applications $AddressBook$ and $PhoneBook$ found on two different devices referred to the phone.

Of course, we should expect our database be designed so that tuples of entity-states such as $\mathbf{contacts}_0$ don't have to have all of Mailing, Email Addresses and Phone Number entries but just some combination of the three. However, we can illustrate key aspects of the view update problem without concerning ourselves with that until we discuss Nulls.

Proceeding, we form a view database consisting of a single entity $no\_Address$ of attributes **Name** **Email** and **Phone** and states given by the projection of the tuples of $ContactsList$ states. That is given $\mathbf{contacts}_0$, there is a state $\mathbf{no\_Address}_0$ given by mapping tuples $((n,a,e),(n,p)) \in \mathbf{contacts}_0$ to tuples $(n,e,p) \in \mathbf{no\_Address}_0$. We may imagine this view being motivated by some mobile application requiring your contacts data but for which you don't want to provide your contacts' mailing addresses due to privacy concerns.

View-Insert Propagations: If we insert a new tuple (n,e,p) into a state $\mathbf{no\_Address}_0$, forming a new state $\mathbf{no\_Address}_1$: the canonical propagation at the moment is to add a $contacts$ tuple $((n,a,e),(n,p))$ for some address $a \in Address$. We correspondingly need to add an (n,a,e) $AddressBook$ tuple and an (n,p) PhoneBook tuple. However, to avoid deprecating the real-world interpretation of the $ContactsList$ data, perhaps we should reserve a special address in the $Address$ attribute domain to mean "the address isn't known". However, how one implements a special value such as that (usually called a *Null*) in the attribute domain is very contentious with respect to how the relational algebra of the database ought to treat it.

View-Deletion Propagations: If we want to allow a user to delete a $\mathbf{no\_Address}_0$ tuple (n,e,p) so that deletion is reflected in the $contacts$ data, we firstly regard this update as the inclusion injection $\mathbf{no\_Address}_1 \subset \mathbf{no\_Address}_0$, where $\mathbf{no\_Address}_1$ precisely has all tuples except (n,e,p).

To propagate this view-deletion canonically, it appears one needs to delete all tuples of the form $((n, a, e), (n, p))$ in the corresponding $ContactsList$ state for any $a \in$ **Address**. Note, if we want a least change propagation, we ought not delete either tuple (n, a, e) of $AddressBook$ nor (n, p) of PhoneBook.

View In-Place Edit Propagations: Firstly, if we are to edit a $\mathbf{no\_Address}_0$ tuple (n_0, e_0, p_0) to (n_1, e_1, p_1), at this point the canonical propagation is for every $\mathbf{contacts}_0$ tuple of the form $((n_0, a, e_0), (n_0, p_0))$ in $\mathbf{contacts}_0$ for some $a \in$ **Address**, we map it to $((n_1, a, e_1), (n_1, p_1))$; we likely have to do the same for $\mathbf{AddressBook}_0$ and $\mathbf{PhoneBook}_0$.

However, if n_0 is updated to an existing name n_1, we may encounter conflicts with existing tuples with name n_1 – some readers may have encountered the inconvenience of having some of their mobile contacts merged.

Thus we can already see, depending on our choice of view-update propagations, in-place edit propagations may not match simulated edit propagations. That is, if $GS \xrightarrow{V} \alpha$ is an in-place edit corresponding to the simulated edit $GS \xleftarrow{\beta} U \xrightarrow{\gamma} V$ then the propagation $\overline{\alpha}$ of α, a source in-place edit, may not correspond to the source simulated edit $GS \xleftarrow{\overline{\beta}} U \xrightarrow{\overline{\gamma}} V$, where each leg is a propagation of β and γ, respectively.
■

Hence, we can see even for a simple specification of a database and a view, it isn't clear whether we have solved the view-update problem (whatever that means) with any of the above choices of propagations or not.

2.3 Generalisations of the Relational Data Model

We conclude this chapter by discussing generalisations to the relational model that have motivated the development existing categorical data models. We utilise nulls and constraints throughout the thesis.

Typed Databases: It is common to desire that attribute domains are typed in the sense programming languages are typed. Some reasons we may desire support for typed attributes include the desire to make more complex queries and the need to define certain constraints on the validity of data (e.g. that the *sum of some cost attributes over some tuples is less than a budget value*). It certainly isn't obvious how one formally specifies in the data model itself that an attribute domain consists of say, integers with operations one would expect on integers. A recent Category Theoretic treatment of this problem, utilising Lawvere theories is given by Spviak, et al. in [30].

Nulls: The need to represent relational tuples with certain attribute values as being missing often arises but the question of how to incorporate missing values into the relational model and particularly how relational algebras should treat Nulls is contentious. As Date puts it: "[Nulls] wreck the relational model" [8, p.591].[2] A classical survey on the topic is given by Zaniolo in [37], arguing that among the many distinct implementations of nulls into the relational model, they can be grouped into *"two basic interpretations:", "a) the known interpretation: a value exists but it is not known" and "b) the unknown interpretation: a value does not exist."* [37, p.142].

[2]Date dedicates a chapter (19 of [8]) to the topic. Some examples of how the 3-valued logic of Nulls affects the relational model are: the equality operator is no longer reflexive nor transitive, and the join operator is no longer reflexive.

We can understand an argument for why the relational model needs to be reformulated to support nulls, pithily, by noting that: if one considers a tuple without all its (attribute) values know, it is by definition *not* a tuple of a relation. So one either needs to define a data model that is a collection of tuples of data of variable length, as we'll see a categorical formulation of this in the Sketch Data Model of Chapter 3, specifically Section 5.2 – originating in [25]; and Section 6.2 originating in [31]. One can also define a data model with tuples with null portions or possible solution-sets that are distinct from tuples in how they can be updated to and fro, an idea we'll explore in section 3.5.

SQL for instance takes the known interpretation, wherein Nulls are supported by redefining each nullable attribute domain to be a sum type with a universal type/value: *Null*; and a relational algebra with a three-valued logic to evaluate expressions involving the Null value.

Constraints: As databases become increasingly complex, the task of checking that conditions on the validity of particular states needs to be automated. In other words, the desire emerges for formally specifying constraints as to what collections of tuples can exist in database states. A special case of this is the known interpretation of nulls, where in some implementations one can constrain two nulls to be equal (if their value is found out). Other examples of constraints include commutativity conditions on inter-table relations and the budget example given earlier. How query languages can make inferences on data with nulls and constraints together has been explored in work stemming from Libkin's such as in [27] and only categorically recently by Diskin et al. in [9] and [10].

Tableau Semantics: As we saw that the set theoretic definition of relation prohibits tables from containing duplicate relations, there exists database formalisms in practice that seek to bypass this, for example "tableaus" in texts such as [32], among the need for databases with tables that have no primary key. Spivak in [31] defines categorical state spaces designed in part to remove the injectivity of primary keys and in turn defines more general constructions of tables out of other tables than is given by traditional relational algebra.

Fuzzy & Temporal Databases: The fuzzy relational model involves giving certain relational tuples an associated probability of "existing" between 0 and 1 such that the interactions of this probability are deeper than merely giving the relation a probability attribute. This is explored categorically by defining database states via indexed categories and allegories in [29] and [38] respectively. Temporal database theory involves endowing relations with a historical log about the truth of data. For example, in a table that stores tax-related information about a person, if at a later time retroactive information that reveals tax was avoided or overpaid is discovered, both facts need be recorded but certain facts are *currently* true. The allegorical description of the logical structure of databases, for which support for modal operators on tuples is given is explored in [39].

We conclude this chapter on relational database theory by discussing some concepts behind presenting the logical structure of a database, which is another common interest of Category Theoretic formulations of data models.

2.4 The Logical Structure of A Database; Schemas & ERA

Formally specifying the logical structure of database is another task with concerns hidden by the simplified set theoretic description of the relational model given thus far. An important abstraction is that of a *Schema*, which is a formal language specification of a database that includes: What Entities a database is comprised of, what Attributes those entities have and stored procedures of a query language that may be necessary for the construction of certain entities – just to name a few. As we generalise the relational model, a Schema may also specify what types those attributes have, what constraints database states need to satisfy etc.

Category Theory has been employed to appropriately encode schemas. Over the page in Chapter 3. we will discuss how certain Schemas can be expressed as Mixed Sketches, as first formulated by Johnson and Rosebrugh in [17]. In Section (6.2) we will discuss how certain Schemas can be expressed as a simplicial set introduced by Spivak in [31]. Other treatments not treated are expressing Schemas as a particular Allegory introduced in [38] and free-monoid objects in a particular topos in [29].

In many cases, the desire to formalise schemas for data models more general than the relational model suggests why more general categorical objects than categorical relations are used. Roughly speaking, the formal specification needs to have more expressive power than the set theoretic specification of a (collection of) relations. Another motivation is the desire for schemas to have a visual presentation and even illuminating graph theoretic or topological features. On one hand visualising schemas can serve to make it easier for users to understand the logical structure of a database and on the other hand comparing visual schemas affords more tools to assessing view update problems.

The most common method of visualising schemas is to form an ERA (Entity (inter-table) Relation Attribute) diagram of it, which is a graph with nodes and edges given by either:

Figure 2.a) Nodes as tables, annotated by its attribute types and Edges as inter-table relations annotated by their properties (E.g. “one-to-one”, “many-to-one”, “one-to-many”) – or:

Figure 2.b) Nodes as either tables or attribute types and Edges as either inter-table relations annotated by their properties or declarations that a table has a particular attribute.

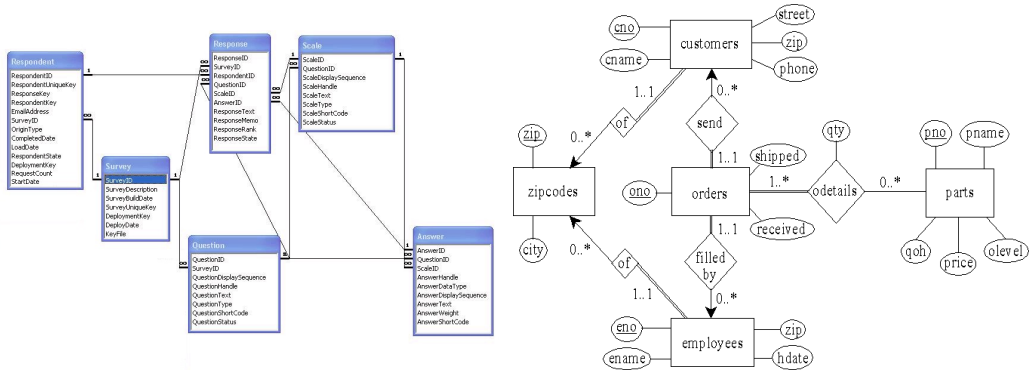


Figure 2.2: Two Kinds ERA Diagrams from [34] and [33] respectively.

Chapter. 3

The Sketch Data Model

The purpose of this chapter is to derive the Sketch Data Model in detail, along with some examples and general results about view-updating between such Categorical state spaces. These exercises are useful so we have a point of comparison as to what a categorical data model can and should do. This, especially because the categorical data model derived in Chapter 4 discards many features of database specifications one would expect from a categorical data model. Furthermore, this Chapter will serve to give us a good intuition as to how insert and delete propagability; and cartesian and opcartesian lifts can be least-change solve view-update problems. The Sketch Data Model also doesn't allow in-place edits and its non-monic updates are redundant in practice. Because in Chapter 4. we allow in-place edits and study (op)cartesian lifts, which don't require updates to be monic, we have another useful point of comparison studying the Sketch Data Model.

In terms of section by section structure: Section (3.1) provides us our basic definitions and shows how attribute domains, entities, relations and keys are formally specified using sketches. Section (3.2) takes a necessary digression into how models on sketches emerged in the world of Categorical Universal Algebra, providing us with the definition of the "theory of a sketch". This definition is crucial to Section (3.3), which shows us how we can specify standard relational algebraic queries and views constructed out of them. We pay special attention to monotonic queries. In section (3.4) we define cartesian and opcartesian lifts and begin to study least change solutions to view-update problems using these constructions. Constant Complement Updating is defined here. Then, finally in Chapter (3.5) we survey two approaches to how typed nulls and partial-map nulls (respectively) can be implemented in the Sketch Data Model.

3.1 Formally Specifying Database Schemas as Sketches

Definition 3.1.1. A (mixed) *sketch* $\mathbb{E} = (\mathcal{G}, L, C)$ is a triple consisting of a small category $\mathcal{G}$, a set L of cones in $\mathcal{G}$ and a set C of cocones in $\mathcal{G}$.

A *morphism of sketches* $\mathbb{E} = (\mathcal{G}, L, C)$ and $\mathbb{E}' = (\mathcal{G}', L', C')$ is given by a functor $h : \mathcal{G} \to \mathcal{G}'$; precomposition by h in turn specifies functions $L \to L'$ and $C \to C'$.

As a notational convenience, we denote the category specified as data of a sketch $\mathbb{E}$ as $C(\mathbb{E})$.

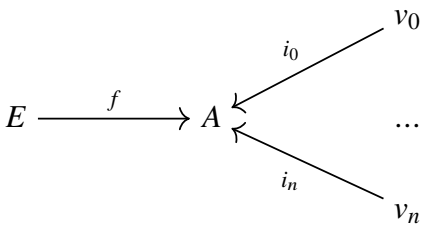
Definition 3.1.2. Given a sketch $\mathbb{E} = (\mathcal{G}, L, C)$ and a category $\mathcal{S}$: a *model of* $\mathbb{E}$ *in* $\mathcal{S}$ is given by a functor $M : \mathcal{G} \to \mathcal{S}$ such that every cone in L of $\mathcal{G}$ is mapped to a limiting cone in $\mathcal{S}$ and every cocone in C of $\mathcal{G}$ is mapped to a colimiting cocone in $\mathcal{S}$.

We write $Mod(\mathbb{E}, \mathcal{S})$ to denote the category with models of $\mathbb{E}$ in $\mathcal{S}$ as objects and natural transformations between them as morphisms.

Note that in computer science literature on the Sketch Data Model, mixed sketches are often defined with a directed multigraph and *set of commutative diagrams* in that graph in place of a category. Consequently, cones, cocones, models etc. are redefined in terms of the graph and set of commutative diagrams.

Example 3.1.3. To show how Attribute domains and the formation of tuples is given, we deriving a sketch $\mathbb{E} = (\mathcal{G}, L, C)$ that corresponds to the schema of a database consisting of a single entity E with a single attribute A, with attribute domain domain $\{0, 1, ..., n\}$.

We take $\mathcal{G}$ to be the following category:



(3.1)

We begin populating L and C so that models of $\mathbb{E}$ in **Set** map A to an appropriate attribute domain corresponding to $\{0, 1, ..., n\}$, further, so E is mapped to a set of instances that take attribute values via f.

We start by specifying v_0, v_1,...., v_n as vertices of cones over the empty diagram in L, that is the diagram given by a functor from the empty category into $\mathcal{G}$. Consequently, any model of $\mathbb{E}$ in **Set** must send v_0, v_1....,v_n to limiting cones over the empty diagram, that is, terminal objects. In **Set**, a terminal object is just any singleton.

We also specify the family $(i_0, i_1, ...i_n)$ as a cocone in C so that any model M of $\mathbb{E}$ into any category must map A to the colimiting cocone of $(Mi_0, Mi_1, ..., Mi_n)$, which is the coproduct of Mv_0, Mv_1, ... , Mv_n. In **Set** that is just a disjoint union of the n sets and because $Mv_0 =: \{s_0\}$, $Mv_1 =: \{s_1\}$, ... , $Mv_n =: \{s_n\}$ are singletons, $MA = \{a_0, a_1, ..., a_n\}$ such that $a_\ell = Mi_\ell(s_\ell)$ for $\ell = 0, 1, ..., n$. Moreover, ME is any set of instances of E so that $Mf : ME \to \{a_0, a_1, ..., a_n\}$ assigns each instance of the entity E an MA value – Note that ME is not the primary key of the table, it is merely the set of its instances.

Now even though MA is in general only isomorphic to the set $\{0, 1, ..., n\}$, the following argument shows why isomorphism suffices. If we take two any other model $N \in Mod(\mathbb{E}, \mathbf{Set})$ and consider any morphism $\alpha : M \Rightarrow N$, if $Nv_\ell = \{t_\ell\}$ for $\ell = 0, 1, ..., n$ and $NA = \{b_0, b_1, ..., b_n\}$, we note the commutativity of the naturality squares for $\ell = 0, 1, ..., n$:

$$\begin{array}{ccc} \{s_\ell\} = Mv_\ell & \xrightarrow{Mi_0} & \{a_0, a_1, ..., a_n\} = MA \\ \downarrow{\scriptstyle \alpha_{v_0}} & & \downarrow{\scriptstyle \alpha_A} \\ \{t_\ell\} = Nv_\ell & \xrightarrow[Ni_0]{} & \{b_0, b_1, ..., b_n\} = NA \end{array}$$

Requires that $\alpha_A(a_\ell) = b_\ell$. Hence for any model $M \in Mod(\mathbb{E}, \mathbf{Set})$, the attribute domain $\{0, 1, ..., n\}$ indexes MA and morphisms in $Mod(\mathbb{E}, \mathbf{Set})$ preserve this indexing. So, without loss of generality, associating attribute values ℓ to i_ℓ, for any model of $\mathbb{E}$ in **Set**, it is well founded saying that the mapping $Mf : ME \to MA$ associates each instance $e \in ME$ an attribute value m precisely when $Mf(e)$ is the image of Mi_m. Note that this association holds for attribute domains that are arbitrary sets, where the attribute-cocones are indexed by a small, discrete category.

To conclude this example, we investigate how α_E behaves. Indeed, because we deduced the mappings of α_A, the commutativity of the naturality square:

$$\begin{array}{ccc} ME & \xrightarrow{Mf} & \{a_0, a_1, ..., a_n\} = MA \\ \downarrow{\scriptstyle \alpha_E} & & \downarrow{\scriptstyle \alpha_A} \\ NE & \xrightarrow[Nf]{} & \{b_0, b_1, ..., b_n\} = NA \end{array}$$

Implies that $\forall x \in ME$, if $Mf(x) = a_m$ for some $m \in \{0, 1, ..., n\}$ then $\alpha_A \circ Mf(x) = b_m$ so that commutativity requiring that $Nf \circ \alpha_E(x) = b_m$ means that:

α_E can only map instances of ME to instances of NE with the same A-attribute value.

Hence $Mod(\mathbb{E}, \mathbf{Set})$ is a state space category with the desired schema, where updates can't make in-place edits.

Note that α_E can emulate deletions by mapping two instances $e, e' \in ME$ to the same instance in NE; although encoding deletions in this way may yield that a particular deletion can be performed in many different ways or none at all. Hence if N is M with some deletions, we should represent that update by a monomorphism $N \to M$, which is a natural transformation where every component is an injection. ■

Given a sketch $\mathbb{E} = (\mathcal{G}, L, C)$, we can in general associate a Schema to $\mathbb{E}$ in the following way:

- We call an object $A \in \mathcal{G}$ an *Attribute in* $\mathbb{E}$ if there is a cocone $(i_x : V_x \to A)_{x \in \mathbf{A}} \in C$ with V_x as the vertex of a cone over the empty diagram in L and every other such cocone is contained in $(i_x)_{x \in \mathbf{A}}$. The associated schema has an Attribute Domain isomorphic to $(i_x)_{x \in \mathbf{A}}$.
- Every other object $E \in \mathcal{G}$ that isn't a cone over the empty diagram is called *Entity in* $\mathbb{E}$. The associated schema has an entity E with an attribute with header h and attribute domain $\mathbf{A}$ for each morphism $h \in Hom_{\mathcal{G}}(E, A)$, where A is an Attribute in $\mathbb{E}$.
- For any two entities $E, F \in \mathcal{G}$, there is a functional, inter-table relation from E to F for each morphism in $Hom_{\mathcal{G}}(E, F)$.

We note that models into **Set** of sketches subsume relational databases that have arbitrary inter-table relations because for any relation $R \subset S \times T$ for sets $S \times T$, we can take a span $S \xleftarrow{f} R' \xrightarrow{g} T$, where $R' \cong R$ with $\forall (s, t) \in R : \ \exists! r' \in R'$ such that $f(r') = s$ and $g(r') = t$. Conversely, given the span, there is a corresponding relation $R = \{(f(r'), g(r')) : r' \in R\} \subset S \times T$.

Furthermore, any additional limits provided in L and C specify other constraints on how tables are composed in relation to each other; in Section (3.3) we describe we describe some basic correspondences between relational algebra and specified limits and colimits on sketches.

One can also reverse the above steps associating a sketch to a relational schema, there are of course many ways to do this. A simplification to keep in mind is that only a single vertex of a cone over the empty diagram is needed as we only use the indexing of morphisms of the attribute-cocone to identify attribute domains and not the source of those morphisms.

We conclude the introduction to this chapter by deriving a way to encode primary keys into sketches, corresponding to a database schema. To begin we need the following result from Category Theory:

Proposition 3.1.4. Let C be a category. A morphism $f : X \to Y$ in C is monic if only if the following commutative square is a pullback:

$$\begin{array}{ccc} X & \xrightarrow{id_X} & X \\ {\scriptstyle id_X}\downarrow & & \downarrow{\scriptstyle f} \\ X & \xrightarrow[f]{} & Y \end{array} \tag{3.2}$$

Proof. ($\Longrightarrow$) Given any cone of $X \underset{f}{\overset{f}{\rightrightarrows}} Y$, which includes a pair of morphisms $g, h : Z \to X$ in C such that $f \circ g = f \circ h$. If f is monic, $f \circ g = f \circ h$ implies that $g = h$ satisfies the universal mapping property of the pullback; and uniquely so because any other morphism $k : Z \to X$ that does so must satisfy $g = k \circ id_X = h$. Hence (3.6) is a pullback.

($\Longleftarrow$) Conversely, if (3.6) is a pullback then there exists a unique morphism $k : Z \to X$ such that $g = id_X \circ k = h$. Thus by the arbitrariness of g and h, the fact that $f \circ g = f \circ h \implies g = h$ implies by definition that f is a monomorphism.

□

We recall that a primary key of an entity is an attribute such that the projection from the set of entity instances to that attribute is injective. So we find firstly for any sketch $\mathbb{E}$ and morphism $f : X \to Y$ in $C(\mathbb{E})$, we can ensure that all models $M \in Mod(\mathbb{E}, \mathbf{Set})$ are such that Mf is injective if we specify the cone (id_X, id_X) as as cone over the diagram $X \underset{f}{\overset{f}{\rightrightarrows}} Y$ in $C(\mathbb{E})$.

The following definition coincides with primary keys in the relational data model:

Definition 3.1.5. For a sketch $\mathbb{E} = (\mathcal{G}, L, C)$, we say an entity $E \in \mathcal{G}$ is *keyed* with respect to an attribute A_E, called its *primary key*, if there is a morphism $k_E : E \to A_E$ in $\mathcal{G}$ and a cone $(id_E, id_E) \in L$ over the diagram $E \underset{k_E}{\overset{k_E}{\rightrightarrows}} A_E$ in $\mathcal{G}$. Moreover, if every entity in $\mathcal{G}$ is keyed we say that $\mathbb{E}$ *is keyed*.

Thus because it was shown that morphisms in $Mod(\mathbb{E}, \mathbf{Set})$ can't change attribute values, morphisms must fix primary key values as well.

Beyond showing how certain standard queries on a database schema given by a sketch correspond to specifying particular cones and cocones as data of that sketch, we also seek to show that a View of a database can be defined by a particular kind of sketch morphisms from a View-sketch. However, View schemas can of course be constructed out of standard queries, so to appropriately define sketch morphisms that reflect this, we need to discuss some seemingly digressive aspects of the study of sketches in the following section.

3.2 Sketches in Logic; Schemas as Type Theories

This section will condense some concepts that have historically motivated the development of sketches in studying universal algebra and logic. However, the brevity of this section may obscure these concepts both in pedagogy and content; the following texts of Barrs and Wells are repeatedly cited: [2] as the more detailed source and [3] in more elementary Category Theoretic terms.

To begin, the ethos of this section can be summarised as John Jane Bell once wrote: "Roughly speaking, a category may be thought of [as] a type theory shorn of its syntax" [5, p.15]. A motivating example of this is the Lawvere Theory of a group, which involves understanding a generic group G as a type theory with one type: group elements of G.[1] The *type theory of a group* can be given by a number of equational conditions on three function symbols: multiplication, denoted $m : G \times G \to G$, a function symbol identifying each element's inverse $i : G \to G$ and the identity picked out by a function symbol $e : 1 \to G$ from a unit type 1.

The equational conditions must precisely encode the axioms of a group, namely, instances of G satisfy associativity with respect to m, that i picks out two-sided inverses and e picks out a two-sided identity. Instead of deriving a formal language to list an encoding of these axioms, we can instead encode them as commutativity conditions on a category $\mathcal{G}$ with binary products and a terminal object $1 \in \mathcal{G}$. For example, for an object $G \in \mathcal{G}$ and morphisms $m : G \times G \to G, i : G \to G$ and $e : 1 \to G$, we require the diagrams of Figure. 3.1. commute. Note that the type theory of a group requires a theory for an *equational logic* that denotes type theoretically how the "equals sign" and variable substitution operate; these can be internalised along with the internalisation of a group using only finite products, see [28, Chapter 1] for more details.

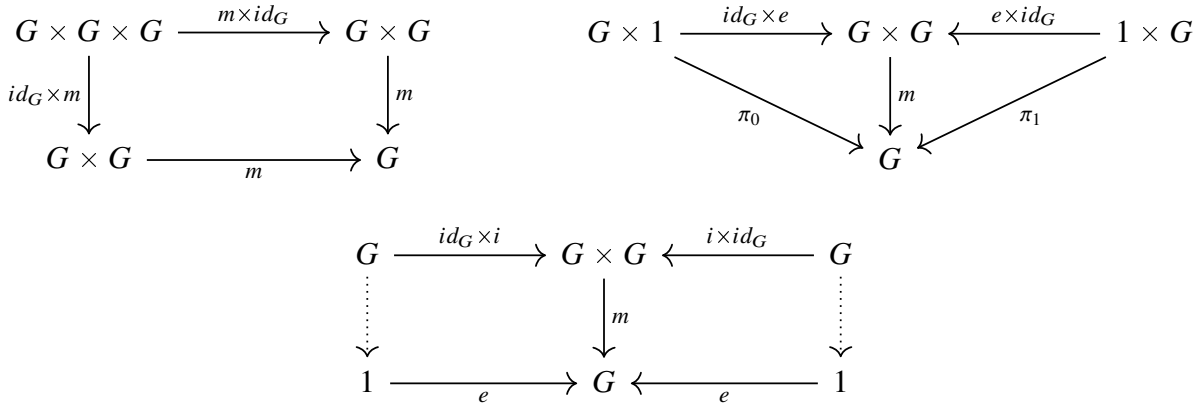


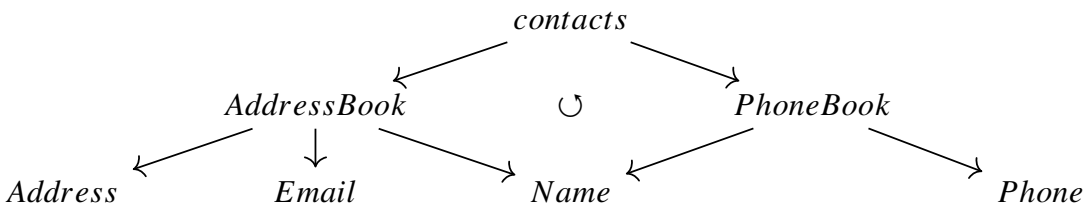
Figure 3.1: Diagrams encoding associativity, existence of a two sided identity, two sided inverses; where π_0, π_1 denote the projections of the first and second factors of their respective products.

The fact we have precisely encoded the axioms of a group as we traditionally know them is clearer when we consider any product preserving functor $M : \mathcal{G} \to \mathbf{Set}$. Indeed MG is a set with a group structure with multiplication given by $Mm : MG \times MG \to MG$, identity $Me(*)$ for $M1 = \{*\}$; for any group element $g \in MG$, its inverse is given by $Mi(g)$. More generally, for a category C with binary products and a terminal object, a product preserving functor $F : \mathcal{G} \to C$ maps G to what is known as a group object.

[1] One type as opposed to say, an R-Module of a ring R over an abelian group M, which has two types: elements of R and elements of M.

Better yet, we can specify our type theory of a group by a sketch $\mathbb{G}$ with $C(\mathbb{G})$ consisting of 1, G and its finite products and can recover groups and group homomorphisms by models and natural transformations $Mod(\mathbb{G}, \mathbf{Set})$. Note that Sketches as we have defined them carry cocone specifications that are unused; sketches where only finite products are specified as cone-data are called *Finite Product Sketches*. An example of of an algebra that requires the expressivity of mixed sketches is the task of specifying the *theory of a field*, see: [3, p.255-256].

This perspective affords a deeper intuition behind the Sketch Data Model, particularly for a schema abstractly named X, given by the type theory of a sketch $\mathbb{X}$; a database state as a model M of $\mathbb{X}$ in say, **Set** is an *X-object in* ***Set***, for instance. For example, the *contacts* relation of Example (2.2.1) has as a sketch $\mathbb{C}$ with underlying category (suppressing the attribute cocones):



(3.3)

Database states with the "ContactsList" schema given by $\mathbb{C}$ are precisely *ContactsLists objects in* ***Set***. Algebraically, there are $Address$, $Email$, $Name$ and $Phone$ types with finite amounts of elements. There are also operations that associate $AddressBook$ and $PhoneBook$ types with an $Address$, $Email$, etc. and likewise for the $contacts$ type. The *type theory of a ContactsList* for has only the equational axiom that the centre diamond commutes. The universal algebraic interpretation of schemas becomes particularly useful when considering an important construction associated to sketches known as the *theory of a sketch*. To motivate what the *theory of a sketch* is, it helps to first motivate it in the case of the Finite Product Theory of a Group.

A familiar question in group theory that arises is *how does one know that given two presentations of group that they define the same group*? Abstractly to answer this, all one needs to do is: for each presentation, generate all the group elements that exist with respect to the generators and commutativity conditions of the presentation; then if the two presentations generate the same elements – they present the same group. The same question can be asked of sketches, for example, *given two finite product theories: how do we know they encode the "same" type theory*? Analogously the answer to this question is to *formally add all finite products* into the sketches and specification of cones and find the resulting sketches are the same – The resulting sketch is called the *theory* of the two sketches. A precise statement of what this sketch is for Finite Product Theories is as follows (see: [3, p.237]):

Theorem 3.2.1. *Given any finite product sketch $\mathbb{E}$, there is a sketch $Th(\mathbb{E})$ with finite products and a model $M_0 \in Mod(\mathbb{E}, C(Th(\mathbb{E})))$ such that for every model $M \in Mod(\mathbb{E}, C)$ into a category with finite products, there is a functor $F : C(Th(\mathbb{E})) \to C$ that preserves finite products for which:*

i) $F \circ M_0 = M$, *and*

ii) if $F' : C(Th(\mathbb{E})) \to C$ is another functor that preserves finite products for which $F' \circ M_0 = M$, then F and F' are naturally isomorphic.

An important consequence of this is given in [3, p.238] is that there is an equivalence of categories $Mod(\mathbb{E}, C) \simeq Mod(Th(\mathbb{E}), C)$, a result more intuitive in the context of the Sketch Data Model.

The formulation of the Theory of a Mixed Sketch is more complex. The details are traced back to [2, Section 8.2] by Johnson in [18, p.10], who describes the construction as "*roughly speaking, the category generated from the sketch* $\mathbb{E}$ *subject to the commutative diagrams in* $\mathcal{G}$ *by closing it under the operations of finite limit and finite coproduct subject to the (co)cones in L and C*." We proceed to apply these concepts to define views in the Sketch Data Model.

3.3 The Theory of a Sketch; Queries & Views

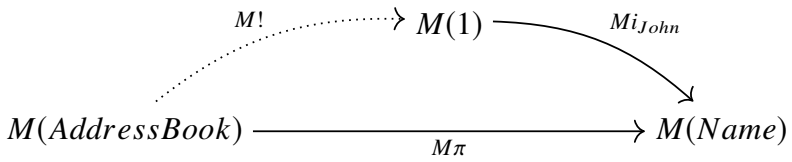
Because views as defined in the relational model are databases derived from some source database using relational algebraic operations; to define views in the Sketch Data Model, we need discuss how to use sketches to encode relational algebraic operations typical to database query languages.

The basic relational-algebraic operators of any database query language are known as *select, project, (natural) join* and *(disjoint) union*; which can be specified in a sketch by the underlying cones and cocones defining equalisers, product-projections, pullbacks and coproducts, respectively.

Example 3.3.1. Considering the schema produced for the database in Example (2.2.1), if we wanted to represent an SQL query of the form:

$$\text{select * from } AddressBook \text{ where } Name = \text{“John”}$$

That lists every instance of $AddressBook$ who's name attribute is "John"; given any state $M \in Mod(\mathbb{E}, \mathbf{Set})$ of the database, if one inspects the diagram:



(3.4)

Where i_{John} is the injection into the $Name$ cocone that defines the index of the name-string "John" in the attribute domain. We find our query requires us to find the subset of elements of $M(AddressBook)$ which map to "John" in $Name$; this is given by the limit (equaliser) of:

$$M(AddressBook) \underset{M\pi}{\overset{M(i_{John} \circ\, !)}{\rightrightarrows}} M(Name) \tag{3.5}$$

Indeed there is a morphism $q : John \to AddressBook$, specified as a cone over (3.5) in the sketch $Th(\mathbb{E})$, for which models into **Set** must map $John$ to the appropriate query output.
■

Hence given a sketch $\mathbb{E}$, we can understand the theory of $\mathbb{E}$: $Th(\mathbb{E})$ as the sketch consisting of all standard queries that can be made on the schema $\mathbb{E}$. This motivates the following definition of a view:

Definition 3.3.2. Let $\mathbb{E}$ be a sketch. A *view* of $\mathbb{E}$ is given by a sketch $\mathbb{V}$ and a sketch morphism $V : \mathbb{V} \to Th(\mathbb{E})$, recall from Definition (3.1.1) this is just given by a functor $V : C(\mathbb{V}) \to C(Th(\mathbb{E}))$.

Then for any category $\mathcal{S}$, there is a view-get $V^* : Mod(\mathbb{E}, \mathcal{S}) \to Mon(\mathbb{V}, \mathcal{S})$ as follows: We recall there exists a model $M_0 \in Mod(\mathbb{E}, C(Th(\mathbb{E})))$ such that for any model $M \in Mod(\mathbb{E}, \mathcal{S})$, there exists a unique (up to natural isomorphism) limit and coproduct preserving functor $F_M : C(Th(\mathbb{E})) \to \mathcal{S}$ such that $F_M \circ M_0 = M$; so that the composite $F_M \circ V : C(\mathbb{V}) \to \mathcal{S}$ canonically defines V^*.

A categorical analogue of views defined by monotonic queries can given as follows, noting that we use the subset symbol $\subset$ for models in **Set** to denote subfunctors:

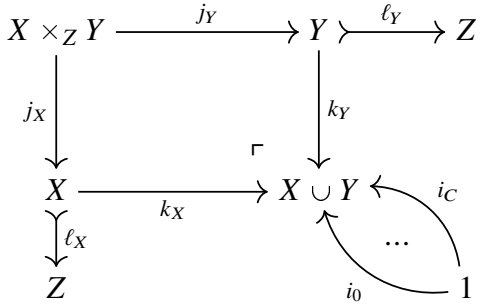
Definition 3.3.3. Let $V : \mathbb{V} \to Th(\mathbb{E})$ be a view of $\mathbb{E}$. We say that V is constructed from a *monotonic query* if:

$$\forall M, N \in Mod(\mathbb{E}, \mathbf{Set}) : \qquad M \subset N \implies V^*M \subset V^*N$$

One can just as well replace the above with "V^* preserves monics"; the subject of subfunctors just aligns the condition with the classical definition of monotonic queries in relational database theory.

Example 3.3.4. (Non Monotonic Queries)

Let $C > 2$ be a natural number constant and $\mathbb{E}$ be the sketch with underlying category:



(3.6)

That is, we specify as cones and cocones of $\mathbb{E}$ as:

1. (id_X, id_X) as a cone over (ℓ_X, ℓ_X) and (id_Y, id_Y) as a cone over (ℓ_Y, ℓ_Y).
2. (j_X, j_Y) as the cone over the inclusions (ℓ_X, ℓ_Y) into Z.
3. (k_X, k_Y) as a cocone over (j_X, j_Y).
4. $(i_0, ..., i_C)$ as a cocone over 1.

Then a model $M \in Mod(\mathbb{E}, \mathbf{Set})$ must satisfy $M(X \times_Z Y) \cong MX \cap MY$ by the cone of 2 becoming a pullback over the monics (see: Proposition (3.1.4) applied to 1). Furthermore, 4 forces $|M(X \cup Y)| = C$ and 4 additionally forced $M(X \cup Y) = MX \cup MY$.

We then define the view $V : \mathbb{1} \to Th(\mathbb{E})$, where $\mathbb{1}$ is the sketch with underlying category comprising of a single object X' and only the identity morphism on X'. We take V to map that object to X so that $V^*M(X') = MX$.

Now showing that V is not monotonic is a simple task: consider any two models $M, N \in Mod(\mathbb{E}, \mathbf{Set})$ such that $V^*M \subsetneq V^*N$. Because $|MX| \lneq |NX| \leqslant C$ in order for $|M(X \cup Y)| = |N(X \cup Y)| = C$ we must have $|NY| = C - |NX| \lneq C - |MX| = |MY|$ and hence $M \not\subset N$.

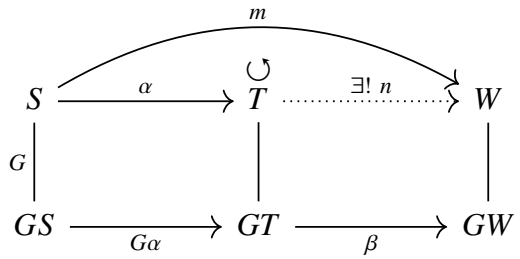
Not only is V not a view definition but the above proof just showed that any propagation $f : M \to L$ of the view-insert defining $V^*M \subset V^*N$ required deletions to be made to MY so that f can't be a monomorphism. Thus V^* can't be insert-propagable.

■

In the next section, we proceed to define cartesian and opcartesian lifts, provide an example of a view-definition that cannot admit all opcartesian lifts and derive Johnson and Rosebrugh's approach of constant complement updating.

3.4 A Categorical Abstraction of View Updating

Definition 3.4.1. Let $\mathcal{S}$ and $\mathcal{V}$ be categories and $G : \mathcal{S} \to \mathcal{V}$ be a functor. We say a morphism $\alpha : S \to T$ is *opcartesian* (with respect to G) if for any morphism $m : S \to W$ such that $Gm = \beta \circ G\alpha$ for some $\beta : GT \to GW$ in $\mathcal{V}$, there exists a unique morphism $n : T \to W$ such that $Gn = \beta$ and $n \circ \alpha = m$, diagramatically:



(3.7)

Moreover, if $\alpha : GS \to V$ is a morphism in $\mathcal{V}$ such that there exists a morphism $\alpha^{\uparrow} : S \to V^{\uparrow}$ such that $G\alpha^{\uparrow} = \alpha$ and $\alpha^{\uparrow}$ is opcartesian, we say that $\alpha^{\uparrow}$ is an *opcartesian lift of* α.

Reversing the arrows of the above definition gives us the definition of a cartesian morphism and lift; that is, cartesian lifts with respect to G are just opcartesian lifts with respect to G^{op}.

Definition 3.4.2. A functor $G : \mathcal{S} \to \mathcal{V}$ is a *(Grothendieck) Fibration (respectively, Opfibration)* if for every morphism $\alpha : V \to GS$ in $\mathcal{V}$, there is an cartesian (respectively, opcartesian) lift of α.

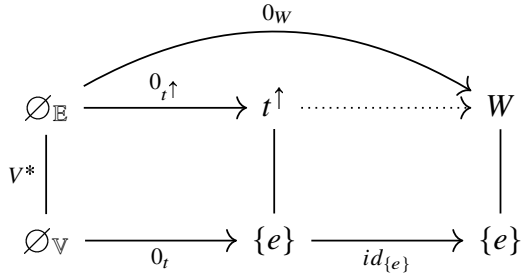
We call a choice of cartesian (respectively, opcartesian) lift for every morphism $\alpha : GS \to V$ (respectively, $\delta : U \to GS$) in $\mathcal{V}$ a *cleavage of* G. Assuming the axiom of choice, a functor $G : \mathcal{S} \to \mathcal{V}$ is an (Op)Fibration if and only if it has a cleavage.

Example 3.4.3. (Forgetting Attributes)

Let $\mathbb{E}$ be the sketch with $C(\mathbb{E})$ given by $E \xrightarrow{f} A \underset{\cdots}{\overset{\cdots}{\rightleftarrows}} 1$, where A is an attribute and let $\mathbb{V}$ be a sketch with $C(\mathbb{V})$ given by just E. We take the obvious $V : \mathbb{V} \to Q(\mathbb{E})$.

We can show that V^* is not an opfibration as follows. Let $\varnothing_{\mathbb{E}} \in Mod(\mathbb{E}, \mathbf{Set})$ be such that $\varnothing_{\mathbb{E}}(E) = \varnothing$ and $\varnothing_{\mathbb{V}} \in Mod(\mathbb{V}, \mathbf{Set})$ such that $\varnothing_{\mathbb{V}}(E) = \varnothing$; we find that $V^*(\varnothing_{\mathbb{E}}) = \varnothing_{\mathbb{V}}$. We then let $t \in Mod(\mathbb{V}, \mathbf{Set})$ such that $t(E) = \{e\}$ and try to find an opcartesian lift of $0_t : V^*(\varnothing_{\mathbb{E}}) \to t$, which is the natural transformation consisting of an empty function to $\{e\}$.

A state $t^{\uparrow} \in Mod(\mathbb{E}, \mathbf{Set})$ such that $V^*(t^{\uparrow}) = t$ must have $t^{\uparrow}(E) = \{e\}$ and some attribute value $t^{\uparrow} f(e) \in t(A)$. Likewise since $V^*(M) = \varnothing_{\mathbb{V}} \iff M = \varnothing_{\mathbb{E}}$ an opcartesian lift of 0_t must be of the form $0_{t^{\uparrow}} : \varnothing_{\mathbb{E}} \to t^{\uparrow}$ where $t^{\uparrow}(e) = a$ for some $a \in t(A)$. However, being an opcartesian lifts requires that for any $W \in Mod(\mathbb{E})$ with $W(E) = \{e\}$ and $Wf(e) \neq a$ the dotted morphism exists:



But we know if $Wf(e) \neq a$ that there can't be a natural transformation $t^{\uparrow} \Rightarrow W$ (see: Example (3.1.3)). Hence there is no opcartesian lift of 0_t and thus V^* is not an opfibration.
■

We proceed to show a sufficient condition for both insert and delete-propagability in the Sketch Data Model formulated by Johnson and Rosebrugh [19] called *constant complement updating*, which refers to a classical class of solutions to the view update problem originating with Brancilhon and Spyratos in [1].

Definition 3.4.4. Let $\mathbb{E}$ be a sketch with a views $V : \mathbb{V} \to Th(\mathbb{E})$ and $C : \mathbb{C} \to Th(\mathbb{E})$. We say that C is a *complement of* V if the functor given by the universal mapping property of products: $\langle V^*, C^* \rangle : Mod(\mathbb{E}, \mathbf{Set}) \to Mod(\mathbb{V}, \mathbf{Set}) \times Mod(\mathbb{C}, \mathbf{Set})$ is full, faithful and injective on objects.

The situation of views with a complement is that where a source state is not only uniquely determined by a certain view state and *complemented* view state but that there is a bijection between updates of source states and pairs of updates between these view states. Having a view-complement isn't enough for universal view-updatability, the following is however.

Definition 3.4.5. Let $V : \mathbb{V} \to Th(\mathbb{E})$ and $C : \mathbb{C} \to Th(\mathbb{E})$ be views such that C is a complement of V. We say that a morphism of the form $\alpha : V \to GS$ in $Mod(\mathbb{V}, \mathbf{Set})$ for any $S \in Mod(\mathbb{E}, \mathbf{Set})$ has a *C-constant update*, there is a morphism $\hat{\alpha}$ in $Mod(\mathbb{E}, \mathbf{Set})$ such that $V^*\hat{\alpha} = \alpha$ and $C^*\hat{\alpha}$ is an isomorphism in $Mod(\mathbb{C}, \mathbf{Set})$. If all morphisms of the form of α have a C-constant update then we say that V has C-constant updates.

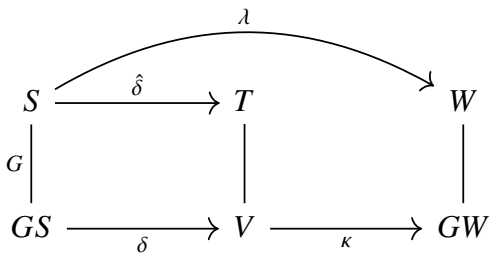
Dually, we say a morphism $\delta : GS \to V$ in $Mod(\mathbb{V}, \mathbf{Set})$ for any $S \in Mod(\mathbb{E}, \mathbf{Set})$ has a *C-opconstant update*, there is a morphism $\hat{\delta}$ in $Mod(\mathbb{E}, \mathbf{Set})$ such that $V^*\hat{\delta} = \delta$ and $C^*\hat{\delta}$ is an isomorphism in $Mod(\mathbb{C}, \mathbf{Set})$. And If all morphisms of the form of δ have a C-opconstant update then we say that V has C-opconstant updates.

We consequently have the following due to Johnson and Rosebrugh in [19, p,248]:

Theorem 3.4.6. *Let $V : \mathbb{V} \to Th(\mathbb{E})$ and $C : \mathbb{C} \to Th(\mathbb{E})$ be views such that C is a complement of V and $\alpha : V \rightarrowtail GS$ is a monic in $Mod(\mathbb{V}, \mathbf{Set})$ with a C-constant update, then $\hat{\alpha}$ is a cartesian lift of α. Dually, if $\delta : GS \rightarrowtail V$ is a monic in $Mod(\mathbb{V}, \mathbf{Set})$ with a C-opconstant update then $\hat{\delta}$ is an opcartesian lift of δ.*

Proof. The construction of a monic, cartesian lfit of $\alpha : V \rightarrowtail GS$ is given explicitly in [19, Theorem 1] and more briefly for $\delta : GS \to V$, so the latter case will be proven in detail.

To show $\hat{\delta}$ satisfies the universal property of opcartesian lifts, If we then consider any diagram:



We find that $\langle V^*, C^* \rangle$ is full implies there is a morphism $n : T \to W$ in $Mod(\mathbb{E}, \mathbf{Set})$ such that $\langle V^*, C^* \rangle(n) = (\kappa, C^*(\lambda) \circ (C^*\hat{\delta})^{-1})$ and I firstly claim this satisfies the commutativity conditions of

the universal mapping property of opcartesian lifts – indeed:

$$\begin{aligned}\langle V^*, C^*\rangle(n \circ \hat{\delta}) = (V^*(n \circ \hat{\delta}), C^*(n \circ \hat{\delta})) &= (\kappa \circ \delta, (C^*(\lambda) \circ (C^*\hat{\delta})^{-1}) \circ C^*(\hat{\delta})) \\ &= (V^*(\lambda), C^*(\lambda))\end{aligned}$$

And because $\langle V^*, C^*\rangle$ is faithful it must be the case that $n \circ \hat{\delta} = \lambda$ and by definition $V^*n = \kappa$. Thus it remains to show that n constructed as such is unique. We prove this by assuming $\exists m : T \to W$ such that $V^*m = \kappa$ and $m \circ \hat{\delta} = \lambda$; and show this implies $m = n$.

Indeed, $m \circ \hat{\delta} = \lambda$ implies that $C^*m = C^*m \circ C^*\hat{\delta} \circ (C^*\hat{\delta})^{-1} = C^*\lambda \circ (C^*\hat{\delta})^{-1} = C^*n$. Then because we additionally know $V^*m = \kappa = V^*n$, by the faithfulness of $\langle V^*, C^*\rangle$, we can conclude that $m = n$.

□

3.5 Two Categorical Approaches to Representing Nulls

In this section we will briefly compare two different approaches to defining nulls in the Sketch Data Model; the comparison of the approaches originates in [25].

Example 3.5.1. Recalling in Example (2.2.1) we motivated the need for there to be a special Null *Address*. Recovering the sketch $\mathbb{C}$ of the *contacts* relation from section 3.2, we can form the sketch $\mathbb{C}_1$, with the objects and morphisms of $\mathbb{C}$ with additional to $\pi : AddressBook \to Address$:

$$\begin{array}{ccccc} & & AddressBook & & \\ & & \downarrow{\scriptstyle \pi_+} & & \\ Address & \xrightarrow{i_{Address}} & Address_+ & \xleftarrow{null_{Address}} & 1 \end{array} \tag{3.8}$$

And $(i_{Address}, null_{Address})$ specified as a cocone over the diagram consisting of just $Address$ and 1, so that models $M \in Mod(\mathbb{C}_1, \mathbf{Set})$ have $M(Address_+) = M(Address) \sqcup M(1)$ and if $M(1) = \{\bot\}$: we understand an instance $a \in M(AddressBook)$ as having a null address if $M\pi_+(e) = \bot$.

Another approach can be given by taking for a sketch $\mathbb{C}_2$ with instead of π we have the span:

$$AddressBook \xleftarrow{i_{Address}} Address_{def} \xrightarrow{m_{Address}} Address$$

Where $(id_{Address_{def}}, id_{Address_{def}})$ is specified as a cone over $Address_{def} \underset{i_{Address}}{\overset{i_{Address}}{\rightrightarrows}} M(Name)$ so that models $N \in Mod(\mathbb{C}_2, \mathbf{Set})$ map $i_{Address}$ to an injection, where we say an instance $a \in N(AddressBook)$ has a null addressbook, precisely when $N(m_{Address})^{-1}(a) = \varnothing$. In other words, the span maps to a partial function $M(AddressBook) \nrightarrow M(Address)$, where $M(Address_{def})$ is its domain of definition. For an account of how the standard relational algebraic operations can be defined to support nulls of this second kind, see [14].

We can infer that $Mod(\mathbb{C}_1, \mathbf{Set}) \not\cong Mod(\mathbb{C}_2, \mathbf{Set})$ because in the latter the domain of definition for $AddressBook_{def}$ can have insertions made into it (until $AddressBook_{def} \cong AddressBook$), that is, we can turn nulls into defined values; however, in the first approach, turning nulls into a defined value involves changing an attribute value, which is not allowed.

■

3.6 Summary

As a categorical data model, the Sketch Data Model gives a very robust specification of schemas, view-definitions and the view-update problem for relational databases (with nulls if desired). This is given by encapsulating standard relational queries in constructions from categorical universal algebra (see: Section (3.3). A caveat to this robustness is that we must work with categorical state spaces of the form $Mod(\mathbb{E}, \mathbf{Set})$ and are restricted to updates that fix attribute values (see Example (3.1.3)). That is, updates that don't allow in-place edits. As a consequence, updates that aren't monomorphisms are redundant. All non-monic updates additionally do is merge some instances (with no differing attributes) by mapping them to the same instance in the target; this is a crude way to delete instances as it isn't always possible nor uniquely defined.

A typical reason for disallowing in-place edits is as follows: we note least change propagations of view-updates, whether insert-propagability (resp. delete propagability) or (op)cartesian lifting are defined up to isomorphism. So, if in-place edits are allowed, we are typically forced to make in-place edits isomorphisms, which implies that composing a least change propagation with an in-place edit may also be a least change propagation. Hence least change view-update propagations can sporadically in-place edit source data.

All being said, the next two chapters will explore examples where least change propagations can't be made without in-place edits being allowed, we also don't restrict to only using monics. However, the conditions of the main results turn out to be much more tractable when re-restricting to the use of simulated edits and heavily relying on updates being monics! Even so, the generalisations of these next chapters have other uses:

For the Sketch Data Model, we began with a categorical definition of least change propagations for view-inserts and view-deletes in the form of insert-propagability, delete-propagability and hence propagations for view simulated edits. There are hosts of conditions on view-definitions with respect to whether the corresponding view-gets are insert or delete propagable. Many are derived by Johnson and Rosebrugh in [18, Section 6], along with constant complement updating in [19].

However, it was shown in Example (3.3.4) that insert and delete-propagability may be untenable for view-definitions constructed out of queries that aren't monotonic. Hence, studying the more general conditions for cartesian and opcartesian lifts expands the scope for which we treat the view-update problem.

So, our more general results about cartesian and opcartesian lifts may depend on the heavy use of monic updates and monotonic views in applications to the view-update problem. However, keep in mind in Section (6.1) we will show that the generality we work in means our main results can be encapsulated in the language of asymmetric delta lenses, used in the study of bidirectional transformations. That is, lenses are applied to problems in Model Driven Engineering other than database design that have computational requirements different to those of relational database theory.

Chapter. 4

View Updating for View-Gets with Left Adjoints

The main purpose of this chapter is to prove Theorem (4.4.2) and Theorem (4.5.5), but also to motivate the constituent parts of these results in reference to issues regarding the view-update problem. Theorem (4.4.2) and Theorem (4.5.5) specify conditions on functors with a certain kind of left adjoint to have opcartesian and cartesian lifts, respectively. The statements and proofs of these results use only elementary Category Theory; at most the definitions of hom-set adjunctions, (op)cartesian lifts, pushouts and products. These results of ordinary Category Theory can be understood in isolation from our studies of categorical data models and the view-update problem. However, we motivate these results by studying examples of view-update problems expressed using a new, rather rudimentary categorical data model.

Before explaining our focus on motivation, recall, we will be moving to a setting where categorical data models have updates allowing in-place edits and that aren't necessarily monic; and part of this thesis' motivation was to reflect on the consequences of generalising in those manners to solving view-update problems.

Indeed, our focus on motivation is important because we need to keep track of how our theoretical progress relates to the quality of opcartesian and cartesian lifts as least change solutions to view-update problems. Particularly, so we can keep track of where interventions need to be made to improve the status of opcartesian and cartesian lifts as solutions to view-update problems. Another use for the new categorical data model we define is to produce results that show how the conditions of Theorem (4.4.2) and Theorem (4.5.5) can be checked explicitly when the categories involved are these novel categorical state spaces. Henceforth, our main results distinguish classes of opcartesian and cartesian lifts that can be directly computed given the appropriate left adjoint.

A more specific account of this chapter is as follows: in Section (4.1), we begin by showing a correspondence between left adjoints of initial object preserving functors and a certain collection of opcartesian lifts with respect to those functors. Section (4.2) motivates and defines our new categorical data model. Then, Section (4.3) derives an example of a view-update problem using this new categorical data model. This example is used repeatedly until the end of Chapter 5. Finally, sections (4.4) amd (4.5) motivate and prove Theorem (4.4.2) and Theorem (4.5.5), respectively. In those last two sections, accompanying results are given showing how their respective main theorems can be checked more explicitly, when the involved categories are state spaces expressed using our new categorical data model.

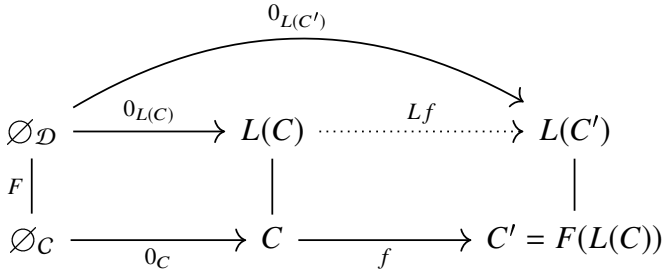
The following are left untreated until Chapter 5: Explorations on refining unsatisfactory view-update policies with a more general conception of nulls – inspired by Diskin's treatment of uncertain information in [9]; and the early stages of exploring the overlap between Theorem (4.5.5) and an analogue of Constant Complement updating – as defined in [19].

4.1 An Introductory Case: Initial Object Preserving View-Gets

Where the main result of this section is Proposition (4.1.1) stated below, we begin by briefly motivating the identification of initial objects in categorical state spaces – to contextualise this result. Indeed, if a categorical state space has a state where no entities have any instances then that state is typically an initial object. A view-get should then map this empty state in the source to an empty state in the view because there is no data to be derived into a view state. Hence, the functor should be initial object preserving. However, if our state space is constrained in such a way that there are no valid states without any user defined data, there may not be initial states in that categorical state space.

Proposition 4.1.1. Let $F : \mathcal{D} \to C$ be a functor between locally small categories $\mathcal{D}$ and C with initial objects $\varnothing_{\mathcal{D}}$ and $\varnothing_C = F(\varnothing_{\mathcal{D}})$. F is an opfibration only if it has a left adjoint $L \dashv F$ such that $FL = id_C$.

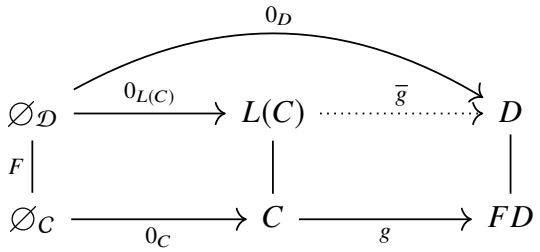
Proof. F being an opfibration admits a cleavage, that is, for every morphism $\alpha : FD \to C$ in C, a choice of opcartesian lifting $\alpha^{\uparrow} : D \to D^{\uparrow\alpha}$. We can then define L explicitly on objects by the mapping $C \mapsto \varnothing_{\mathcal{D}}^{\uparrow 0_C}$, the target of the opcartesian lifting of $0_C : F(\varnothing_{\mathcal{D}}) \to C$ in C (with respect to our cleavage of F). Then, for any morphism $f : C \to C'$ in C we take Lf to be the following unique, dotted morphism given by universal mapping property of opcartesian lfits:



(4.1)

L is functorial as $L(id_C) = id_{L(C)}$ by definition and for any sequence $C \xrightarrow{f} C' \xrightarrow{g} C''$ that $L(g \circ f) = L(g) \circ L(f)$ by the uniqueness of morphisms satisfying the universal mapping property of (4.1).

To show L induces a hom-set adjunction with respect to G, for any $D \in \mathcal{D}$ and $g : C \to FD$, there is a morphism $\overline{g} : L(C) \to D$ given by the universal mapping property as:



(4.2)

We then need to show the family $(\Phi_{C,D} : Hom(C, GD) \to Hom(FC, D))_{C \in C,\ D \in \mathcal{D}}$, where $\Phi_{C,D}$ is given by the mapping $(-) \mapsto \overline{(-)}$, is a family of bijections satisfying the naturality condition given by (4.3). To show $\Phi_{C,D}$ is bijective for any C and D, the universal property satisfied by the assignment $(-) \mapsto \overline{(-)}$ implies that $\Phi_{C,D}$ is injective. Then, $\Phi_{C,D}$ is surjective as well because for any morphism $h : L(C) \to D$, we must have that $h = \overline{Fh}$ because $h \circ 0_{L(C)} = 0_D$ as there is only one morphism in $Hom_{\mathcal{D}}(\varnothing_{\mathcal{D}}, D)$. Hence, $\Phi_{C,D}$ is a bijection for every $C \in C$ and $D \in \mathcal{D}$.

It remains to show for any $f : D \to D'$ and $g : C \to C'$ we have the following, commutative mappings along the perimeter of the inner, naturality square:

$$\begin{array}{ccccc}
h & \longmapsto & & & \overline{h} \\
\Big\downarrow & & & & \Big\downarrow \\
 & & Hom(C, FD) \xrightarrow{\Phi_{C,D}} Hom(L(C), D) & & \\
 & & \Big\downarrow{\scriptstyle Hom(g,Ff)} \qquad \Big\downarrow{\scriptstyle Hom(Lg,f)} & & \\
 & & Hom(C', FD') \xrightarrow[\Phi_{C',D'}]{} Hom(L(C'), D') & & \\
\Big\downarrow & & & & \Big\downarrow \\
Ff \circ h \circ g & \longmapsto & \overline{Ff \circ h \circ g} & = & f \circ \overline{h} \circ Lg
\end{array} \tag{4.3}$$

We can show that $\overline{Ff \circ h \circ g} = f \circ \overline{h} \circ Lg$ because

$$F(f \circ \overline{h} \circ Lg) = Ff \circ F(\overline{h}) \circ F(Lg) = Ff \circ h \circ g$$

And by the universal mapping property of opcartesian lifts, $\overline{Ff \circ h \circ g}$ is unique such that $F(\overline{Ff \circ h \circ g}) = Ff \circ h \circ g$ hence $\overline{Ff \circ h \circ g} = f \circ \overline{h} \circ Lg$ and thus $L \dashv F$ such that $FL = id_C$ by (4.1).

□

We can reverse the argument of the above proof to show:

Corollary 4.1.2. Let $F : \mathcal{D} \to C$ be an initial object preserving functor between locally small categories. If F has a left adjoint $L \dashv F$ such that $FL = id_C$, then, for every object $C \in C$: $0_{L(C)} : \varnothing_{\mathcal{D}} \to L(C)$ is an opcartesian lift of $0_C : \varnothing_C \to C$ and L is defined on morphisms as given by (4.1).

Thus for initial object preserving functors F, we have determined precisely when they have a left invertible, left adjoint $L \dashv F$ and how L is defined – up to isomorphism.

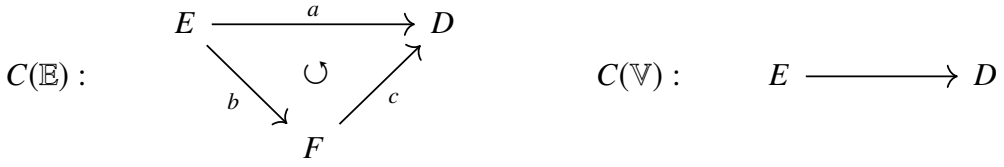
There is of course a dual result for terminal object preserving functors relating cartesian lifts to right adjoints. However, for categorical state spaces, terminal states are rarely tenable. A terminal state would have to be a state with a maximal amount of data that can be inserted into it, for which all states can be updated to it in only one way.

Interpreting Proposition (4.1.1) in the view-updating context: when F is defined on categorical state spaces with monotonic updates, the interpretation of L given by Proposition (4.1.1) is that: for every view-state $C \in C$: $L(C)$ is above all, the source-state containing the *minimal* amount of data in it needed to derive C from the view get F. Note, where the term 'minimal' is informally used in relation to the initial objects $\varnothing_{\mathcal{D}}$ and $\varnothing_C$; finding correspondences between the action of L and something similar enough to opcartesian lifts to generalise the main results of this chapter is a future interest that will be discussed towards the end of this thesis.

An immediate use Proposition (4.1.1) is as a means to showing a view-get cannot be an opfibration – That is, a view-get cannot be an opfibration if it doesn't have a left adjoint. This places a strict upper bound on how satisfying a solution to the view-update can be given for that view-get. An example of this use is given as follows:

Example 4.1.3. (*Forgetting Commutative Diagrams*)

Consider a view definition of sketches given by $V : \mathbb{V} \to \mathbb{E}$ where $\mathbb{V}$ and $\mathbb{E}$ with underlying categories:



And the obvious choice of view definition V^*.

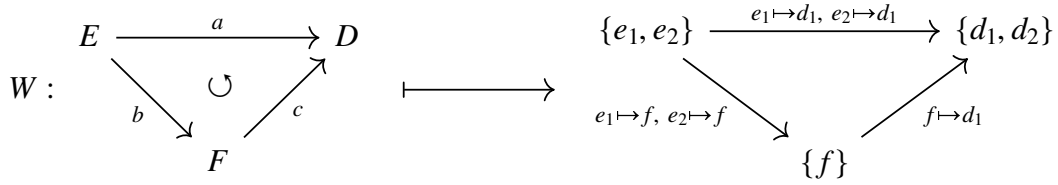
$Mod(\mathbb{E}, \mathbf{Set})$ and $Mod(\mathbb{V}, \mathbf{Set})$ have initial objects given by $\varnothing_{\mathbb{E}} : C(\mathbb{E}) \to \mathbf{Set}$ and $\varnothing_{\mathbb{V}} : C(\mathbb{V}) \to \mathbf{Set}$ where $\varnothing_{\mathbb{E}}(X) = \varnothing$ for every $X \in C(\mathbb{E})$ and likewise for $\varnothing_{\mathbb{V}}$ and V^* is thus initial object preserving.

Assume then that V^* is an opfibration and hence for the following state $V \in \mathcal{B}$ given by:

$$V: \; E \longrightarrow D \quad \longmapsto \quad \{e_1, e_2\} \xrightarrow{e_1 \mapsto d_1,\ e_2 \mapsto d_2} \{d_1, d_2\}$$

There is an opcartesian lift for the morphism $0_V : V^*(\varnothing_{\mathbb{E}}) \to V$, given by a morphism $\varnothing_{\mathbb{E}} \to \varnothing^{\uparrow}_{\mathbb{E}}$. Consequently we must have that $|\varnothing^{\uparrow}_{\mathbb{E}}(F)| > 1$ or else we contradict the commutativity of $C(\mathbb{E})$.

However, if we consider the state $W \in Mod(\mathbb{E}, \mathbf{Set})$ given by:



We find that there is a morphism $V \to V^*W$ in $Mod(\mathbb{V}, \mathbf{Set})$ but no morphism $\varnothing^{\uparrow}_{\mathbb{E}} \to W$ as any choice of natural transformation $V \Rightarrow V^*W$ yields that the naturality square on c fails to commutes. Hence V^* is not opfibration; equivalently we can not form the left invertible, left adjoint to V^*.

We find this example elusive as letting $\mathcal{A}$ be a category such that $ob(\mathcal{A}) = ob(Mod(\mathbb{E}, \mathbf{Set}))$, if we take morphisms between $M, N \in Mod(\mathbb{E}, \mathbf{Set})$ to be any collection $(\alpha_X : MX \to NX)_{X \in C(\mathbb{E})}$, then taking $W' \in \mathcal{A}$ with $|W'(F)| > 1$ yields multiple distinct morphisms $LV \to W'$ so that 0_{LV} can't be an opcartesian lift (this is clear by noticing $Hom_{\mathcal{A}}(LV, W) \ncong Hom_{Mod(\mathbb{V}, \mathbf{Set})}(V, V^*W)$. If we instead try represent all our updates between M and N be a collection of injections $(\beta_X : MX \rightarrowtail NX)_{X \in C(\mathbb{E})}$, we still find $Hom_{\mathcal{A}}(LV, W) = \varnothing$ and hence $L \not\dashv V^*$. Hence we find in this simple example, something deeper seems to be preventing an opfibrational view-get existing.
■

There are uses for these sorts of negative result in solving view-update problems where, depending on how optimistic a database designer is about the existence of opfibrational view-gets for their needs, these results indicate when the designs of source and view state spaces need to be reconsidered. However, we will instead begin moving towards positive results for solving the view-update problem, particularly, the results of sections (4.4) and (4.5) that provide cases where opcartesian and cartesian lifts can be computed using the left adjoint of the respective view-get.

Before proceeding, the reader should keep in mind that if the nature of left adjoints, pushouts and products are clear in their context of interest in cartesian and opcartesian lifts, they can proceed straight to theorems (4.4.2) and (4.5.5). The content of the next two sections above all is to produce scaffolding to contextualise the elementary nature and application of these theorems.

4.2 A Rudimentary Categorical Data Model

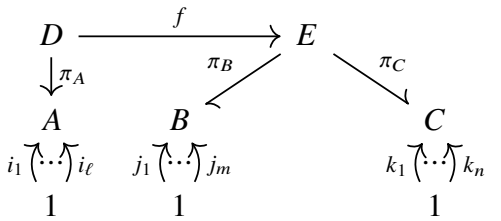
Until Chapter 6, we will regard the purpose of defining the categorical data model of this section as means of forming categorical state spaces that are 'lightweight' in the sense, a select few features of database schemas and sates are encoded, while many other features are invisible with respect to the categorical structure.

To begin, each state (object in the categorical state space) contains its schema, where entities and entity relations are specified; the equivalent notion for the Sketch Data Model is that a state, a model $M \in Mod(\mathbb{E}, \mathbf{Set})$ comes with a sketch $\mathbb{E}$, where we can identify the tables and the functional relations between them – independently from the instances of those tables. However, unlike the Sketch Data Model, we will not generally be able to identify attribute domains in the schema nor identify an entity as being constructed out of others using standard queries.

Counter to that lack of information typically specified for a database state space, we will be able to decide which attributes values can be in-place edited. Also, with standard categorical constructions we will be able to identify states whose entity-instances are given by the *intersection* or *(non-disjoint) union* of those entity-instances of other states.

While, this new categorical data model won't be dependent on sketches for specifications, we will motivate its definition by bastardising a category of models on a sketch to a category that allows in-place editing; that in turn will be a categorical state space of our new categorical data model.

Example 4.2.1. Consider the sketch $\mathbb{E}$ with underlying category:



Where we specify 1 as the vertex of a cone over the empty diagram; A, B and C as attributes and the harpoon arrows denoting partial functions of the form $E \xleftarrow{d_B} B_{def} \xrightarrow{\pi_B} B_m$, where the monic arrow is appropriately specified as a cone (see: Proposition (3.1.4)).

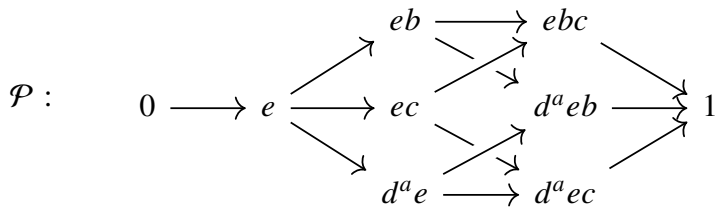
One way of describing models $M :\in Mod(\mathbb{E}, \mathbf{Set})$ is as a collection of models $\mathscr{C}(M)$ of the form $S \in Mod(\mathbb{E}, \mathbf{Set})$ where $|SD| \leqslant |SE| \leqslant 1$. That is, states that contain *no more information than a tuple* of either the relation $ME \subset MB_{def} \times MC_{def}$ or the functional relation $Mf \subset MD \times ME$ –beating in mind a tuple of MD can't exist without the information of its related tuple in ME. We call these states *records* (of $\mathbb{E}$).

We take $\mathscr{C}(M)$ to consist only of the following records of $\mathbb{E}$ – noting the use of $\subset$ to denote a subfunctor of model (a set-valued functor):

- For every $e \in ME$, we include the record $S^e \subset M$ in $\mathscr{C}(M)$, where $S^e(D) = \varnothing$ and $S^e(E) = \{e\}$. Note that rest of S^e is determined by this choice because it is both a subfunctor of M and model of $\mathbb{E}$.

- For every $d \in MD$, we similarly include the record $S^d \subset M$ in $\mathscr{C}(M)$, where $S^d(D) = \{d\}$, $S^d(E) = \{Mf(d)\}$. Likewise, the rest of S^d is thus determined.

We define the schema of our new categorical state space as the poset of subobjects for the maximally defined records. That is, a record with an instanced defined for every entity, with all attribute valued defined. The maximally defined record of $\mathbb{E}$ are those of the form S^f such that $|S^f(B_{def})| = |S^f(C_{def})| = 1$; the poset of subobjects for any of these records can be given by the following category:



Where we read the poset elements as *entity instances, raised to their non-nullable attribute values, followed by their nullable attribute values that are defined*. For example, $d^a ec$ corresponds to the record defined at D and hence E; and has A and C attribute values but has a null B attribute value. 0 denotes the record where D and E map to the empty set and 1 denotes a maximally defined record. We denote the poset that $\mathcal{P}$ corresponds to by $(P, \leqslant)$.

Now we can then collect and organise the records of $\mathbb{E}$ by a contravariant functor $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Set}$, where on objects $\mathbf{U}$ maps $p \in \mathcal{P}$ to the set of all records defined precisely on p. E.g.

$$S \in \mathbf{U}(d^a eb) \iff |S(X)| = \begin{cases} 1 & \text{if } X = D, E \text{ or } B_{def} \\ 0 & \text{otherwise.} \end{cases}$$

Then because $Hom_{\mathcal{P}^{op}}(p, q)$ has at most one morphism corresponding to $p \geqslant q$ in P, we denote $\mathbf{U}(p \geqslant q)$ as the action of $\mathbf{U}$ on that morphism, defined by the mapping taking each $S \in \mathbf{U}(p)$ to the unique record $R \in \mathbf{U}(q)$ such that R is a subfunctor of S.[1]

In fact, now we can associate M with a subfunctor of $\mathbf{M} \subset \mathbf{U}$, where $\forall p \in \mathcal{P}$: $S \in \mathbf{M}(p) \iff S$ is a subfunctor of M. However, the question of how to represent updates of states by some map between subfunctors of $\mathbf{U}$. For subfunctors $\mathbf{M}, \mathbf{N} \subset \mathbf{U}$ remains. It turns out that letting morphisms $\alpha : \mathbf{M} \to \mathbf{N}$ consist of families $(\alpha_p : \mathbf{M}(p) \to \mathbf{N}(p))_{p \in \mathcal{P}}$ – however relaxed or constrained – leaves us no better than relaxing or constraining families of morphisms $(\beta_X : MX \to NX)_{X \in C(\mathbb{E})}$.

The preferred option is to instead take $\mathbf{U}$ to be a functor $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$. The idea is that $\forall p \in \mathcal{P} : \mathbf{U}(p)$ consists of all p-records with morphisms $k \to k'$ in $\mathbf{U}(p)$ if the *record k can be updated to the record k'*. Functoriality then requires that there is a morphism $k \to k'$ only if it is updatable at every *sub-record level of $\mathcal{P}$*. That is, $\forall p \geqslant q$ in $\mathcal{P}^{op}$: $\exists\, \mathbf{U}(p \geqslant q)(k) \to \mathbf{U}(p \geqslant q)(k')$ in $\mathbf{U}(q)$.

[1] Note that the collection of records of a sketch can be a proper class rather than a set as the set of singletons is not a set. Care has been taken to not operate on such functors in any way that assumes the category of records, with at most one morphism per objects is small.

We then populate $\mathbf{U}(p)$ with arrows based on the kind of state space we desire. For example, consider that we want to allow in-place edits for C attribute values but not B attribute values. We implement this by requiring for every pair $S, T \in \mathbf{U}(ebc)$ that $Hom_{\mathbf{U}(ebc)}(S, T) \neq \varnothing$ if and only if $S\pi_B(SB_{def}) = T\pi_B(TB_{def})$. This determines $\mathbf{U}(eb)$, $\mathbf{U}(ec)$, $\mathbf{U}(eb)$ and $\mathbf{U}(0)$.

■

To generally define our new categorical state spaces, we need an analogue of subfunctor for preorder valued functors:

Definition 4.2.2. For $\mathcal{P}$ a posetal category and $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ a functor, we define a *subfunctor* of $\mathbf{U}$ to be any functor $S : \mathcal{P}^{op} \to \mathbf{Pre}$ such that $S(p)$ is a full subcategory of $\mathbf{U}(p)$ and $\forall p \geqslant q$ in P we have that $S(p \geqslant q)$ is given by $\mathbf{U}(p \geqslant q)$ restricted to $S(p)$.

Note if we wish to describe states on a sketch whose underlying category has non-commutative diagrams, it makes sense take our preorders of models to be those which map objects to sets with more than one element.

Example 4.2.3. Let $\mathbb{M}$ be a sketch with underlying category  for an entity married people: P and an injective self-relation denoting marriage as $m : P \rightarrowtail P$ such that the following hold: $m \neq id_P$ and $m \circ m = id_P$ to characterise that people aren't married to themselves and if a person is married to another person, that person is married to them; injectivity disallows polygamy in this example.

In this scheme we let $\mathbf{1}$ denote the one object poset with element 1 and $V : \mathbf{1}^{op} \to \mathbf{Pre}$ pick out a preorder $V(1)$ of models in $Mod(\mathbb{M}, \mathbf{Set})$, where $S \in V(1)$ if and only if $SP = \{a, b\}$. Moreover, it must be the case $Sm(a) = b$ and $Sm(b) = a$, then any model $M \in Mod(\mathbb{M}, \mathbf{Set})$, again corresponds to a subfunctor $\mathbf{M} \subset V$.
■

Now we can define these sorts of state spaces more generally by:

Definition 4.2.4. For a posetal category $\mathcal{P}$ corresponding to $(P, \leqslant)$ and functor $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$, we denote $ST(\mathbf{U})$ as the category consisting of subfunctors of $\mathbf{U}$ as objects and for subfunctors $S, T \subset \mathbf{U}$: a morphism $\alpha : S \to T$ is given by a family of morphisms $(\alpha_p : S(p) \to T(p))_{p \in \mathcal{P}}$ satisfying:

$$\forall p \in \mathcal{P},\ k \in S(p) :\ k \preceq \alpha_p(k) \text{ in } \mathbf{U}(p) \tag{4.4}$$

That is, records in the source state of an update have to be updated to records that are deemed updatable by $\mathbf{U}$. However, note that when $ST(\mathbf{U})$ is built out of records of a sketch – objects in $ST(\mathbf{U})$ may not correspond to valid models on that sketch using our original association – as seen in the following example:

Example 4.2.5. (Example (4.2.1) Continued)
For $ST(\mathbf{U})$ and $\mathbf{U}$ derived from records of $\mathbb{E}$ as before, we can construct a state $S \in ST(\mathbf{U})$ with $S(d^a eb) = \{S_1, S_2\}$ consisting of the same e-entity, that is, $\mathbf{U}(d^a eb \geqslant e)(S_1) = \mathbf{U}(d^a eb \geqslant e)(S_2)$ but with different B values, that is $\mathbf{U}(d^a eb \geqslant eb)(S_1) \neq \mathbf{U}(d^a eb \geqslant eb)(S_2)$. However, if $\exists S \in Mod(\mathbb{E}, \mathbf{Set})$ such that $S_1, S_2 \subset S$ then $S(E) = S_1(E) = S_2(E)$ but $|S(B_{def})| = 2$, contradicting that Sd_B is injective.
■

Hence we tend to deal in full subcategories of the form $\mathcal{S} \subset ST(\mathbf{U})$ and are interested if say, $\mathcal{S}$ inherits certain (co)limits from $ST(\mathbf{U})$. We discuss shortcomings of describing categorical state spaces in this way in Chapter 6 and note it may be interesting in the future to explore restrictions on categories $ST(\mathbf{U})$ to prevent asymmetries such as in Example (4.2.5) from forming.

Conversely, an important lack of restriction on $ST(\mathbf{U})$ is that there is no naturality condition with respect to $\mathcal{P}$ on morphisms. This is because if for some $p \geqslant q$ in $\mathcal{P}^{op}$ we understand the morphism $\mathbf{U}(p \geqslant q) : \mathbf{U}(p) \to \mathbf{U}(q)$ as primitively denoting a relation between a system of entities at the p^{th} *record level* level to its q^{th} *record level*; naturality on $\alpha : S \to T$ requiring:

$$\forall k \in S(p) : \ \alpha_q \circ \mathbf{U}(p \geqslant q)(k) = \mathbf{U}(p \geqslant q) \circ \alpha_p(k)$$

Which in applications means that updates can't change relations. This is the restriction we observed on updates on the Sketch Data Model between *keyed* sketches, for which we don't want to exclusively study. A tempting weakening of naturality is that:

$$\forall k \in S(p) : \alpha_q \circ S(p \geqslant q)(k) \cong T(p \geqslant q) \circ \alpha_p(k)$$

This was what we found was required for updates in the Sketch Data Model, for sketches that aren't necessarily *keyed*, recalling foreign keys could only be permuted if their attribute values are left unchanged with respect to the relation.

Finally, denoting $\preceq_q$ as the relation on $\mathbf{U}(q)$ for any $q \in \mathcal{P}$ we may further try to weaken naturality to:

$$\forall k \in S(p) : \alpha_q \circ S(p \geqslant q)(k) \preceq_q T(p \geqslant q) \circ \alpha_p(k)$$

That is, a p-record is updatable only if every q-record in it is updatable. An example of when couldn't require that is if $\mathbf{U}(q)$ is to only allow updates that fix keys but $\mathbf{U}(p)$ is to allowed to edit relations.

We now formalise the association in Example (4.2.1) between a record, as in an object in $\mathbf{U}(p)$ for some $p \in \mathcal{P}$ and a state in the existing categorical state space with only that tuple as data.

Definition 4.2.6. We say an object $\tau \in ST(\mathbf{U})$ is a *record object* if $\exists p_\tau \in \mathcal{P}$ such that $\forall q \in \mathcal{P}$:

$$|\tau(q)| = \begin{cases} 1 & \text{If } q \leqslant p_\tau \text{ in } \mathcal{P}. \\ 0 & \text{Otherwise.} \end{cases} \tag{4.5}$$

We denote $Rec(\mathbf{U}) \subset ST(\mathbf{U})$ as the full subcategory consisting of record objects in $ST(\mathbf{U})$.

For any $\varphi \in Rec(\mathbf{U})$ we denote $p_\varphi \in \mathcal{P}$ as the unique element for which (4.5) holds with respect to φ. The following is a categorically motivated understanding of objects $\tau \in Rec(\mathbf{U})$ and p_τ:

Proposition 4.2.7. Let $U : \mathbf{Pre} \to \mathbf{Set}$ be the forgetful functor on preorders. $\tau \in ST(\mathbf{U})$ is a record object *if and only if* $U \circ \tau : \mathcal{P}^{op} \to \mathbf{Set}$ is *representable* with respect to p_τ.

Proof. ($\Longrightarrow$) Since $U \circ \tau$ is contravariant, it is representable with respect to p_τ if and only if there is a natural isomorphism $Hom(-, p_\tau) \cong U \circ \tau$. For any $q \in \mathcal{P}$, if $q \leqslant p_\tau$ then $Hom(q, p_\tau) = \{q \leqslant p_\tau\} \cong U \circ \tau(q)$ *if and only if* $\tau(q)$ is a singleton contained in $\mathbf{U}(q)$. On the other hand if $q \not\leqslant p_\tau$ then $Hom(q, p_\tau) = \varnothing = U \circ \tau(q)$, *if and only if* $\tau(q) = \varnothing$; thus together, τ is a record object by definition. □

Another manner of describing this is that $\tau \in Rec(\mathbf{U})$ precisely when $U \circ \tau$ is the representation of p_τ with universal element for which $U \circ \tau(p_\tau)$ is a singleton of.

Proposition 4.2.8. For $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ and $\tau, \sigma \in Rec(\mathbf{U})$: $Hom_{ST(\mathbf{U})}(\sigma, \tau) \neq \varnothing$ only if $p_\tau \leqslant p_\sigma$.

Proof. For $\tau, \sigma \in Rec(\mathbf{U})$ such that $p_\tau > p_\sigma$ in $\mathcal{P}$ assume rather, that there is a morphism $\alpha : \tau \to \sigma$ in $ST(\mathbf{U})$ then α has a component $\alpha_{p_\tau} : \tau(p_\tau) \to \sigma_{p_\tau}$, which is a morphisms between preorders. However, $\tau(p_\tau)$ is the one object preorder and σ_{p_τ} is the empty preorder because $p_\sigma < p_\sigma$; hence α_{p_τ} consists of a function from a non-empty set into the empty set, which is a contradiction on α existing. □

Record objects and more interesting constructions will be used in Chapter 5. but we proceed with just the elementary definitions of our new categorical data model, only furthering the theory of this model by describing products and pushouts in categories of the form $ST(\mathbf{U})$.

4.3 The Main Example

To derive the example of this section, we firstly define two properties binary relations can have:

Definition 4.3.1. Let $R \subset A \times B$ be a binary relation and write aRb to denote $(a, b) \in R$. We say that R is:

- *Injective* if $\forall x, y \in A$ and $z \in B$: xRz and $yRz \implies x = y$.
- *surjective* if $\forall z \in B : \exists x \in A$ such that xRz.

These properties are also referred to as “left unique” and “right total”, respectively.

The following example describes the schema of a simple database and a view on it. We then discern what kind of categories the source and view state spaces ought to be by requiring that the view-get has a left adjoint. In turn, the view-get will be an opfibration and a fibration, which will be made evident by the main results of this chapter. Note, while deriving this example will be rather involved, successive examples of how cartesian and opcartesian lifts are computer will be brief – as we will have Theorem (4.4.2) and (4.5.5) to prove those claims.

Example 4.3.2. (Forgetting Constrained Relations)

In this example we model a database with a table of students, a table of supervisors and an injective but not surjective binary relation from the supervisors table to the students table. Note that we will define these constraints and several more for a functor $\mathbf{U}_0 : \mathcal{P}^{op} \to \mathbf{Pre}$ but will outline them in prose first.

In other words, these first two constraints on the relation can be read as:

1. A student can be supervised by at most one supervisor (Injectivity of the binary relation)
2. There are students that cannot be supervised (The binary relation is not surjective)

The idea is that we can partition the students into ‘graduates’ (those that can be supervised) and ‘undergraduates’ (those that cannot). For simplicity, we will have:

3. If a student can be supervised, they can be supervised by any supervisor. That is, for any student g with at least one state where g is supervised, any valid state with g can have added to it any supervision relationship, with one of its supervisors and g.

And to make the partition of students clearer:

4. If a student can be supervised, they must be supervised by someone. That is, for any student g: if there is at least one state such that g has a supervisor s, there is no state with g as a student with no supervisor.

We form a categorical state space satisfying these requirement as follows: we defining a poset $\mathcal{P}$ given by the category $Spvs \to Spvn \leftarrow Stud$ representing a relation between supervisor entity instances and student entity instances. Then, for functor $\mathbf{U}_0 : \mathcal{P}^{op} \to \mathbf{Pre}$, $ST(\mathbf{U}_0)$ satisfies the constraints 1-3 if we require (minding the abuse of notation, performing set theoretic operations on sets of preorder elements):

1. $\mathbf{U}_0(Spvn) \subset \mathbf{U}_0(Spvs) \times \mathbf{U}_0(Stud)$ and that this is an injective binary relation.
2. $Grad \coloneqq \mathbf{U}_0(Stud) \backslash im(\mathbf{U}_0(Spvn \geqslant Stud)) \neq \varnothing$.
3. $im(\mathbf{U}_0(Spvn \geqslant Spvs)) = \mathbf{U}_0(Spvs)$.

To satisfy constraint 4. we need to take our state space to be the subcategory $\mathcal{S} \subset ST(\mathbf{U}_0)$, where $S \in \mathcal{S}$ if and only if the following holds:

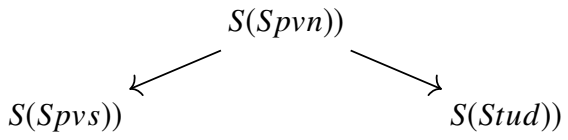
4. $\forall g \in Grad \cap S(Stud) : \exists sRg \in S(Spvn)$ such that $\mathbf{U}_0(Spvn \geqslant Stud)(sRg) = g$.

We constrain the preorders in the image of $\mathbf{U}_0$ and $\mathcal{S}$ in the following ways, to make the meaning of the updates clearer:

5. Undergraduates can be updated to Graduates but not visa versa, that is: $x \preceq_{Stud} y$ only if $x \in Grad \implies y \in Grad$
6. Updates must fix the student component of supervision relations, that is: $Hom_{\mathcal{S}}(S,T)$ only has morphisms in $\alpha \in ST(\mathbf{U}_0)$ satisfying: $T(Spvn \geqslant Stud) \circ \alpha_{Spvn} = \alpha_{Stud} \circ S(Spvn \geqslant Stud)$.

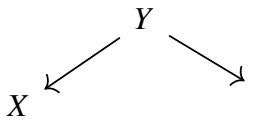
We will proceed to define a view-update problem on $\mathcal{S}$ by taking the view just displaying the students of our database. That is, the state space $\mathcal{V} \coloneqq ST(\mathbf{U}_1)$, for a functor $\mathbf{U}_1 : 1 \to \mathbf{Pre}$ from the one object poset 1 such that $\mathbf{U}_1(1) = \mathbf{U}_0(Stud)$. We can then define the view-get $G : \mathcal{S} \to \mathcal{V}$ as the obvious projection.

Before looking for a left adjoint of G, we will introduce a last notational convenience for states $S \in \mathcal{S}$, drawing them as:



And the arrows mapping according to $\mathbf{U}_0(- \geqslant -)$.

Note that $\mathcal{S}$ and $\mathcal{V}$ contain the empty functor and so that G is initial object preserving. Hence by Proposition (4.1.1), we can define a left adjoint $L \dashv G$ by finding (targets of) opcartesian lifts of $0_V : \varnothing_{\mathcal{V}} \to V$ for each $V \in \mathcal{V}$. Indeed, the target of any lift $\varnothing^{\uparrow 0_V}$ has the form:



Where there is a bijection between Y and $V(1) \cap Grad$ because of constraints 1 and 3 and X is empty if and only if $Y = \varnothing$ because S is a functor. We now seek to discern some facts about $\mathcal{S}$, requiring $Hom_{\mathcal{S}}(LV, W) \cong Hom_{\mathcal{V}}(V, GW)$ for any $W \in \mathcal{S}$ by the adjunction.

Consider any $h \in Hom(V, GW)$, which is an order preserving morphism $h : V(1) \to W(Stud)$. Because $GLV = V$, we have $V(1) = LV(1)$. The corresponding morphism $\overline{h} \in Hom(LV, W)$ such that $G\overline{h} = h$ must be given by a triple (f, g, h), where $f : X \to W(Spvs)$ and $g : Y \to W(Spvn)$ are order preserving morphisms satisfying 6, among the condition on morphisms given in Definition (4.2.4).

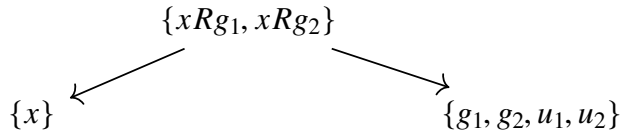
Since f and g don't contribute to the action of G on $\overline{h}$, requiring the hom-set adjunction holds means that f and g must be uniquely determined. We already know how g must be defined by the naturality condition given by constraint 6 but to make sure there is only one possibility of how to define f, we will use the fact that X is a singleton in the case $Y \neq \varnothing$. The argument is as follows:

Assume instead that X has at least two distinct elements $x, x' \in X$. Then, we define $T \in \mathcal{S}$ to be the state with $T(Spvn) = LV(Spvn)$, $T(Stud) = LV(Stud)$ and $T(Spvs) = \{x\}$. Because $Hom_{\mathcal{S}}(LV, T) \cong Hom_{\mathcal{V}}(V, GT) = Hom(V, V)$ we must have $x' \preceq_{Spvs} x$ for there to be a morphism $\overline{id_V} \in Hom_{\mathcal{S}}(LV, T)$ such that $G\overline{id_V} = id_V$. However, $x' \preceq_{Spvs} x$ implies there are four maps in $Hom_{\mathcal{S}}(LV, LV)$ for every map in $Hom_{\mathcal{V}}(V, V)$ sending $x \mapsto x'$ or x, and $x' \mapsto x'$ or x. This contradicts $L \dashv G$ and hence $|X| = 1$ when $Y \neq \varnothing$.

Knowing this, we are then forced to require that for every supervisor in $W(Spvs)$ there is one and only one supervisor $y \in W(Spvs)$ such that $x \preceq_{Spvs} y$. This amounts to saying: every state in $\mathcal{S}$ with at least one graduate student has a single *default supervisor* that updates have to fix; the action of L assigns any single default supervisor to all new graduate students added in the view.

Now adding a default supervisor for view-inserts from $\varnothing_{\mathcal{V}}$ isn't a very satisfying solution to the view update problem as it deprecates the real world meaning of whom and what a supervisor *really is* – we will discuss how to formulate appropriate null supervisors in Chapter 6. Regardless, we have our left adjoint and can discern what opcartesian must look like in the next section.

To see L in action, say $V = \{g_1, g_2, u_1, u_2\}$ for $g_1, g_2 \in Grad$ and $u_1, u_2 \in \mathbf{U}_0(Stud) \backslash Grad$, then LV is given by:

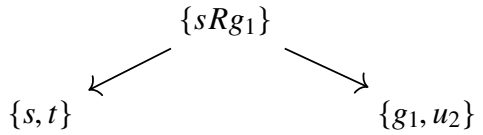


This concludes this Section's derivation of the main example, Example (4.3.2). Over the page, we extend this example to briefly show a case of an opcartesian lift, describing the basic ethos of how it's categorically constructed. We are spared from proving that the opcartesian lift is structured in the way we claim because we will successively prove the main, general Category Theory result: Theorem (4.4.2) and then show Proposition (4.4.3), which shows more explicitly how the main Categorical instrument of this construction – pushouts – are computed for categories of the form $ST(\mathbf{U})$. Hopefully then, it is clear that our claim in the continuation of this example is true.

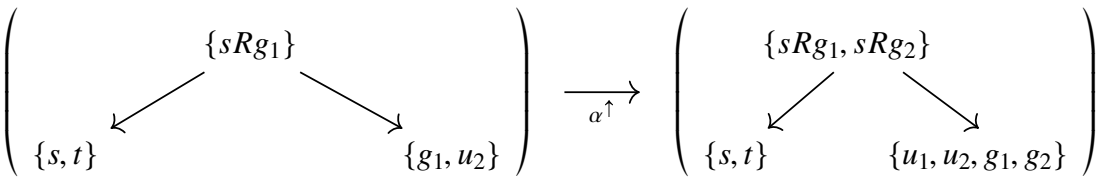
4.4 Opcartesian View-Updating

The purpose of this section is to show how opcartesian lifts can be computed for functors (view-gets) $G : \mathcal{S} \to \mathcal{V}$ with left adjoints $L \dashv G$ such that $GL = id_{\mathcal{V}}$. The basic idea is: given a view-update $\alpha : GS \to V$ in $\mathcal{B}$, to form a propagation $S \to S^{\uparrow}$ in $\mathcal{S}$, $S^{\uparrow}$ is formed by *appropriately stitching the existing information in S with the minimal data required to represent V, that is, LV*. We briefly show what this looks like continuing Example (4.3.2).

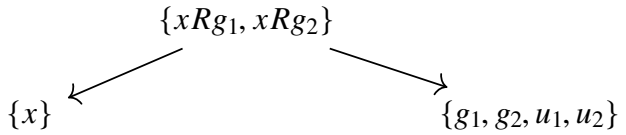
Example 4.4.1. (Example (4.3.2) Continued) Consider a source state in $S \in \mathcal{S}$ given by:



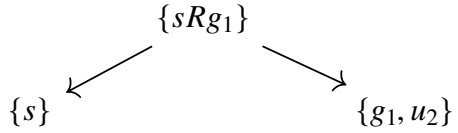
For $g_1 \in Grad$ and $u_2 \in \mathbf{U}_0(Stud) \backslash Grad$. Then, we define a view update $\alpha : GS \to V$, for V as before, given by $\{g_1, g_2, u_1, u_2\}$. An opcartesian lift $\alpha^{\uparrow} : S \to S^{\uparrow}$ of α is given by the following map:



Where the default supervisor we give to g_2 in $S^{\uparrow}$ is s because that is the default supervisor in S. Furthermore, we find there is an obvious inclusion $LV \hookrightarrow S^{\uparrow}$, matching default supervisors $x \mapsto s$. We recall that LV is given by:



And note, although a tempting guess, $S^{\uparrow}$ is not the coproduct of S and LV. Instead, we note there is a state that maps to both S and LV, given by LGS as:



Where we have the inclusion (counit of $L \dashv G$) $\epsilon_S : LGS \to S$ and the inclusion $L\alpha : LGS \to LV$. There is a pushout of these two morphisms given by a pair $\alpha^{\uparrow} : S \to S^{\uparrow}$ (which turns out to be an opcartesian lift of α) and another morphism $j : LV \to S^{\uparrow}$, which we'll understand better in Theorem (4.4.2) following this example.

Note, as pushouts and opcartesian lifts are given up to isomorphism, we find that we could have also taken $S^{\uparrow}(Spvn)$ to be given by $\{tRg_1, sRg_2\}$ – Yet it is bad policy to randomly edit the supervisor of g_1 in the event of α suggesting no such change. This demonstrates the alluded fact that the meaning of assigning/updating supervisors is compromised in this example because of the previously mentioned notion of a *default supervisor.* We will remedy this issue in Section (5.2).

Theorem 4.4.2. *Let $G : \mathcal{A} \to \mathcal{B}$ be a functor with a left adjoint $L \dashv G$ such that $GL = id_{\mathcal{B}}$.*

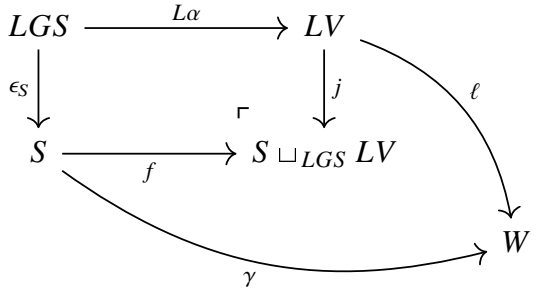
If we define $\forall S \in S$: $\epsilon_S : LGS \to S$ to be the counit of $L \dashv G$ then for any morphism $\alpha : GS \to V$ in $\mathcal{B}$:

α has an opcartesian lift if and only if the following pushout exists in $\mathcal{A}$:

$$\begin{array}{ccc} LGS & \xrightarrow{L\alpha} & LV \\ {\scriptstyle \epsilon_S}\downarrow & & \downarrow{\scriptstyle j} \\ S & \xrightarrow[f]{} & S \sqcup_{LGS} LV \end{array} \tag{4.6}$$

For $j = \Phi_{(V, S \sqcup_{L(GS)} LV)}(id_V)$.

Proof. ($\Longleftarrow$) I claim for any pushout of the form (4.6) that f is an opcartesian lift of α. Indeed, for any $\gamma : S \to W$ in $\mathcal{A}$ such that $G\gamma = \beta \circ \alpha$, we note the following commutes:



(4.7)

Where $\ell = \Phi_{(V,W)}(\beta)$.

This is because $G(\ell \circ L\alpha) = G(\ell) \circ G(L\alpha) = \beta \circ \alpha$ but also $G(\gamma \circ \epsilon_S) = G(\gamma) \circ G(\epsilon_S) = (\beta \circ \alpha) \circ id_{GS} = \beta \circ \alpha$. Thus the definition of the hom-set adjunction $Hom_{\mathcal{A}}(LGS, W) \cong Hom_{\mathcal{B}}(GLGS, GW) = Hom_{\mathcal{B}}(GS, GW)$ implies that $\gamma \circ \epsilon_S = \Phi_{(GS,W)}(\beta \circ \alpha) = \ell \circ L\alpha$ (see: the proof of Proposition (4.1.1)). Hence γ and ℓ together, form a cocone of ϵ_S and $L\alpha$ so that by the universal mapping property of the pushout, there is a unique morphism $k : S \sqcup_{L(GS)} L(V) \to W$ commuting with (4.7).

Now, we seek to show that k satisfies the universal property of an opcartesian lift with respect to lift: f and an arbitrary γ. This involves first showing that $k \circ f = \gamma$ and $Gf = \beta$; and f is unique in doing so.

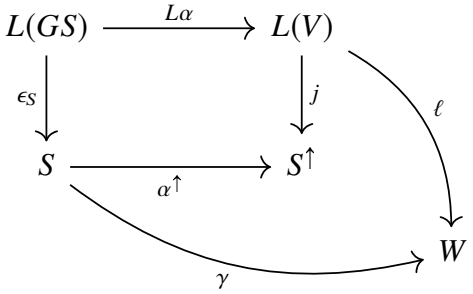
For the above equational conditions, we note: $G(k) = G(k) \circ id_V = G(k) \circ G(j) = G(k \circ j) = G(\ell) = \beta$ and $k \circ f = \gamma$ is given by the pushout property. For uniqueness, consider there another $k' : S \sqcup_{L(GS)} L(V) \to W$ satisfying $k' \circ f = \gamma$ and $Gk' = \beta$ then since this implies that $G(k' \circ j) = \beta$ by the hom-set adjunction $Hom_{\mathcal{A}}(LV, W) \cong Hom_{\mathcal{B}}(V, GW)$, it must be the case that $k' \circ j = \ell$. However, this implies that k' satisfies the universal property of the pushout (4.7) but k is unique in doing so, hence $k' = k$.

($\Longrightarrow$) Conversely, if α has an opcartesian lift $\alpha^{\uparrow} : s \to S^{\uparrow}$, we are to show the following diagram is a pushout:

$$\begin{array}{ccc} LGS & \xrightarrow{L\alpha} & LV \\ {\scriptstyle \epsilon_S}\downarrow & & \downarrow{\scriptstyle j} \\ S & \xrightarrow[\alpha^{\uparrow}]{} & S^{\uparrow} \end{array} \tag{4.8}$$

The square commutes because $G(\alpha^\uparrow \circ \epsilon_S) = G(\alpha^\uparrow) \circ G(\epsilon_S) = \alpha \circ id_{GS} = \alpha$ and $G(j \circ L\alpha) = G(j) \circ G(L\alpha) = id_V \circ \alpha = \alpha$, which by the hom-set adjunction $Hom_{\mathcal{A}}(LGS, S^\uparrow) \cong= Hom_{\mathcal{B}}(GLGS, GS^\uparrow) Hom_{\mathcal{B}}(GS, V)$ implies $\alpha^\uparrow \circ \epsilon_S = j \circ L\alpha$.

Moreover, the square is a pushout as considering any commuting diagram:



(4.9)

We have that $G\gamma = G\gamma \circ id_{GS} = G(\gamma \circ \epsilon_S) = G(\ell \circ L\alpha) = G(\ell) \circ \alpha$ implies by $\alpha^\uparrow$ being an opcartesian lift that there exists a unique $n : S^\uparrow \to W$ satisfying $n \circ \alpha^\uparrow = \gamma$ and $G(n) = G(\ell)$ and seek to show it satisfies the pushout property.

The commutativity conditions are given as $n \circ \alpha^\uparrow = \gamma$ and $Hom_{\mathcal{A}}(LV, W) \cong Hom_{\mathcal{B}}(V, GW)$ implies that $n \circ j = \ell$. Uniqueness is given by the fact any other morphism $m \in Hom(S^\uparrow, W)$ satisfying the pushout property must have $\gamma = m \circ \alpha^\uparrow$ and also $G(m \circ j) = G(\ell)$ by the definition of j, we know $G(\ell) = G(m \circ j) = G(m)$ so that m satisfies the commutativity properties of $\alpha^\uparrow$ as an opcartesian lift of α with respect to γ and $G\ell$ and know n is unique in doing so – hence $m = n$ and thus (4.9) is a pushout.

□

Note, as we saw in Proposition (4.1.1) that if an initial object preserving functor G has a left adjoint $L \dashv G$, then GL is the identity on the target of G and hence Theorem (4.4.2) holds just as well for initial object preserving functors with left adjoints.

We proceed to show how pushouts on monics are constructed in categories of the form $ST(\mathbf{U})$ to see that it is a relatively simple computation. This shows us that (4.4.2) not only characterises how all opcartesian lifts are structured for adjunctions $L \dashv G$, where GL is an identity but that this characterisation is useful in the context of view-insert propagating. However, we will note that the computation of pushouts is not as simple for general morphisms in categories $ST(\mathbf{U})$.

Note that for preorders we denote $a \vee b$ as the set of least upper bounds of a and b and $a \wedge b$ for the set of greatest lower bounds of a and b in preorders.

Proposition 4.4.3. Let $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ and $A \xleftarrow{f} T \xrightarrow{g} V$ be a span of monomorphisms in $ST(\mathbf{U})$. There exists a pushout of f and g in $ST(\mathbf{U})$ if and only if there is a state $A \sqcup_T B$ and a cospan of monics: $A \xrightarrow{i} A \sqcup_T B \xleftarrow{j} B$ such that $\forall p \in \mathcal{P}$ and $\forall x \in A(p)$ and $y \in B(p)$:

$$i_p(x) = \begin{cases} v_t \in x \vee g_p(t) & \text{If } f_p^{-1}(x) = \{t\}. \\ x' \cong x & \text{Otherwise} \end{cases} \quad \text{and} \quad j_p(y) = \begin{cases} w_t \in f_p(t) \vee y & \text{If } g_p^{-1}(y) = \{t\}. \\ y' \cong y & \text{Otherwise} \end{cases}$$

and $A \sqcup_T \mathcal{B}$ is defined such that:

$$u \in A \sqcup_T B(p) \iff i_p(u) \neq \varnothing \text{ or } j_p(u) \neq \varnothing. \tag{4.10}$$

Proof. ($\Longleftarrow$) Given i and j as in (4.10), for any cocone $A \xrightarrow{a} N \xleftarrow{b} B$ over f and g. There is a unique morphism $k : A \sqcup_T B \to N$ where for each $p \in \mathcal{P}$:

$$\forall v \in A \sqcup_T B(p) : \qquad k_p(v) = \begin{cases} a_p(x) = b_p(y) & \text{If } \exists x \in i_p^{-1}(v) \text{ and } y \in j_p^{-1}(v) \\ a_p(x) & \text{If } j_p^{-1}(u) = \varnothing \\ b_p(y) & \text{If } i_p^{-1}(u) = \varnothing \end{cases} \tag{4.11}$$

($\Longrightarrow$) Now we must show that given a pushout $A \xrightarrow{i} A \sqcup_T B \xleftarrow{j} B$ over f and g; we must show that i and j are monomorphisms and defined as in (4.10).
If we fix any $p \in \mathcal{P}$ and $t \in T(p)$, we find for any upper bound u' of $f_p(t)$ and $g_p(t)$, we can define the cocone $A \xrightarrow{a} U \xleftarrow{b} B$ with $a_p(x) = \begin{cases} u' & \text{If } x = f_p(t) \\ x & \text{Otherwise} \end{cases}$ and $b_p(y) = \begin{cases} u' & \text{If } y = g_p(t) \\ y & \text{Otherwise} \end{cases}$

and since $k_p(f_p(t)) = k_p(g_p(t)) \preceq_p u'$; by the arbitrariness of the upper bound u', we find that $k_p \in f_p(t) \vee g_p(t)$. Then we can see that the definitions of i and j must hold and (4.10) holds because *it can* and pushouts are defined up to isomorphism.

□

For general pushouts, we have to replace the equations above (4.10) with:

$$i_p(x) = \begin{cases} v_t \in \bigvee_{t \in f_p^{-1}(x)} x \vee g_p(t) & \text{If } f_p^{-1}(x) \neq \varnothing. \\ x' \cong x & \text{Otherwise} \end{cases}, \quad j_p(y) = \begin{cases} v_t \in \bigvee_{t \in f_p^{-1}(x)} f_p(t) \vee y & \text{If } g_p^{-1}(y) \neq \varnothing. \\ y' \cong y & \text{Otherwise} \end{cases}$$

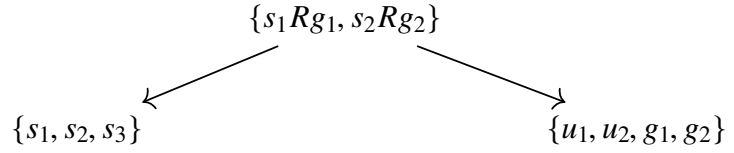
And can imagine in examples, it is much harder to check these morphisms are well-defined.

4.5 Cartesian View-Updating

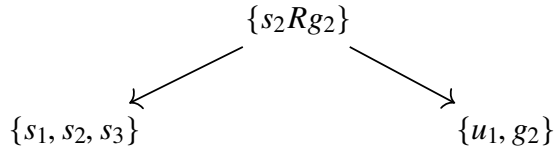
In this section we derive a setting, where initial object preserving view-gets with a left adjoint are fibrations. We begin by motivating the setting by returning to Example (4.3.2).

Example 4.5.1. (Example (4.3.2) Continued)

Incidentally, in our Student-Supervisor example, with $G : \mathcal{S} \to \mathcal{V}$, it is less involved a task to see that G is a fibration as well. For example, consider a source state $R \in \mathcal{S}$ given by:



And a view state U given by the set $\{u_1, g_2\}$. If we then consider the view update $\delta : U \to GR$ fixing u_2 and g_2, we find there is a cartesian lift of δ given by the inclusion $\delta^{\downarrow} : R^{\downarrow} \to R$, where $R^{\downarrow}$ is given by:

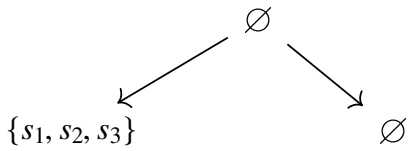


Indeed, the update policy for cartesian lifts is to *delete as little as possible from R to retain synchronisation*, that is $GR^{\downarrow} = U$. As for opcartesian lifts, two pieces of information are paramount to determining a cartesian lift.

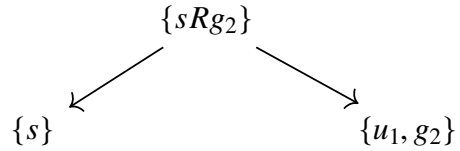
- The maximal amount of information we can retain in R after removing all the data used to derive GR, that is, the source of the cartesian lift of $0_{GR} : \varnothing_{\mathcal{V}} \to GR$, which we denote $0^{\downarrow}_{GR} : R^{\downarrow 0} \to R$
- The minimal source data in $R^{\downarrow}$ we need to keep in order to still have $GR^{\downarrow} = U$, which is data isomorphic to $L(U)$ by definition of a left adjoint.

These are given by:

$R^{\downarrow 0}$:

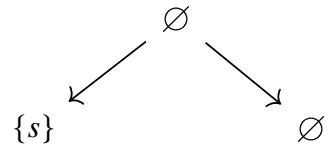


$L(U)$:



We in fact attain $R^{\downarrow}$ by taking a particular pushout of $L(U)$ and $R^{\downarrow 0}$.

The question again is how to stitch, $R^{\downarrow 0}$ and $L(U)$ to form $R^{\downarrow}$; again it turns out a pushout does the job but the head of the span to be:



This turns out to be $R^{\downarrow 0} \times S^{\downarrow}$ but we'll leave the computational generalisation for after it has been proven that the existence of pushout against $R^{\downarrow 0} \xleftarrow{\pi_0} R^{\downarrow 0} \times R^{\downarrow} \xrightarrow{\pi_1} L(U)$ implies $\delta^{\downarrow}$ exists. ■

To prove Theorem (4.5.5), we need to form a number of definitions and elementary lemmas:

Definition 4.5.2. Let $G : \mathcal{A} \to \mathcal{B}$ be a functor. We say that a state $S \in ST(\mathbf{U})$ has a *$\mathcal{B}$-independent form with respect to G* if $0_{GS} : \varnothing_{ST(\mathbf{U})} \to GS$ has a cartesian lift, denoted $0^{\downarrow}_{GS} : S^{\downarrow 0} \to S$. If every state in $\mathcal{A}$ has a *$\mathcal{B}$-independent form* then we say that $\mathcal{A}$ has *$\mathcal{B}$-independent forms* (with respect to G).

If every $S \in \mathcal{A}$ has a $\mathcal{B}$-independent form then we say that $\mathcal{A}$ *has $\mathcal{B}$-independent forms*.

Definition 4.5.3. Let $G : \mathcal{A} \to \mathcal{B}$ be an initial object preserving functor with a left adjoint $L \dashv G$ and $S \in \mathcal{A}$ with a $\mathcal{B}$-independent form. We say S is *pseudo-complemented with respect to G* the following pushout exists:

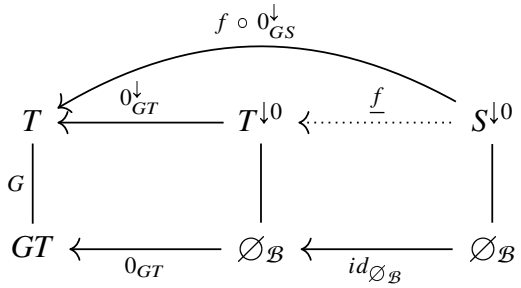
$$\begin{array}{ccc} LGS \times S^{\downarrow 0} & \xrightarrow{\pi_1} & S^{\downarrow 0} \\ \downarrow{\scriptstyle \pi_0} & & \downarrow{\scriptstyle 0^{\downarrow}_{GS}} \\ LGS & \xrightarrow[\epsilon_S]{} & S \end{array}$$

If $\mathcal{A}$ has $\mathcal{B}$ independent forms and all above such pushouts exists, we say that $\mathcal{A}$ *is pseudo-complemented with respect to G*.

We use similar terminology to constant complement updating as in [19] because we show in Chapter 5 that there is an interesting overlap between the strategy derived from the above definition and a more general, rudimentary definition of constant complement updating.

The following lemma is the last prerequisite for proving Theorem (4.5.5), for which we adopt the notation:

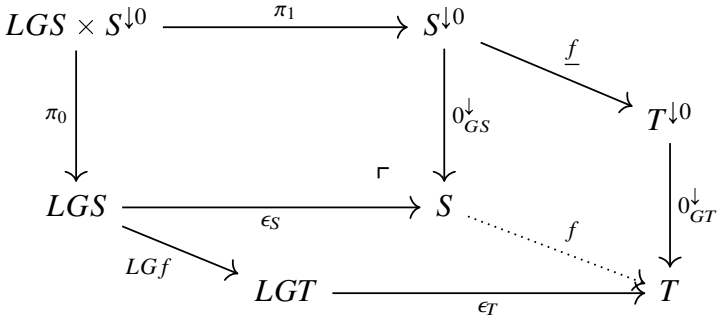
Given any $f : S \to T$ in $\mathcal{A}$, we define $\underline{f} : S^{\downarrow 0} \to T^{\downarrow 0}$ to be the unique morphism given by the universal mapping property of the cartesian lift $0^{\downarrow}_{GT}$ with respect to $f \circ 0^{\downarrow}_{GS}$ and $id_{\varnothing_{\mathcal{B}}}$, that is:



(4.12)

Lemma 4.5.4. Let $G : \mathcal{A} \to \mathcal{B}$ be an initial object preserving functor with a left adjoint $L \dashv G$ and $S \in \mathcal{A}$ be pseudo-complemented with respect to G. For any morphism $f : S \to T$:

If the outer hexagon commutes of the following diagram commutes, so that $\epsilon_T \circ LGf$ and $0^{\downarrow 0}_{GT} \circ \underline{f}$ form a cocone over the product projections π_0 and π_1 then f is the unique morphism solving the pushout property below.



(4.13)

Proof. We need only show that $0^{\downarrow}_{GT} \circ \underline{f} = f \circ 0^{\downarrow}_{GS}$ and $\epsilon_T \circ LGf = f \circ \epsilon_S$ and the pushout property ensures f is unique in doing so.

Indeed, we have that $0^{\downarrow}_{GT} \circ \underline{f} = f \circ 0^{\downarrow}_{GS}$ by (4.12). Then we note that:

$$G(\epsilon_T \circ LGf) = G(\epsilon_T) \circ G(LGf) = id_{GT} \circ Gf = Gf = Gf \circ id_{GS} = Gf \circ G(\epsilon_S) = G(f \circ \epsilon_S)$$

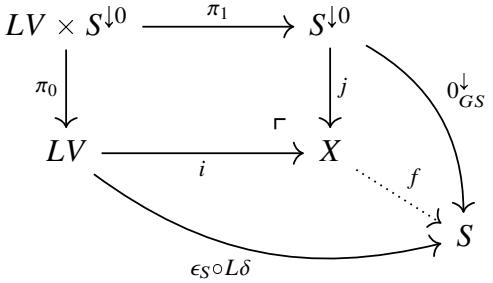
So that by the homset adjunction $Hom_{\mathcal{A}}(LGS, T) \cong Hom(GLGS, GT) = Hom_{\mathcal{B}}(GS, GT)$ implies $\epsilon_T \circ LGf = g = f \circ \epsilon_S$.

□

And now we can prove the main theorem of this chapter over the page as:

Theorem 4.5.5. *Let $G : \mathcal{A} \to \mathcal{B}$ be an initial object preserving functor with a left adjoint $L \dashv G$ such that $\mathcal{A}$ is pseudo-complemented with respect to G.*

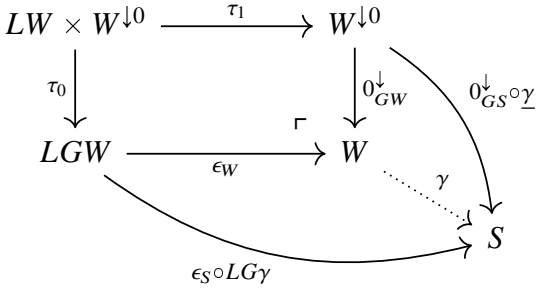
If for every morphism of the form $\delta : V \to GS$ in $\mathcal{B}$, the following diagram commutes, where the inner square is a pushout:



(4.14)

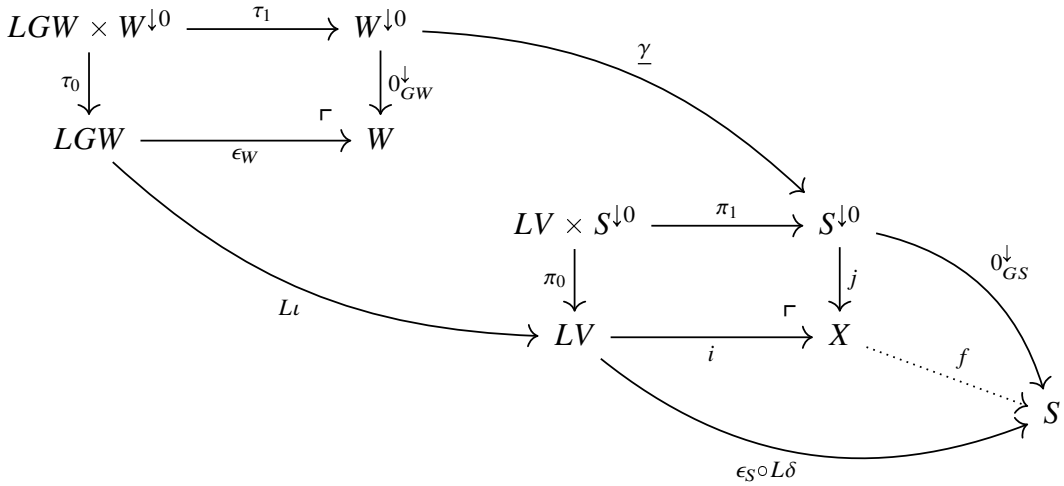
Where $i = \Phi_{(V,S^{\downarrow})}(id_V)$, then: if f is a monomorphism, it is a cartesian lift of δ.

Proof. To see f is a cartesian lift: consider any $\gamma : W \to S$ in $\mathcal{A}$ such that $G\gamma = \iota \circ \delta$ for some $\iota : GW \to V$; we firstly note by Lemma (4.5.4) that γ is the unique morphism making the below diagram commute:



Where τ_0 and τ_1 are product projections.

We proceed to find a unique $n \in Hom_{\mathcal{A}}(W, X)$ satisfying the cartesian lifting property with respect to γ and ι by investigate the following commutative diagram:



(4.15)

Firstly, we note there is a unique morphism $m : LGW \times W^{\downarrow 0} \to LV \times S^{\downarrow 0}$ such that $\pi_0 \circ m = L\iota \circ \tau_0$ and $\pi_1 \circ m = \underline{\gamma} \circ \tau_1$, due to the universal mapping property of the product.

Then by the commutativity of the above diagram with m, we have that $i \circ L\iota \circ \tau_0 = j \circ \underline{\gamma} \circ \tau_1$; hence by the universal mapping property of the upper-leftmost pushout, there is a unique morphism $n : W \to X$

such that $n \circ \epsilon_W = i \circ L\iota$ and $n \circ 0^{\downarrow}_{GW} = j \circ \underline{\gamma}$.

Moreover, we show that $\gamma = f \circ n$ by applying lemma (4.5.4) as follows: we have by the commutativity of (4.15) with m and n that $f \circ n$ satisfies the following two equalities:

- $(f \circ n) \circ 0^{\downarrow}_{GW} = 0^{\downarrow}_{GS} \circ \underline{\gamma}$
- $(f \circ n) \circ \epsilon_W = f \circ i \circ L\iota = \epsilon_S \circ L\delta \circ L\iota = \epsilon_S \circ LG\gamma$

But (4.5.4) implies that γ is unique in satisfying those two equalities, so it must be the case that $\gamma = f \circ n$.

We then see that $Gn = \iota$ as $Gn = G(n \circ \epsilon_W) = G(\bar{\iota} \circ i) = \iota \circ G(i) = \iota$. Thus to conclude that f is a cartesian lift by the existence of n for our arbitrary γ and ι, all that is left to show is that n is unique in satisfying the commutativity conditions with respect to γ and ι. Indeed if we have another morphism $k : W \to S$ just satisfying $f \circ k = \gamma$, because if $f \circ k$ and $Gk = \iota$ we have:

- $k \circ \epsilon_W = i \circ L\iota$ arguing as usual, showing G acting on both sides yields ι then applying the bijection$Hom_{\mathcal{A}}(LGW, X) \cong Hom_{\mathcal{B}}(GW, GX)$ for $L \dashv G$.
- $f \circ k \circ 0^{\downarrow}_{GW} = f \circ j \circ \underline{\gamma} \implies k \circ 0^{\downarrow}_{GW} = j \circ \underline{\gamma}$ by the assumption that f is a monomorphism.

Hence k satisfies the commutativity conditions of the upper-leftmost pushout but n does so uniquely, so $k = n$ and thus f is a cartesian lift.

□

We find that for (5.3) not only can the computation of the pushout part be done explicitly in categories of the form $ST(\mathbf{U})$ but better yet, for a large class of view definitions the outer cocone commutes automatically. We conclude this chapter by showing – beginning with a statement of how products are computed in categories of the form $ST(\mathbf{U})$:

Proposition 4.5.6. Let $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ and $A, B \in ST(\mathbf{U})$. The product $A \times B$ exists in $ST(\mathbf{U})$ if and only if for every $p \in \mathcal{P}$ and any $a \in A(p)$ and $b \in B(p)$ there is an injection:

$$\{(c, d) \in A(p) \times B(p) \text{ in } \mathbf{Pre} \text{ such that } c \wedge d = a \wedge b \text{ in } \mathbf{U}(p)\} \rightarrowtail a \wedge b. \tag{4.16}$$

Proof. ($\implies$) Assume that $A \times B$ exists with projections $\pi_0 : A \times B \to A$ and $\pi_1 : A \times B \to B$ in $ST(\mathbf{U})$. Then for any $p \in \mathcal{P}$ consider any $a \in A(p)$ and $b \in B(p)$ such that there is a lower bound $z \in \mathbf{U}(p)$ of a and b, we can take τ_z representing p, with universal element z in $Rec(\mathbf{U})$.

Then there are unique morphisms $f : \tau_z \to A$ and $g : \tau_z \to B$ so that by the universal mapping property of the product there is unique morphism $k : \tau_z \to A \times B$ such that $\pi_0 \circ k = g$ and $\pi_1 \circ k = f$.

By definition of $ST(\mathbf{U})$ we must have that $z \preceq k_p(z)$ in $\mathbf{U}(p)$ and $k_p(z) \preceq \pi_{0_p}(k_p(z)) = a$ and likewise $k_p(z) \preceq \pi_{1_p}(k_p(z)) = b$, so that $k_p(z)$ is a lower bound of a and b that is greater than z, which by the arbitrariness of z implies that $k_p(z) \in a \wedge b \neq \varnothing$.

However, because we have $\pi_{0_p}(k_p(z)) = a$ and $\pi_{1_p}(k_p(z)) = b$; for π_{0_p} and π_{1_p} to be well defined, we need *enough choices of* $k_p(z)$ that is, (4.16) holds.

($\Longleftarrow$) If (4.16) holds, the following is well defined: there exists an object $K \in ST(\mathbf{U})$ with $\forall p \in \mathcal{P}$: $k \in K(p)$ if and only if: $k \in a \wedge b$ for some $a \in A(p)$ and $b \in B(p)$; and furthermore, $\forall j \in K(p)$: $\pi_0(j) = a$ and $\pi_1(j) = b$ if and only if $j = k$.

Running the first half of the proof in reverse yields that K is a product of A and B.

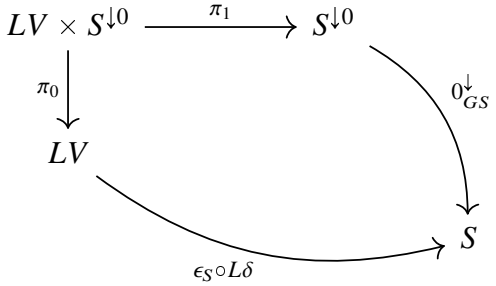
□

Now that we have products in categories of the form $ST(\mathbf{U})$ once we define our rudimentary version of constant complement updating as in [19], we can remove a condition of Theorem (4.5.5).

Definition 4.5.7. We say a functor $G : ST(\mathbf{U}) \to ST(\mathbf{V})$ has a *pointwise view definition* $(\mathbb{V}, \mathbb{G})$ for a functor $\mathbb{V} : \mathcal{Q}^{op} \to \mathcal{P}^{op}$ and natural transformation $\mathbb{G} : \mathbf{U} \circ \mathbb{V} \Rightarrow V$ if:

1. For every $S \in ST(\mathbf{U})$ and $q \in \mathcal{Q}$: the full subcategory $GS(q) \subset V(q)$ has objects $\mathbb{G}_q(S \circ \mathbb{V}(q))$.
2. For every morphism $\delta : S \to T$ in $ST(\mathbf{U})$, the morphism $G\delta$ has components $(G\alpha)_q : GS(q) \to GT(q)$ given by the mapping $\mathbb{G}_q(k) \mapsto \mathbb{G}(\alpha_{\mathbb{V}(q)}(k))$ is well defined over each $k \in S \circ \mathbb{V}(q)$.

Proposition 4.5.8. Let $G : ST(\mathbf{U}) \to ST(\mathbf{V})$ satisfy the conditions of Lemma (4.5.4) with a pointwise view definition $(\mathbb{V}, \mathbb{G})$. We have: for any state $S \in ST(\mathbf{U})$ and view update $\delta : V \to GS$, the following commutes in $ST(\mathbf{U})$:



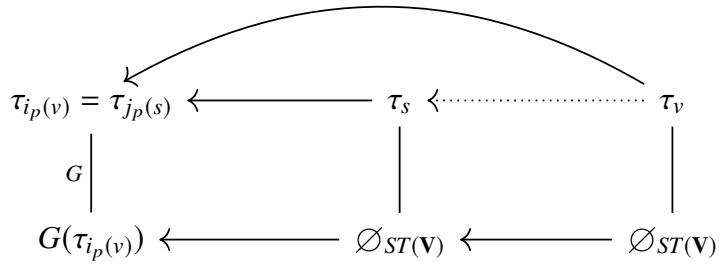
(4.17)

Proof. We prove this by showing that given $\delta : V \to GS$, there is only one morphism $h : LV \times S^{\downarrow 0} \to S$ such that $Gh = \delta$. The structure of the proof is involves showing:

1. $\forall p \in \mathcal{P}$ such that $LV \times S^{\downarrow 0}(p) \neq \varnothing$ we have that $\nexists q \in \mathcal{Q}$ such that: $p = \mathbb{V}(q)$.
2. Then, noting by (4.5.6): $\ell = v \wedge s$ for some $v \in LV(p)$ and $s \in S^{\downarrow 0}(p)$; $G\tau_s = G\tau_v = \varnothing$, whereby we can show $v \preceq s$ recalling τ_v is the record object with universal element v, see the remark under Definition (4.2.6)).
3. Finally, as the hom-set adjunction $Hom_{ST(\mathbf{U})}(LV, S) \cong Hom_{ST(\mathbf{V})}(V, GS)$ implies there is a unique morphism $L\alpha$ such that $GL\alpha = \alpha$, there must only one element $s' \in S(p)$ such that $\pi_{0_p}(\ell) = v \preceq_p s'$ so that $(0^{\downarrow}_{GS} \circ \pi_i)_p(\ell) = 0^{\downarrow}_{GS_p}(s) = s'$.
4. Then we conclude by the arbitrariness of ℓ and p that $|Hom_{ST(\mathbf{U})}(LV \times S^{\downarrow 0}, S)| = 1$.

To prove 1, we note that because G being a right adjoint implies it preserves limits: $G(LV \times S^{\downarrow 0}) = GLV \times GS^{\downarrow 0} = V \times \varnothing_{ST(\mathcal{V})} = \varnothing_{ST(\mathcal{V})}$ (see: Proposition (4.5.6)). Hence if $p \in \mathbb{V}(q)$ for some $q \in \mathcal{Q}$, $\nexists \ell \in LV \times S^{\downarrow 0}$ because then $G(LV \times S^{\downarrow 0})(q) = \mathbb{G}_q(LV \times S^{\downarrow 0}(\mathbb{V}(q))) \neq \varnothing_{ST(\mathcal{V})}$, contradicting that we previously showed that $G(LV \times S^{\downarrow 0}) = \varnothing_{ST(\mathcal{V})}$. Hence $\nexists q \in Q$ such that $p = \mathbb{V}(q)$.

To prove 2, we observe that $\forall s \in S^{\downarrow\mathcal{V}}(p)$: $GS^{\downarrow 0} = \varnothing_{ST(\mathbf{V})} \implies G\tau_s = \varnothing_{ST(\mathbf{V})}$. Furthermore, if $G\tau_v \neq \varnothing$, because G is a pointwise view-definition, that would imply that $\exists q \in Q$ such that $p \geqslant \mathbb{V}(q)$ but that would imply $\varnothing_{ST(\mathbf{V})} \xrightarrow{\not\neq} G\tau_{\mathbf{U}(p \geqslant \mathbb{V}(q))(s)} \xrightarrow{\exists!} G\tau_s = \varnothing_{ST(\mathbf{V})}$, which is impossible as the only endomorphism on an initial object is its identity. Hence, $G\tau_v = \varnothing_{ST(\mathbf{V})}$ and we thus conclude that $v \preceq_p s$ by oberving the following cartesian lifting property:



(4.18)

And the existence of the dotted morphism implies that $v \preceq_p s$ (see: Proposition (4.2.8)).

Statement 3 arises from the fact that if $\exists s' \neq 0^{\downarrow}_{GS_p}(s)$ such that $(L\alpha)_p(v) = s'$ then we would have $v \preceq_p s \preceq_p s'$, which by transitivity implies that there is a distinct $g \in Hom_{ST(\mathbf{U})}(LV, S)$ such that $Gg = \delta$, contradicting the bijection $Hom_{ST(\mathbf{U})}(LV, S) \cong Hom(V, GS)$ given by $L \dashv G$.

Hence we can conclude with statement 4 and see that (4.17) commutes.

□

This concludes Chapter 4. The technical takeaways are Theorem (4.4.2) and Theorem (4.5.5), which are general Category Theory results. When we introduce related work on asymmetric delta lenses in Section (6.1), we will be able to encapsulate these results in a more general theory of synchronisation than the view-update problem. That said, in the context of the view-update problem Proposition (4.4.3) and Proposition (4.5.6) suggest that the main theorems are most applicable to state spaces of monic updates; such as inserts and deletes (possibly with in-place edits) for a monotonic view-definition (that is, monic preserving). However, when we allow in-place edits, cartesian lifts and opcartesian lifts are only defined up to isomorphic in-place edit, which as we saw in Example (4.3.2) and Example (4.4), could yield pathological insert propagations.

We also found that without having appropriate support in our categorical data model for nulls, least change insert propagations became less consistent with respect to how much data they new data they had to add to source states. For more complex examples, this could yield to overall less predictable and less composable insert-propagations. Furthermore, a lack of support for nulls forced least change insert-propagations to add 'default data' to form a valid state to synchronise with the new view state; and this deprecates the real-world meaning of data. In the second half of the next chapter, we will derive a way to add a certain kind of null support to source state spaces, relative to a view-definition.

Chapter. 5

Special Topics

In this section we derive two additional results: Theorem (5.1.4) and Theorem (5.2.4) that further contextualise the applications of Theorem (4.5.5) and (4.4.2) (respectively) to the View-Update Problem. Section (5.1) derives an analogue of a view get having constant updates with respect to another view-get, then shows that constant updates yield cartesian lifts. From these we show Theorem (5.1.4), which shows for psuedo-complemented view-gets with pointwise view-definitions: If those view-get have constant updates, their cartesian lifts are computed using the same pushouts as (4.5.5). This is what is meant by the "overlap" between the two kinds of delete propagations.

In Section (5.2) we derive a new method of defining typed nulls, 'relative to a view-get'. Then Theorem (5.2.4) shows that if we can extend a view-get to act on these nulls in a well defined way, then a left adjoint of the original view-get can be extended to lift view-inserts to these nulls. We see that the issues with the left adjoints for view-gets given without nulls disappear. This allows us to re-evaluate cartesian and opcartesian lifts as per Theorem (4.4.2) and Theorem (4.5.5).

5.1 Constant Complement Updating

We begin by clarifying what is to be meant by "constant-complement":

Definition 5.1.1. For $G : ST(\mathbf{U}) \to ST(\mathbf{V})$, with $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ and $V : \mathcal{Q}^{op} \to \mathbf{Pre}$. We say that $H : ST(\mathbf{U}) \to ST(\mathbf{W})$ with $\mathbf{W} : \mathbf{R}^{op} \to \mathbf{Pre}$ is a *complement of* G if $\langle G, H\rangle : ST(\mathbf{U}) \to ST(\mathbf{V}) \times ST(\mathbf{W})$ is full, faithful and injective on objects.

And analogously to [19] define constant updating as:

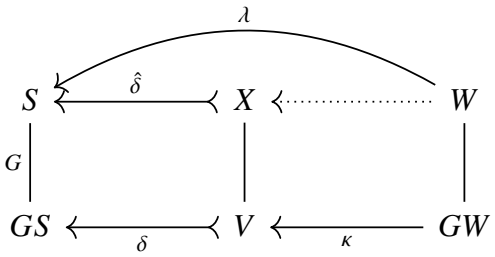
Definition 5.1.2. For $G : ST(\mathbf{U}) \to ST(\mathbf{V})$, with a complement $H : ST(\mathbf{U}) \to ST(\mathbf{W})$, we say a monic $\delta : V \to GS$ in $ST(\mathbf{V})$ has *H-constant updates* if $\exists \hat{\delta}$ in $ST(\mathbf{U})$ such that $G\hat{\delta} = \delta$ and $H\hat{\alpha}$ is an isomorphism.

if every $S \in ST(\mathbf{U})$ has H-constant updates we say that G *has H-constant updates*.

We then have the expected correspondence between constant updates and cartesian lifts, as we can write the following as a close analogy of [19, Theorem 1.]:

Proposition 5.1.3. Let $G : ST(\mathbf{U}) \to ST(\mathbf{V})$ be a functor with complement $H : ST(\mathbf{U}) \to ST(\mathbf{W})$. If $\delta : V \rightarrowtail GS$ is a monic with an H-constant update $\hat{\delta}$, then $\hat{\delta}$ is a cartesian lift of δ.

Proof. We proof $\hat{\delta} : X \to S$ is a cartesian lfit by finding the following, unique dotted morphism satisfying the unique cartesian lifting property as in:



(5.1)

To find this morphism, firstly note that there is a morphism in $ST(\mathbf{V}) \times ST(\mathbf{W})$ given by:

$$\langle G, H\rangle(\kappa, (H\hat{\delta})^{-1} \circ H\lambda) : (GW, HW) \to (GX, HX) \tag{5.2}$$

Where $H\hat{\delta}$ is invertible because $\hat{\delta}$ is an H-constant update with respect to δ. Then, because $\langle G, H\rangle$ is full, there exists a morphism $k : W \to X$ with $Gk = \kappa$ and $Hk = (H\hat{\delta})^{-1} \circ H\lambda$. I claim that k is the necessary dotted morphism, that is, $\hat{\delta} \circ k = \lambda$. Note, that suffices because we can easily verify that k satisfies the necessary uniqueness property because $\hat{\delta}$ is monic – following the same reasoning as was used in Theorem (3.4.6)).

Indeed $\hat{\delta} \circ k = \lambda$ follows from the following calculation, now using the faithfullness of $\langle G, H\rangle$:

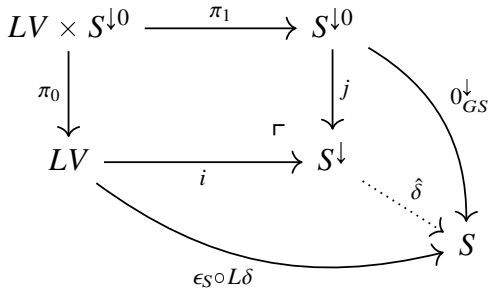
$$\begin{aligned}\langle G, H\rangle(\hat{\delta} \circ k) &= (\delta \circ \kappa, H\hat{\delta} \circ (H\hat{\delta})^{-1} \circ H\lambda) \\ &= (\delta \circ \kappa, H\lambda) = (G\lambda, H\lambda) = \langle G, H\rangle(\lambda)\end{aligned}$$

□

We are particularly interested in G with complement H, when they are given by pointwise view definitions. Also, in reference to [19], we relate cartesian lifts of monics with constant-updates, and find:

Theorem 5.1.4. *Let $G : ST(\boldsymbol{U}) \to ST(\boldsymbol{X})$ and $H : ST(\boldsymbol{U}) \to ST(\boldsymbol{W})$ be functors with pointwise view-definitions such that G has H-constant updates and is initial object preserving with $L \dashv G$.*

Then, for any monic $\delta : V \rightarrowtail GS$ in $ST(\boldsymbol{U})$, the pushout exists:



(5.3)

Where $i = \Phi_{(V,S^{\downarrow})}(id_V)$ and $\hat{\delta}$ is a cartesian lift of δ.

Note this pushout is equivalent to (5.3) *in the statement of Theorem* (4.5.5)*.*

Proof. For $\delta : V \rightarrowtail GS$ in $ST(\mathbf{V})$, we note that $ST(\mathbf{U})$ being pseudo-complemented with respect to G implies that $LGS \times S^{\downarrow 0}$ exists in $ST(\mathbf{U})$ and for every $p \in \mathcal{P}$ and $k \in LV(p)$: $(L\delta)_p : LV(p) \rightarrowtail LGS(p)$ provides an upper bound of k in $LGS(p)$.

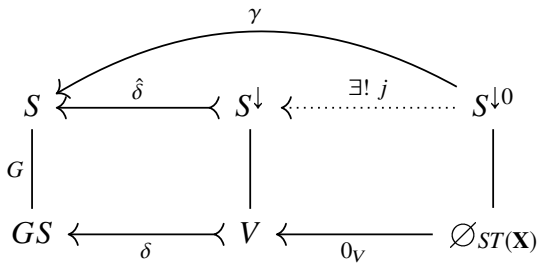
Then, arguing as in Lemma (4.5.8), using the pointwise view definition of G, one can show there is a product $LV \times S^{\downarrow 0}$ such that $\forall \ell \in LV \times S^{\downarrow 0}(p)$, $\ell \in LV(p)$ [1].

[1] Note in reference to Proposition (4.5.6) that (4.16) is satisfied by the fact that there is an injection $LV \rightarrowtail LGS$

Continuing to repeat the argument of Lemma (4.5.8), we can conclude the following is a cocone of π_0 and π_1:

$$\begin{array}{ccc} LV \times S^{\downarrow 0} & \xrightarrow{\pi_1} & S^{\downarrow 0} \\ {\scriptstyle \pi_0}\downarrow & & \downarrow{\scriptstyle j} \\ LV & \xrightarrow[i]{} & S^{\downarrow} \end{array} \tag{5.4}$$

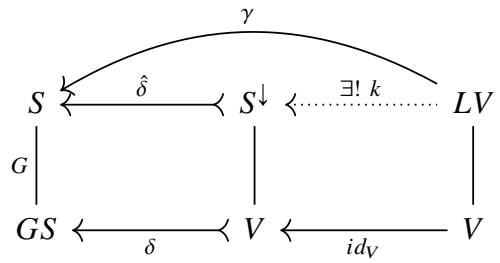
Where, j is given as follows by the cartesian lifting property:



(5.5)

Furthermore, we find by the universal mapping property that j is the cartesian lift of $0_{S^{\downarrow}} = 0_V$, so j is an H-constant update of 0_V.

It remains to show that (5.4) satisfies the pushout property. We begin by defining a morphism k by the following universal mapping property:



(5.6)

The show (5.4) is a pushout by showing the following cocone is a pushout in $ST(\mathbf{X}) \times ST(\mathbf{W})$:

$$\begin{array}{ccc} (\varnothing_{ST(\mathbf{X})}, H(LV \times S^{\downarrow 0})) & \xrightarrow{(G\pi_1, H\pi_1)} & (\varnothing_{ST(\mathbf{X})}, HS^{\downarrow 0}) \\ {\scriptstyle (G\pi_0, H\pi_0)}\downarrow & & \downarrow{\scriptstyle (0_V, Hj)} \\ (V, HLV) & \xrightarrow[(id_V, Hk)]{} & (V, HS^{\downarrow}) \end{array} \tag{5.7}$$

and conclude (5.4) is a pushout using the fullness of $\langle G,\ H\rangle$.
We begin by showing the following is a pushout:

$$\begin{array}{ccc} H(LV \times S^{\downarrow 0}) & \xrightarrow{\pi_1} & HS^{\downarrow 0} \\ {\scriptstyle \pi_0}\downarrow & & \downarrow{\scriptstyle j} \\ HLV & \xrightarrow[i]{} & HS^{\downarrow} \end{array} \tag{5.8}$$

Indeed, we fix any $p \in \mathcal{P}$ and note that j is H-constant, that Hj is an isomorphism. Then, because the product projection π_{0_p} fixes elements in $H(LV \times S^{\downarrow 0})(p)$ – so maps elements to isomorphic targets;

(5.8) has mappings matching the sufficient condition for pushouts in $ST(\mathbf{W})$ given by Proposition (4.4.3). Then we observe that:

$$\begin{array}{ccc} \varnothing_{ST(\mathbf{X})} & \xrightarrow{\pi_1} & \varnothing_{ST(\mathbf{X})} \\ {\scriptstyle \pi_0}\downarrow & & \downarrow{\scriptstyle j} \\ V & \xrightarrow[id_V]{} & V \end{array} \tag{5.9}$$

Is obviously a pushout so that (5.7) is a pushout and thus by the fullness of $\langle G,\ H\rangle$, (5.4) is a pushout. □

Thus we have shown that initial object preserving Fibrations computed by Theorem (4.5.5) include constant-complement updating policies as in [19], when the view-get has a pointwise view definition. We proceed to the return to the question of resolving a lack of support for nulls that was encountered in the opfibrational solution to Example (4.3.2).

5.2 A More General Conception Of Nulls

We saw that in Example (4.3.2) that for any state $V \in ST(\mathbf{U}_0)$ where $V(Students)$ contains at least one graduate student implies that $L(V)(Spvs)$ contains exactly one 'default supervisor' and further, it could be any supervisor. While this isn't problematic theoretically, it isn't an ideal policy for update propagations in practice. On one hand the real world meaning of the data is lost as in one can no longer be sure if a supervisor is *really* a supervisor or a view-insert was performed. We then saw by Theorem (4.4.2) that this pathological behaviour then permeates to all opcartesian lifts.

Now if we are able to add support for nulls in Example (4.3.2) we can't use a partial-map definition of nulls. That is, we can't make the multi-funcional relation $\mathbf{U}_0(Spvn) \subset \mathbf{U}_0(Spvs) \times \mathbf{U}_0(Stud)$ a partial multi-functional relation or we violate the constraint that every grad student has a supervisor. Hence we need to 'formally add' typed null supervisors to $\mathbf{U}_0(Spvs)$ and supervision relations for them in $\mathbf{U}_0(Spvn)$.

Example 5.2.1. (Example (4.3.2) Continued)

Our task is to find a functor $\mathbf{U}_{0_\perp} : \mathcal{P}^{op} \to \mathbf{Pre}$ such that $\mathbf{U}_0 \subset \mathbf{U}_{0_\perp}$ and $\mathbf{U}_{0_\perp}$ has desirable *Null* supervisors and supervision relations. Further, we need to find some $\mathcal{S} \subset \mathcal{S}_\perp \subset ST(\mathbf{U}_{0_\perp})$ and $G_\perp : \mathcal{S}_\perp \to ST(\mathbf{U}_1)$ such that $G_\perp$ restricts to G and is an opfibration with some $L_\perp \dashv G_\perp$.

We begin by reconsidering $V \in ST(\mathbf{U}_1)$ corresponding to the set $\{u_1, u_2, g_1, g_2\}$ and seek to define "Null supervision relations" $\perp_{g_1} Rg_1$ and $\perp_{g_2} Rg_1 \in \mathbf{U}_{0_\perp}(Spvn)$, where $L_\perp(V)(Spvn) = \{\perp_{g_1} Rg_1, \perp_{g_2} Rg_1\}$. Moreover, for $i = 1, 2$: $\perp_{g_i} := \mathbf{U}_{0_\perp}(Spvn \geqslant Spvs)(\perp_{g_i} Rg_i)$ in $\mathbf{U}_{0_\perp}$ needs to somehow represent that *they are supervisors but we aren't sure who they are*.

The principal manner in which say, $\perp_{g_1}$ should be distinguished from *known* supervisors $s \in \mathbf{U}_0(Spvs)$ that g_1 can have assigned is that $\perp_{g_1} \preceq_{Spvs} s$ but that $s \not\preceq_{Spvs} \perp_{g_1}$ in $\mathbf{U}_{0_\perp}(Spvs)$. This is because on one hand we need $\perp_{g_1} \preceq_{Spvs} s$ for $0_{L_\perp(V)}$ to possibly be an opcartesian lift. On the other hand, if $s \preceq_{Spvs} \perp_{g_1}$ so that $s \cong \perp_{g_1}$ in $\mathbf{U}0_\perp$ then categorically speaking s and $\perp_{g_1}$ can't be distinguished by any basic categorical constructions.

By the same token we have to ask whether $\bot_{g_1} \cong \bot_{g_2}$? That is, whether the null-supervisor for g_1 should be (categorically) distinguishable from the null supervisor of g_2? An example of why we may want to consider that $\bot_{g_1} \ncong \bot_{g_2}$ is: perhaps our source state space may be further constrained to say g_1 is a graduate student in biology and must only be in states with supervision relations involving a supervisor in "the faculty of science". Likewise, g_2 may be fine arts graduate student with different restrictions on their eligible supervisors.

One change to $\mathbf{U}_{0_\bot}$ we need to be cognisant of is that objects $L_\bot(V)$ and more specifically opcartesian lifts $0_{L_\bot(V)}$ of 0_V need to satisfy the universal mapping property with respect to new elements in $\mathbf{U}_{0_\bot}(Spvs)$ of the form $\bot_{(-)}R(-)$. We can do this by requiring that $j \preceq_{Spvn} \bot_{g_i} R g_i$ if and only if $j = \bot_h Rh$ for some $h \preceq_{Stud} g_i$.

As before $\mathcal{S}_\bot \subset ST(\mathbf{U}_{0_\bot})$ such that $S \in \mathcal{S}_\bot$ if and only if for every graduate student $x \in S(Stud)$ there exists a unique $yRx \in S(Spvn)$ noting this time y can be of the form $\bot_x$.

■

We now proceed to define $\mathbf{U}_{0_\bot}$ and $L_\bot \dashv G_\bot$ more generally.

Definition 5.2.2. Given a functor $G : ST(\mathbf{U}) \to ST(\mathbf{U})$ with a pointwise view definition $(\mathbb{V}, \mathbb{G})$, we define the $\mathbb{G}$ *nullable form* of $\mathbf{U}$ as the functor $\mathbf{U}_\bot : \mathcal{P}^{op} \to \mathbf{Pre}$ with $\mathbf{U} \subset \mathbf{U}_\bot$ and $\forall p \in \mathcal{P}\backslash\mathbb{V}(\mathcal{Q})$:

For every $k \in \mathbf{U}(p)$ there is a unique $k_\bot \in \mathbf{U}_\bot(p)\backslash\mathbf{U}(p)$ such that:

$$\begin{aligned} k_\bot \preceq_p m \text{ in } \mathbf{U}_\bot(p) &\iff k \preceq_p m \text{ in } \mathbf{U}(p) \\ n \preceq_p k_\bot \text{ in } \mathbf{U}_\bot(p) &\iff n = \ell_\bot \text{ for } \ell \preceq k \text{ in } \mathbf{U}(p). \end{aligned}$$

And for every $p \geqslant q$ in $\mathcal{P}$: $\mathbf{U}_\bot(p \geqslant q)(k_\bot) = \mathbf{U}(p \geqslant q)(k)_\bot$.

We operate over $\mathcal{P}\backslash\mathbb{V}(\mathcal{Q})$ because we don't want to add null symbols to our source state space, for which we don't know how to derive view data out of.

In reference to our last example, we note that finding the $\mathbf{U}_0$ nullable form of $ST(\mathbf{U})$ is straight forward for supervisors and for supervision relations $sRg \in \mathbf{U}_0(Spvn)$ we have that $(sRg)_\bot = s_\bot Rg$ as we have we previously written. After introducing some more notation, we discern certain kinds of view-gets G can be redefined on $\mathbb{G}$ nullable forms.

For any category $ST(\mathbf{U}_\bot)$ where $\mathbf{U}_\bot$ is the nullable form of some $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$ for any $\mathbf{X} \in ST(\mathbf{U}_\bot)$ we denote:

$$\mathbf{X}_\vDash := \{S \in ST(\mathbf{U}) \mid \forall p \in \mathcal{P},\ k \in S(p) : k \in \mathbf{X}(p) \text{ or } k_\bot \in \mathbf{X}(p)\}$$

Then any morphism $\alpha : \mathbf{X} \to \mathbf{Y}$ in $ST(\mathbf{U}_\bot)$:

$$\alpha_\vDash := \left\{ \delta : S \to T \text{ in } ST(\mathbf{U}) \;\middle|\; S \in \mathbf{X}_\vDash,\ T \in \mathbf{Y}_\vDash \text{ and } \forall p \in \mathcal{P},\ k \in S(p) : \delta_p(k)_\bot = \begin{cases} \alpha_p(k) & \text{if } k \in \mathbf{X}(p) \\ \alpha_p(k_\bot) & \text{otherwise.} \end{cases} \right\}$$

Definition 5.2.3. Let $G : ST(\mathbf{U}) \to ST(\mathbf{V})$ be a functor. We say G has a $\boldsymbol{U}_\perp$*-nullable form* if there is a $G_\perp : ST(\mathbf{U}_\perp) \to ST(\mathbf{V})$ such that the following action on objects and morphisms is well defined:

- $\forall \mathbf{X} \in ST(\mathbf{U}_\perp)$: $G_\perp \mathbf{X} = GS$ for any $S \in \mathbf{X}_\models$.
- $\forall \alpha : \mathbf{X} \to \mathbf{Y}$ in $ST(\mathbf{U}_\perp)$: $G_\perp \alpha = G\delta$ for any $\delta \in \alpha_\models$.

We will discuss at the conclusion of this section an example of a view-get that does not have a nullable form but we find that G of Example (4.3.2) does and we can redefine $L \dashv G$ to an appropriate $L_\perp \dashv G_\perp$ and hence find a much better policy for propagating view-update as:

Theorem 5.2.4. *If $G : ST(\boldsymbol{U}) \to ST(\boldsymbol{V})$ is an initial object preserving functor with a left invertible, left adjoint $L \dashv G$ and a $\boldsymbol{U}$-nullable form $G_\perp : ST(\boldsymbol{U}_\perp) \to ST(\boldsymbol{V})$ then $G_\perp$ has a unique left invertible, left adjoint $L_\perp \dashv G_\perp$.*

Proof. Because $G_\perp$ is an initial object preserving functor by Proposition (4.1.1) we need only show that $\forall Z \in ST(\mathbf{V})$, there is an opcartesian lift of the morphism $0_Z : G_\perp(\varnothing_{ST(\mathbf{U}_\perp)}) \to Z$. We define $L_\perp Z \dashv G_\perp$ as follows:

$$\forall p \in \mathcal{P} : \ ob(L_\perp Z(p)) = \{x_\perp : x \in LZ(p)\}$$

And find that $0_{L_\perp Z} : \varnothing_{ST(\mathbf{U}_\perp)} \to L_\perp Z$ is an opcartesian lift of 0_Z and hence $L_\perp \dashv G_\perp$.

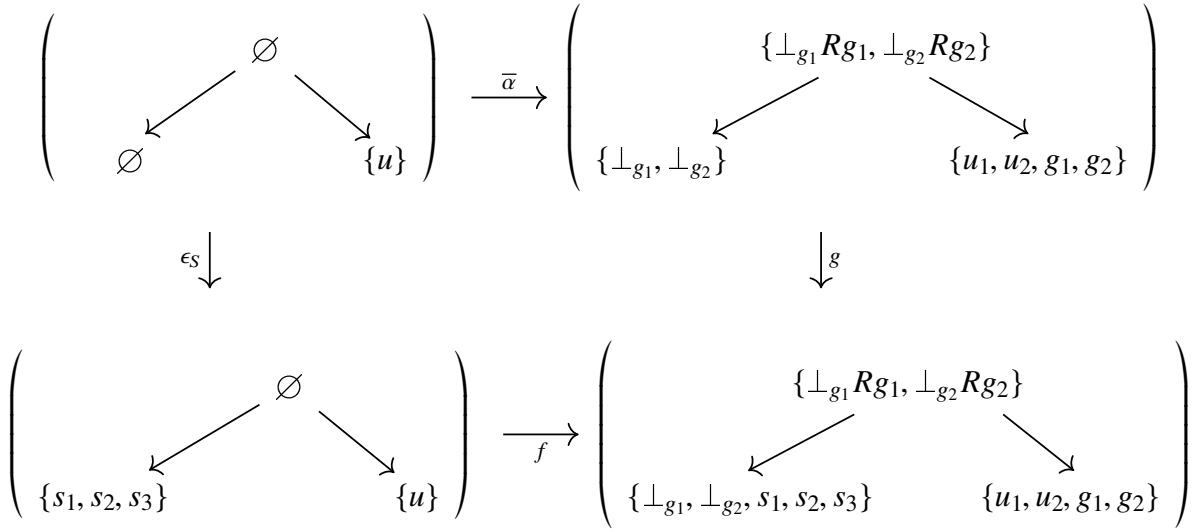
We can verify that $L_\perp$ is unique in doing so as left adjoints are defined up to isomorphism so that if there were a distinct left adjoint $M \dashv G_\perp$ then given $\varphi : L_\perp \cong M$, there then exists a $Z' \in ST(\mathbf{V})$, $p \in \mathcal{P}$ and $k \in L_\perp Z(p)$ such that $\varphi_p(k) \neq k$ but by definition $\varphi_p(k) \cong k \implies \varphi_p(k)_\perp = k_\perp$. Contradicting that there is a distinct $M \dashv G_\perp$.

□

Example 5.2.5. (Continued) Forming $\mathbf{U}_{0_\perp}$ with respect to $\mathbf{U}$, $\mathcal{S}_\perp$ with respect to $\mathcal{S}$ and $G_\perp : \mathcal{S}_\perp \to ST(\mathbf{U}_1)$ with respect to G, we find that the following pushout square exists:

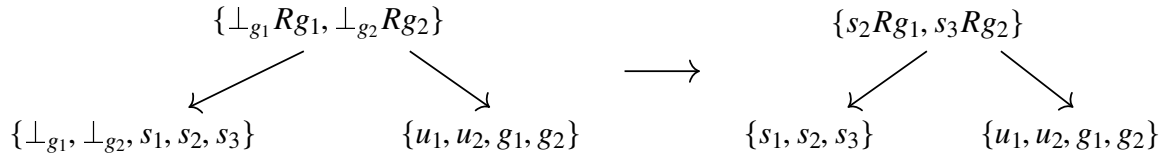
$$\begin{array}{ccc} L_\perp(G_\perp S) & \xrightarrow{\overline{\alpha}} & L_\perp V \\ {\scriptstyle \epsilon_S}\downarrow & \ulcorner & \downarrow \\ S & \longrightarrow & S \sqcup_{L_\perp(G_\perp S)} L_\perp V \end{array} \tag{5.10}$$

Corresponding to:



And because we can see that $j = \Phi_{(V,S \sqcup_{L_\perp (G_\perp S)} L_\perp V)}$ we can conclude by Theorem (4.4.2) that f is an opcartesian lift of α.

Furthermore, we understand $\perp_{g_1}, \perp_{g_2}$ as nulls because if we decide that say, g_1 is supervised by s_2 and g_2 is supervised by s_3 there is a morphism:



That sends $\perp_{g_1} \mapsto s_2$ and $\perp_{g_2} \mapsto s_3$ to reflect this. Note also, this is a non trivial manner of requiring non-monic updates in categorical state spaces of models on sketches.

■

We should note that not all (pointwise) view-gets G have a $\mathbf{U}_\perp$ nullable form. For example consider a source state space with one Employee-Entity with a Salary Attribute and a view consisting of a Cost-Entity constrained to having a single instance with a cost-integer attribute. If we define a view get that maps a source state to the sum of its Employee Salaries then we find that the mappings of Definition (5.2.3) may not be well defined because any source state $\mathbf{X}$ with null-employees may have states in $\mathbf{X}_\mathsf{F}$ with many different combinations of Salaries. Thus different representatives from $\mathbf{X}_\mathsf{F}$ may map to different view-states under the view-get. Hence investigating other approaches to forming nulls, particularly dependent on the view-update itself is a topic of future interest.

Chapter. 6

Related Work

This chapter discusses related work both in the sense of work that is a natural extension of this thesis, and in the sense of work concerning important, close-by topics not treated in this thesis. In Section (6.1), we begin by encapsulating our main results in the more general theory of Asymmetric Delta Lenses, an item of Category Theory recently applied to studying bidirectional transformations in computer science. This encapsulation will allow us to rephrase Chapter 1's outline of issues encountered by Category Theoretic descriptions of least change propagations. This encapsulation will also give us a better sense of why we approached the view-update problem, Categorically, without reference to relational algebra as is expected in standard database theory texts such as Date in [7]. Then, in Section (6.2) we will discuss related work in the study of other categorical data models that offer refinements to how we approached forming and using categorical state spaces of the form $ST(\mathbf{U})$. Particular emphasis will be placed on the Simplicial Data Model of Spivak in [31].

6.1 Asymmetric Delta Lenses

A *Lens* is a construct in computer science that links two state spaces, for example states of: databases, software specifications, programs, etc. A Lens is equipped with operations that define:

- When a pair of states, (one) from each state space are *synchronised*.
- Given two synchronised states A and B, how a state change from A to another state A' should be responded to a state change from B to some other state B' such that A' and B' are synchronised (and visa versa).

There are many kinds of lenses, appropriate for different sorts of state spaces and synchronisation. An important class of lenses are known as *Asymmetric Lenses*, where the term 'asymmetric' often refers to one state space consisting of states derived from the other state space.

We begin by recalling the following construction from Category Theory: given a functor $G : \mathcal{S} \to \mathcal{V}$, the comma category $(G, id_{\mathcal{V}})$ has as objects pairs $(S, \alpha : GS \to V)$ for some $S \in \mathcal{S}$ and $V \in \mathcal{V}$ and morphisms $(S, \alpha : GS \to V) \to (S', \beta : GS' \to V')$ are given by a pair of morphisms $(f : S \to S', g : V \to V')$ such that $g \circ \alpha = \beta \circ Gf$.

We denote for a category C, the discrete category with objects given by $ob(C)$ as $|C|$.

Definition 6.1.1. An *(Asymmetric) d-Lens* from $\mathcal{S}$ to $\mathcal{V}$ is given by a pair (G, P), where $G : \mathcal{S} \to \mathcal{V}$ is a functor and $P : |(G, id_V)| \to |Arr(\mathcal{S})|$ a function on objects satisfying, for view updates: $\alpha : GS \to V$ and $\beta : GS' \to V'$ in $\mathcal{V}$:

i) The domain of $P(S, \alpha)$ is S.

ii) $P(S, id_{GS}) = id_S$.

iii) $G(P(S, \alpha)) = \alpha$.

iv) If S' is the domain of $P(S, \alpha)$ then $P(S, \beta \circ \alpha) = P(S', \beta) \circ P(S, \alpha)$ in $\mathcal{S}$.

For an asymmetric delta lens (G, P), G is often called the "get" and P is often called the "put". Indeed, given a state in $\mathcal{S}$, G *gets* the derived state of S: GS in $\mathcal{V}$ and likewise for state changes (morphisms). Moreover, a state S in $\mathcal{S}$ and a state U in $\mathcal{V}$ are synchronised if $GS = U$. So, given a state change from U (synchronised with S) to a new state V, thence given by a morphism $\alpha : GS \to V$; P can put the state change α into $\mathcal{S}$ to retrieve a state space in response to α. The four conditions on an asymmetric delta lens can be understood as follows:

i) Sometimes called "put-id" says that the put of a view state change must be state change from the synchronised source state in question.

ii) Often called "put-get" says that putting a state change that makes "no change", should not change the synchronised source state in question.

iii) Often called "get-put" says that a put state change should be synchronised with the view state change in question.

iv) Often called "put-put" says that iterated state-changes in $\mathcal{V}$ should result in the same put, whether we put the changes one by one or all at once (The functoriality of G ensures this for gets).

Conditions i)-iii) may seem fairly innocent and expected for most contexts of synchronisation; sometimes they are only true up to isomorphism. However, the expectation of put-put in applications is controversial. Related work concerning this controversy will be discussed at the end of this section.

Recalling that a (op)fibration is a functor with all (op)cartesian lifts, the encapsulation of our work into the study of lenses is as follows: It is shown in [20] that if $G : \mathcal{S} \to \mathcal{V}$ is a (split) opfibration, it corresponds to a *d-Lens* (G, P) where P maps each $(S, \alpha : GS \to V)$ to $(S, \alpha^{\uparrow})$, the opcartesian lift given by the cleavage of the specified opfibration. In elementary terms $i) - iii)$ hold by definition of opcartesian lifts and $iv)$ is precisely the definition of an opfibration having a *splitting*. Note, the composite of (op)cartesian lifts is always an (op)cartesian lift of the composites, so put-put always holds up to isomorphism for opfibrations in general.

Johnson and Rosebrugh prove in [20] the above fact by showing opfibrations are equivalent to another kind of lens called a *c-Lens*, which is any algebra for a monad with respect to a functor $G : \mathcal{S} \to \mathcal{V}$ in the slice category $\mathbf{Cat}/\mathcal{V}$ (see: [23, p.16])[1]. In the same paper, Johnson and Rosebrugh also derive a dual result for fibrations and *c′-lenses*. Note however, the converse statement – that every *d-Lens* is an opfibration is not true.

The theory of asymmetric d-lenses both provides a more general theory of view-update style synchronisation and has provided a useful framework for constructing lenses out of other lenses. An application of this is automating the construction of lenses between state spaces, which allows data to be shared between state spaces without entrusting a party to have access to all the data on both sides. A result demonstrating this principle, found in [16, p.3] as:

Proposition 6.1.2. Let $\mathcal{S} \xrightarrow{G} \mathcal{V} \xleftarrow{H} \mathcal{W}$ be a cospan of functors and $P : |(G, id_{\mathcal{V}})| \to |Arr(\mathcal{S})|$ a function of objects making (G, P) an asymmetric d-lens. Then given a pullback square:

$$\begin{array}{ccc} \mathcal{T} & \xrightarrow{G'} & \mathcal{W} \\ {\scriptstyle H'}\downarrow & \lrcorner & \downarrow{\scriptstyle H} \\ \mathcal{S} & \xrightarrow[G]{} & \mathcal{V} \end{array}$$

[1]There are also results in [23] showing that d-lenses are pseudo-algebras for a pseudo-monad on $\mathbf{Cat}/\mathcal{V}$.

There is a d-lens (G', P') defined with put $P' : |(G', id_{\mathcal{W}})| \to |Arr(\mathcal{W})|$ given by the mapping $((S, W), \alpha : G'(S, W) \to W') \longmapsto (P(S, H\alpha), \alpha)$.

This demonstrates the aforementioned application when pulling back over a cospan of d-lenses to get a square of d-lenses. This scenario occurs when synchronisation between state spaces $\mathcal{S}$ and $\mathcal{W}$ is to be achieved when their owners are only willing to share data in $\mathcal{V}$ as witnessed by G and H. Synchronisation can be done as follows: when $\mathcal{S}$ incurs as state change, we put the change into $\mathcal{T}$ via the put for H' and move the $\mathcal{W}$ state to G' of the target of our put. State changes in $\mathcal{W}$ are handled analogously using the put on G' and get H'.

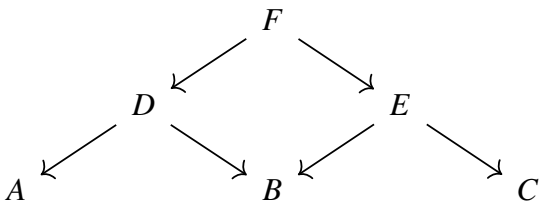
We should note that Johnson and Rosebrugh showed in [16] that the category of (equivalence classes of) spans of asymmetric d-lenses is equivalent to the category of symmetric lenses, which are a widely used class of lenses between state spaces for the case that neither state space can't be (functorially) derived from the other (such as $\mathcal{S}$ and $\mathcal{W}$). Indeed, symmetric lenses come with a relation on states that tells us which states are synchronised and a two 'put' operations, one for each state space to the other.

An issue with this arrangement is that in applications $\mathcal{T}$ ends up having all the data of $\mathcal{S}$ and $\mathcal{W}$, which may render the third party $\mathcal{T}$ as undermining the security concerns of each state space's owners. However, a party $\mathcal{T}$ may exist regardless if all the data of $\mathcal{S}$ and $\mathcal{W}$ are on the same cloud service for example. Johnson and Rosebrugh have been exploring ways to acquire more appropriate symmetric lenses from cospans of asymmetric lenses in [22] and [13]).

Hence a topic of future interest is to see whether the study of asymmetric delta lenses with left adjoints as we have used them, aid in these tasks; particularly, if there is an appropriate generalisation of Theorem (4.4.2) and (4.5.5) for asymmetric d-lenses.

To conclude this section, we restate two interesting topics of Category Theory research pertaining to put-put. However, we can understand these issues in terms of problems encountered when propagating simulated edits.

For the first topic, we recall that a simulated edit is given by a span of updates and a proposed least change propagation consisted of a span of a cartesian lift followed by an opcartesian lift. However, this means view simulated edits and propagated simulated edits alike aren't directly composable, so there isn't an immediate analogue of put-put for simulated edits. However, given two simulated edits from A to B and from B to C that is, a diagram $A \leftarrowtail D \rightarrowtail B \leftarrowtail E \rightarrowtail C$; there is an analogue of composition given by taking the (isomorphism class) of pullbacks of the form:



(6.1)

By taking our composite span to be the (equivalence class) of spans $A \leftarrowtail F \rightarrowtail C$. Indeed in [15], Johnson and Rosebrugh derive conditions for which a pair of put-put laws for arrows pointing in the opposite direction imply a *mixed put-put law* for propagations of composites of the form of (6.1). The ethos of why such a result is useful is that put-put isn't anywhere near as controversial for state spaces consisting just of monic inserts (without view-edits) or state spaces with just monic deletions (without

view-edits) as morphisms in the opposite direction to inserts.

Furthermore, Johnson and Rosebrugh show in [21] that for a pair of lenses, one a c lens and one a c' lens (both algebras for a monad), a condition for when a distributive law (in the sense of Beck in [4]) holds between them, yielding propagations of the form of (6.1)[2].

The second topic concerns cases where propagating view simulated edits as a view-delete propagation followed by a view-insert propagation may deviate from what we would expect a least change propagation to do to edits. An example of this that Diskin observed in [10] is as follows: consider a view of a database that contains the employee information for a person, say Mary. If we perform a deletion of Mary followed by an re-insertion of Mary in the view, there may be two choices of how to interpret this and respond:

a) Mary left and rejoined the company and the least change propagation should yield no change in the source state space.

b) Mary left and another Mary joined the company in her place, which should result in a new employee, with different source-state space details to be added in place of the original Mary's.

For relational databases, this is typically solved by a method of generating unique keys for unique users (never overwriting previously used keys). However, the general issue above remains for less trivial examples, including those outside the databasing context of the lens theory. As a result Diskin defines a new sort of Category Theoretic Lens, where the get is a "lax functor".

6.2 Schema Structure & Abstracting the View Update Problem

To describe the purpose of this section, we recall the basic organisation of specifying categorical state spaces of the form $ST(\mathbf{U})$. We begin by taking $\mathcal{P}$, a poset of inclusions with respect to all the combinations of entities that have valid states with at least one instance in every such entity. We define $\mathbf{U} : \mathcal{P}^{op} \to \mathbf{Pre}$, where for each $p \in \mathcal{P}$, $\mathbf{U}(p)$ is a preorder on all possible examples of tuples defined over that combination of entities. $ST(\mathbf{U})$ then defines something akin to relational states on the universe of instances given by $\mathbf{U}$.

The purpose of this section is to one one hand describe related work that utilises a preordering on relational instances, that doesn't exist in state spaces of the form $Mod(\mathbb{E}, \mathbf{Set})$ or Spivak's Simplicial Data Model of [31]. On the other hand, we will describe many shortcomings of categorical state spaces of the form $ST(\mathbf{U})$. Shortcomings particularly for attaining results about least change view-updating based on Schema design. We will describe how state spaces of the form $Mod(\mathbb{E}, \mathbf{Pre})$ may prove to be a better categorical data model to work with in the future; and will derive how schemas are defined in the simplicial data model to demonstrate an interesting alternative.

The notion of an *information (pre)order* on relational instances and by extension on relational states; whereby the ordering is on how much missing or how uncertain the information in instances and states there is. The idea goes back to at least Hegner's paper [12], which featured an early generalisation of constant complement updating. In Category Theory, the concept is implemented in the "second approach to partiality" of Johnson and Rosebrugh in [25] to add nulls to the sketch data model; and the Categorical State Spaces and u-lenses of Diskin in [9] to better define lenses on states

[2] In [21] a pseudo-distributive law and analogous propagation results are given for d-lenses.

with uncertain information. We also find applications of the concept outside of formal specifications and view-updating. Diskin's approach is based on Libkin's "Database Domains" with "informational preorders", which are applied to the problem of how certain query outputs can be on uncertain data. Lastly, Wisjen in [36] defines "information orders" with respect to "database homomorphisms" to fix invalid database states by iteratively making them less certain and then more certain until they are valid once again.

Hence we can see, having a categorical data model that effectively ascertains the information orderings on both instances and by extension states is very useful. However, encoding the schema of a database as a partial order – as has been done in this thesis – obscures how entities are constructed using relational algebra. Hence, we seem to lose the generality of studying state spaces as ordinary categories as is done in the theory of delta lenses and also lose the ability to study least change propagations in terms of schema design as is done by Johnson and Rosebrugh in [18].

One manner of defining categorical data models that allows us to combine both sophisticated schemas and view definitions and information orderings is to instead look at state spaces of the form $Mod(\mathbb{E}, \mathbf{Pre})$ rather than $Mod(\mathbb{E}, \mathbf{Set})$. Indeed, Johnson, Rosebrugh and Wood study properties of Models of sketches in categories other than **Set** in [24]. However, there are alternatives to using sketches and by extension models on sketches. We conclude by showing how Spivak defines schemas using simplicial sets and – space permitting – note that he defines actual database states by taking certain kinds of sheaves on the poset of subsimplicial-sets with respect to the schema in question.

To begin, we distinguish a function $\pi : D \to \mathbf{DT}$ called the *type specification* of the state space. D is a set of attribute domains called the *domain bundle* and **DT** is a set of attribute headers. Given the type specification π as above and a header $H \in \mathbf{DT}$, we understand $\pi^{-1}(H)$ as the *domain of attribute* H and an element $x \in \pi^{-1}(H)$ as an *object of type* H.

Example 6.2.1. Consider a $GradStudent$ entity given by a table with three attributes with headers $StudentID$, $Email$, $Degree \in \mathbf{DT}$. The attribute headers have attribute domains **8-Dig**, **eString** and **Faculty** respectively denoting 8 digit Student ID's, valid email address strings and some code strings denoting various university faculties. So we can take D to be the disjoint union: **8-Dig** $\sqcup$ **eString** $\sqcup$ **FacCode** $\sqcup$ ... and π matching the domains to headers as desired.
■.

Definition 6.2.2. A *simple schema* of type specification $\pi : D \to \mathbf{DT}$ is a function $\sigma : C \to \mathbf{DT}$.

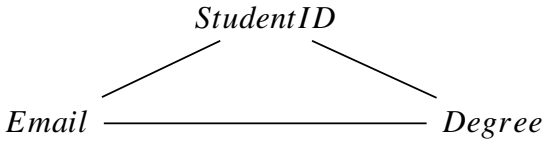
The simple schema of the $GradStudent$ table can be given by $\sigma : \{StudentID, Email, Faculty\} \to$ **DT** given in the obvious way.

Definition 6.2.3. Given two simple schemas (C, σ) and (C', σ') with respect to $\pi : D \to \mathbf{DT}$, we define a morphism of simple schemas as a function $f : C \to C'$ such that $\sigma' \circ f = \sigma$.

We denote $\mathcal{S}^{\pi}$ as the category with objects as simple schemas and the above morphisms.

In this setting we use the term simple schema, synonymously to how we have previously used the term 'relation' to refer to the attribute headers and domains that entity-states are comprised of. We proceed to show how *Schemas on a type specification* π, that is, collections of somehow inter-related simple schemas can be regarded as a modified notion of a simplicial set.

To begin, consider our familiar attribute headers such as *StudentID*, *Email*, *Degree*, labelling vertices by attribute headers; a simple schema can be understood as a standard simplex on its attributes headers. The simple schema of the students table in Example (6.2.1) is then geometrically realised by the whole triangle:

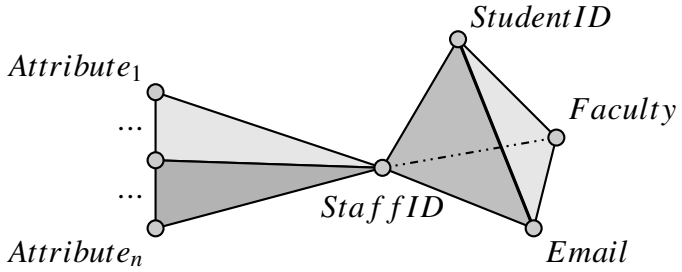


If *Students* only had two attributes it would be realised as a line and if it had four attributes, a triangular pyramid and so on. More generally, as Spivak states in [31, Remark 2.2.7]:

Proposition 6.2.4. If we let Δ denote the category of finite ordered sets then $\mathcal{S}_\pi \cong \Delta/\mathbf{DT}$, the slice category on a total order on the set **DT** given by any type specification $\pi : U \to \mathbf{DT}$.

Of course the above isomorphism only depends on **DT**, altough the data of π is used in full when we consider states on a particular *schema on* π. We next define *category of schemas on* π to be the functor category $[S_\pi^{op}, \mathbf{Set}]$. Hence referring to Proposition (6.2.4), we find that the only difference between schemas on π and simplicial sets, that is, functors $\Delta^{op} \to \mathbf{Set}$ is that each *simplex* in a schema on π has labels in **DT**, and simplices can only be joined at simplices with respect to a functor in $[S_\pi^{op}, \mathbf{Set}]$ where the labels match.

Example 6.2.5. Consider we have a *Supervisor* entity, a table with $n+1$ attributes with headers $StaffID$, $Attribute_1$, $Attribute_2$, ..., $Attribute_n$ such that there is a functional relation $GradStudent \to Supervisor$, or equivalently the $GradStudent$ table has a column $StaffID$; we can then represent the schema denoting the relation $GradStudent \to Supervisor$ a simplicial set with geometric realisation:



Where pictorially, we can only see 3 of the n dimensions of the *Supervisor* simple schema.
■

Thus we see that Categorical State Spaces can incorporate formal specifications of database schemas drawing from Category Theoretic discourse in Algebraic Topology as well as Categorical Universal Algebra in the case of models on sketches. Moreover, there is a growing body of general Category Theory being used in the theory of Lenses used to study synchronisation problems (including the view-update problem). Furthermore, as witnessed by this thesis, there is an interesting interplay between these two Category Theoretic approaches. That is, propagating updates for synchronised state spaces may have informative structures at the general Category Theoretic level but the computational concerns of finding these propagations and reflecting on how schematic designs have to be made are informed at the more specific Category Theoretic level.

Chapter. 7

Conclusion

The motivating setting of this thesis has been the intersection between the tasks of giving a Category Theoretic description of least change solutions to view update problems and giving Category Theoretic descriptions of the relational data model. Least change solutions to view-update problems are desirable because they best keep the side effects of view-updating contained and have more predictable accompanying source-insertions and deletions.

A Category Theoretic account of these things is desirable for the following reasons:

- Category Theory treats updates as *first class citizens*; only defining the structure of least change propagations up to isomorphism. This gives database designers an effective language to discern: to what degree their policy of propagating view-updates respects composition and associativity. Designers are also informed whether propagations may unintentionally modify source-data as a result of choosing an inappropriate isomorphic copy of a propagation.
- Categorical data models can provide formal specifications and visual (diagrammatic) data modelling tools for more general data models than the relational data model (See: the references given in Section (2.3)).
- Relevant general results in Category Theory can be ported into other applications in Model Driven Engineering, particularly those in the field of bidirectional transformations.

An exemplary approach to these tasks is given by the Sketch Data Model. The Sketch Data Model is certainly advantageous in its ability to formally specify at the same categorical level: ERA-diagrammatic information, constraints on entity-relations (e.g. functional, one to one relations, commutativity etc.) and constraints on the make-up of tables via standard relational algebraic operations. This capability arises from the use of Categorical Universal Algebra to produce a schema encoding all standard queries that can be performed on a database schema (see: Section 3.3). This adds deep structure to both the basics of query languages and of view definitions formed using relational algebraic operations..

However, a caveat to that deep structure is that the categorical state spaces of the Sketch Data Model only allow morphisms that are *homomorphisms* of databases, so to speak. In particular, in-place edits of data can't be performed (see: example (3.1.3)). This restriction has been argued to be appropriate in various applied contexts, such as in Libkin's account of querying uncertain data (e.g. [27]) and restoring consistency to database states violating the constraints of their schemas (e.g. Wisjen in [36]). However, judging the appropriateness of this restriction isn't obvious for the view-update problem. In example (2.2.1), we saw 'canonically' propagating in-place edits was not necessarily equivalent to propagating simulated edits. Further yet, in example (4.3.2) we showed least-change view-updating may be untenable without in-place edits.

In the Sketch Data Model, least change propagations of view-updates came in two varieties: insert and delete propagabiltiy; and opcartesian and cartesian lifts with respect to a functorial view-get. The former variety more clearly represents least change propagations for view-inserts, view-deletes and view simulated edits. There are also a host of sufficient and necessary conditions for the existence of these propagations with respect to the design of the source schema and view definition.

These conditions are informative for database designers; several of these conditions are derived by Johnson and Rosebrugh in [18, Section 6]. However, in Example (3.3.4) we showed that insert and delete propagability may be untenable when views are constructed out of non-monotonic queries. This is not the case for opcartesian and cartesian view-updating, which thus has a larger scope of applicability.

Keeping an open mind to in-place editing motivated the main results of this thesis: Theorem (4.4.2) and Theorem (4.5.5). These theorems presented classes of opcartesian and cartesian lifts, respectively, that could be explicitly computed for view-update propagations if the respective view-get has a left adjoint that is also its right inverse.

Theorem (4.4.2) is a complete characterisation of opcartesian lifts for view-gets with a left adjoint that is also its right inverse. Although, applying this result to categorical state spaces only consisting of monic updates was far more tractable. Furthermore, the continuations of Example (4.3.2) suggested allowing in-place edits could be problematic for the following reasons:

- Least change propagations of view-inserts may cause varying amounts of 'default data' to be added with respect to new tuples needed to derive the updated view. These default values may deprecate the real-world interpretation of the data.
- State spaces of monics are desirable because we saw that pushouts – a key categorical instrument for constructing our propagations – seem to be much simpler to construct (let alone check if they exist) when the pushout is over a span of monics (see Proposition (4.4.3)).

That said, in section (5.2), we derived cases where some of these issues could be resolved if a certain support for nulls existed in the categorical data model in question. The typed nulls we used were derived relative to a view definition, which is distinct from more standard approaches with relational algebra in mind, for example: Johnson and Kasangian's approach in [14], Zaniolo's in [37] and Libkin's in [26], to name a few. Moreover, in example (5.2.5) we saw our definition admitted non-trivial, non-monic updates used to replace nulls with defined values. Hence, we found a context in the study of the view-update problem where it is worth to not restricting to monic morphisms only.

Theorem (4.5.5) is a sufficient condition for initial object preserving functors with a left adjoint to have (monic) cartesian lifts. We saw in section (5.1) that this result had an interesting overlap with constant complement updating, first defined categorically by Johnson and Rosebrugh in [19] in reference to the original concept formulated by Brancilhon and Spyratos in [1].

The abstract categorical account of opcartesian lifts and cartesian lifts on one hand can be reinterpreted in more sophisticated categorical data models such as the Sketch Data Model, the Simplicial Data Model of [31] or cases of the Indexed Categorical Data Model of [29, Example 3]. On the other hand, Section (6.1) showed how Theorem (4.4.2) and (4.5.5) can be rephrased in the language of asymmetric delta lenses (and by extension symmetric delta lenses). Hence these results are applicable to the study of bidirectional transformations, which itself has many applications in Model Driven Engineering.

References

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

[2] Michael Barr and Charles Wells. *Toposes, triples and theories*. Springer-Verlag, 1985.

[3] Michael Barr and Charles Wells. *Category theory for computer science*. Prentice Hall, 1996.

[4] Jon Beck. *Distributive Laws*, pages 119–140. Springer Verlag, 1969.

[5] John L. Bell. Types, sets, and categories. *Handbook Of The History Of Logic, Sets and Extensions in the Twentieth Century*, 6:633–687, 2012.

[6] Edgar F Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13:377–387, 1970.

[7] C. Date. *View Updating and Relational Theory*. Theory in practice. O'Reilly Media, Incorporated, 2013.

[8] C. J. Date. *An introduction to database systems*. Addison-Wesley, eighth edition, 2004.

[9] Zinovy Diskin. Asymmetric delta-lenses with uncertainty: Towards a formal framework for flexible bx. (GSDLAB–TR 2016-03-01), 03/2016 2016.

[10] Zinovy Diskin. *Compositionality of Update Propagation: Lax PutPut*, pages 74–89. 2017.

[11] Peter J. Freyd and Scedrov Andrej. *Categories, allegories*. North-Holland, 2006.

[12] Stephen J. Hegner. An order-based theory of updates for closed database views. *Annals of Mathematics and Artificial Intelligence*, 40(1):63–125, Jan 2004.

[13] Michael Johnson. Universal updates for symmetric lenses. *CEUR Proceedings*, 1827:39–53, 2017.

[14] Michael Johnson and Stefano Kasangian. A relational model of incomplete data without nulls. In *Proceedings of the Sixteenth Symposium on Computing: The Australasian Theory - Volume 109*, CATS '10, pages 89–94, Darlinghurst, Australia, Australia, 2010. Australian Computer Society, Inc.

[15] Michael Johnson and Robert Rosebrugh. *Lens put-put laws: monotonic and mixed*, volume 49, pages 1–13.

[16] Michael Johnson and Robert Rosebrugh. Spans of delta lenses. *CEUR Proceedings*, 1396:1–15.

[17] Michael Johnson and Robert Rosebrugh. View updatability based on the models of a formal specification. *Lecture Notes in Computer Science FME 2001: Formal Methods for Increasing Software Productivity*, pages 534–549, 2001.

[18] Michael Johnson and Robert Rosebrugh. Fibrations and universal view updatability. *Theor. Comput. Sci.*, 388:109–129, 2007.

[19] Michael Johnson and Robert Rosebrugh. Constant complements, reversibility and universal view updates. In *Proceedings of the 12th International Conference on Algebraic Methodology and Software Technology*, AMAST 2008, pages 238–252, Berlin, Heidelberg, 2008. Springer-Verlag.

[20] Michael Johnson and Robert Rosebrugh. Delta lenses and opfibrations. *International Workshop on Bidirectional Transformations*, 2:1–18, 2013.

[21] Michael Johnson and Robert Rosebrugh. Distributing commas, and the monad of anchored spans. *CEUR Proceedings*, 1396:31–42, 2015.

[22] Michael Johnson and Robert Rosebrugh. Symmetric delta lenses and spans of asymmetric delta lenses. *The Journal of Object Technology*, 16:1–32, 2017.

[23] Michael Johnson, Robert Rosebrugh, and R. J. Wood. Lenses, fibrations and universal translations. *Mathematical Structures in Computer Science*, 22:25–42, 2012.

[24] Michael Johnson and Robert D. Rosebrugh. Entity-relationship-attribute designs and sketches. volume 10, pages 94–112, 2002.

[25] Michael Johnson and Robert D. Rosebrugh. Three approaches to partiality in the sketch data model. *"Electr. Notes Theor. Comput. Sci"*, 78:82–99, 2003.

[26] Leonid Libkin. A relational algebra for complex objects based on partial information. In *Proceedings of the 3rd Symposium on Mathematical Fundamentals of Database and Knowledge Base Systems*, MFDBS 91, pages 29–43, New York, NY, USA, 1991. Springer-Verlag New York, Inc.

[27] Leonid Libkin. Certain answers as objects and knowledge. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*, 2014.

[28] A. M. Pitts. *Categorical logic*. University of Cambridge, Computer Laboratory, 1995.

[29] Robert Rosebrugh and R J Wood. Relational databases and indexed categories. *Proceedings of the International Category Theory Meeting 1991, CMS Conference Proceedings, 13*, pages 391–407, 1992.

[30] P. Schultz, D. I. Spivak, C. Vasilakopoulou, and R. Wisnesky. Algebraic Databases. http://adsabs.harvard.edu/abs/2016arXiv160203501S, February 2016.

[31] D. I. Spivak. Simplicial Databases. http://adsabs.harvard.edu/abs/2009arXiv0904.2012S, April 2009.

[32] Ross Tate. Opfibrations. *Accessed 21/7/17. Lecture Notes For CS 6118 (Fall 2012) - Types and Semantics - Cornell University*.

[33] Wikipedia, the free encyclopedia. E-R Diagram for Relational Mail Order Database. http://www.augustana.ab.ca/~hackw/csc430/exhibit/mailOrder.html. [Online; accessed October, 2017].

[34] Wikipedia, the free encyclopedia. Inquisite 9 ER Diagram. https://oregondas.allegiancetech.com/help/Inquisite_ER_Diagram.htm. [Online; accessed October, 2017].

[35] Wikipedia, the free encyclopedia. File:Relational db terms.png. https://en.wikipedia.org/wiki/File:Relational_db_terms.png, 2007. [Online; accessed October, 2017].

[36] J. Wisjen. Making more out of an inconsistent database. pages 291–305, 2004.

[37] Carlo Zaniolo. Database relations with null values. In *Proceedings of the 1st ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, PODS '82, pages 27–33, New York, NY, USA, 1982. ACM.

[38] Bartosz Zielinski, Pawet Maslanka, and Scibor Sobieski. Allegories for database modeling. *Model and Data Engineering Lecture Notes in Computer Science*, pages 278–289, 2013.

[39] Bartosz Zielinski, Pawet Maslanka, and Scibor Sobieski. Modalities for an allegorical conceptual data model. *Axioms*, 3:260–279, 2014.