On The Relational Foundations Of Functorial Data Migration

David I. Spivak *
Massachusetts Institute of Technology
dspivak@math.mit.edu

Ryan Wisnesky
Harvard University
ryan@cs.harvard.edu

October 13, 2013

Abstract

We study the data transformation capabilities associated with schemas that are presented by labeled directed multi-graphs and path equivalence constraints. Unlike most approaches which treat graph-based schemas as abbreviations for relational schemas, we treat graph-based schemas as categories. A morphism M between schemas S and T, which can be generated from a visual mapping between graphs, induces three adjoint data migration functors, $\Sigma_M : S \to T$, $\Pi_M : S \to T$, and $\Delta_M : T \to S$. We present an algebraic query language FQL based on these functors, investigate its metatheory, and present SQL query generation algorithms.

1 Introduction

In this paper we use the tools of category theory to develop a model theory and operator algebra for database instances that satisfy path equality constraints. In doing so we pick up a long line of work aptly summarized by Melnik in his thesis [20]. He describes three ways for achieving an "executable model theory" suitable for large-scale and generic information-integration efforts [21]; we quote these here:

1. One way is to consider schemas, instances, and mappings as syntactic objects represented in a common meta-theory, e.g, as graphs. This approach has been pursued in almost all prior work on generic model management. In essence, the operators are specified by means of graph transformations. As long as the graph transformations do not exploit any knowledge of what the graphs actually represent, the operators can be considered truly generic. Unfortunately, there are very few useful operations that can be defined in such an agnostic fashion. Largely, they are limited to Subgraph, Copy, and the set operations on graphs.

2. A second way to achieve generic applicability is by using state-based semantics. In this approach, the properties of the operators are characterized in terms of instances of schemas that are taken as input and produced as output. Under the assumption that schemas possess well-defined sets of instances, all key operators can be characterized in a truly generic fashion. Such characterization is applicable to very complex kinds of schemas and mappings that are used in real applications, including XML Schemas, XQuery, and SQL. Although state-based characterization does not provide a detailed implementation blueprint, it is sufficiently specific so that the effect of the operators can be worked out for concrete languages.

3. A third way for addressing generic applicability is an axiomatic one, e.g., using a category-theoretic approach. The idea of the approach is to define the operators using axioms that are expressed in terms of the operators to be defined. Associativity of compose or commutativity of merge are examples of such axioms. This approach seems to be the most challenging, both in terms of determining a useful set of axioms and implementing the operators in such a way that the axioms hold when the operators are applied to concrete languages.

Whereas Melnik proceeds to develop approach 2, and to implement a prototype called Rondo [22], in this paper we develop a new approach that combines 1 and 3. In particular, our schemas are *categories*, and our operators are those of the functorial data model [26]. Because they are categories, our schemas are strongly related to graphs yet significantly more expressive (approach 1); they also form a category themselves, the category of categories, which is well understood categorically (approach 3). We believe that our combined approach enjoys the benefits of approaches 1 and 3 while nullifying their disadvantages. The goal of this paper is to develop enough of a connection between functorial data migration and traditional relational database theory that this claim can be evaluated.

*Spivak acknowledges support from ONR grants N000141010841 and N000141310260.

1.1 Background and Motivation

In the functorial data model [26], database schemas are finitely presented categories [4]: directed labeled multigraphs with path-equivalence constraints. Database instances are functors from categories/schemas to the category of sets. By targeting the category of sets, database instances can be stored as relational tables. Database morphisms are natural transformations (morphisms of functors) from database instances to database instances. The database instances and morphisms on a schema S constitute a category, denoted S–**Inst**. A morphism M between schemas S and T, which can be generated from a visual correspondence between graphs, induces three adjoint data migration functors, $\Sigma_M \colon S\text{–}\mathbf{Inst} \to T\text{–}\mathbf{Inst}$, $\Pi_M \colon S\text{–}\mathbf{Inst} \to T\text{–}\mathbf{Inst}$, and $\Delta_M \colon T\text{–}\mathbf{Inst} \to S\text{–}\mathbf{Inst}$. At a high-level, this functorial data model provides an alternative category-theoretic foundation from which to study problems in information management. The mathematical foundations of this data model are developed in [26] using the language of category theory, but few specific connections are made to relational database theory.

Although labeled mutli-graphs with path equivalence constraints are a very common notation for schemas [6], there has been very little work to treat such schemas categorically [11]. Instead, most schema transformation frameworks treat graphs as relational schemas. For example, in Clio [14], users draw lines connecting related elements between two schemas-as-graphs and Clio generates a relational query that implements the user's intended data transformation. Behind the scenes, the user's correspondence is translated into a formula in the relational language of second-order tuple generating dependencies, from which a query is generated [10]. As another example, in the Rondo system [22], users are presented with an ad-hoc collection of operators over schema-as-graphs that they then script together to implement a data transformation. Although graphs and path-equivalences are used as inputs for both Clio and Rondo, both Clio and Rondo immediately translate from graphs and path-equivalences into a relational schema before proceeding further.

1.2 Contributions and Outline

In this paper we make the following contributions:

1. As described above, the functorial data model [26] is not quite appropriate for many practical information management tasks. Intuitively, this is because every instance in the pure functorial data model behaves like a relational database instance where all values are IDs, no values are constants, and equality is actually isomorphism. So, we extend the functorial data model with "attributes" to capture meaningful concrete data. The practical result is that our schemas become special kinds of entity-relationship (ER) diagrams, and our instances can be represented as relational tables that conform to such diagrams.

2. We define a simple algebraic query language FQL where every query denotes a data migration functor in this new extended sense. We show that FQL is closed under composition, and how every query in FQL can be written as three graph correspondences roughly corresponding to projection, join, and union. Determining whether three arbitrary graph correspondences are FQL queries is semi-decidable.

3. We provide a translation of FQL into SQL, by which we mean the union of two languages: 1) the SPCU relational algebra of selection, projection, cartesian product, and union, under its typical set semantics, and 2) a globally unique ID generator that constructs $N+1$-ary tables from N-ary tables by adding a globally unique ID to each row. This allows us to easily generate SQL programs that implement FQL. An immediate corollary is that materializing result instances of FQL queries has polynomial time data complexity.

4. We show that every relational conjunctive (SPC) query *under bag semantics* is expressible in FQL, and how to extend FQL to capture every relational conjunctive query under set semantics.

5. We show that FQL is a schema transformation framework in the sense of Alagic and Berstein's categorical model theory [2].

6. We have implemented FQL in a prototype visual schema mapping and SQL generation tool in the spirit of Clio and Rondo, available at wisnesky.net/fql.html. The tool includes over twenty built-in examples, and the webpage provides a link to a hands-on tutorial for database practitioners.

FQL has much interesting structure that, for reasons of space, we cannot discuss. For example, FQL schemas and instances both form cartesian closed categories, meaning they can be programmed using a λ-calculus. We give the definition of "full" FQL in the appendix. We conclude the introduction by reviewing some basic concepts in category theory.

1.3 Review of Category Theory

Category theory is an axiomatically specified algebra of abstract functions suitable for formalizing mathematics. In contrast to traditional set theory, where structures

are defined by what they *are*, in category theory structures are defined by how they *interact*. Compared to set theory, category theory is notable for its high-level of abstraction and focus on compositionality. Since its inception in the 1940s, category theory has been applied in many disciplines, including information management, where categorical methods inspired the design of functional query languages such as the nested relational algebra [28]. The adjacent fields of mathematical logic and programming language theory also employ categorical techniques [18].

A *category* $\mathbf{C}$ consists of a class of *objects* $\mathrm{Ob}(\mathbf{C})$ and a class of *morphisms* or *arrows* $\mathrm{Hom}_{\mathbf{C}}$ between objects. Each morphism m has a source object S and a target object T, which we write as $m\colon S \to T$ or $S \xrightarrow{M} T$. Every object X has an identity morphism $id_X\colon X \to X$. Two morphisms $f\colon B \to C$ and $g\colon A \to B$ may be *composed*, written $f \circ g\colon A \to C$ or $g; f\colon A \to C$. Composition is associative and id is its unit:

$$f \circ id = d \qquad id \circ f = f \qquad f \circ (g \circ h) = (f \circ g) \circ g$$

A morphism $f\colon X \to Y$ is an isomorphism when there exists another morphism $g\colon Y \to X$ such that

$$f \circ g = id \qquad g \circ f = id$$

Two objects are *isomorphic* when there exists an an isomorphism between them. A *functor* $F\colon \mathbf{C} \to \mathbf{D}$ between two categories $\mathbf{C}$ and $\mathbf{D}$ is a mapping of objects of $\mathbf{C}$ to objects of $\mathbf{D}$ and morphisms of $\mathbf{C}$ to morphisms of $\mathbf{D}$ that preserves identities and composition:

$$F(f\colon X \to Y)\colon F(X) \to F(Y)$$

$$F(id) = id \qquad F(f \circ g) = F(f) \circ F(g)$$

We will write the identity functor for a category C as $1_C : C \to C$. A *natural transformation* $\alpha : F \Rightarrow G$ between two functors $F\colon \mathbf{C} \to \mathbf{D}$ and $G\colon \mathbf{C} \to \mathbf{D}$ is a family of morphisms $\alpha_X\colon F(X) \to G(X)$ in D, one for each $X \in \mathrm{Ob}(\mathbf{C})$, such that for every $f\colon X \to Y$ in $\mathbf{C}$

$$\alpha_Y \circ F(f) = G(f) \circ \alpha_X$$

A natural transformation α is an isomorphism when for every object X in $\mathbf{C}$, the morphism α_X is an isomorphism in $\mathbf{D}$. Two functors F and G are (naturally) isomorphic if there exists an isomorphism from F to G. An *adjunction* between categories C and D consists of

- A functor $F : D \to C$ called the left adjoint
- A functor $G : C \to D$ called the right adjoint
- A natural isomorphism $\phi : Hom_C(F-,-) \Rightarrow Hom_D(-,G-)$
- A natural transformation $\epsilon : FG \Rightarrow 1_C$
- A natural transformation $\eta : 1_D \Rightarrow GF$

Almost all of the category-theoretic content of this paper can be understood using the above concepts. The proofs, however, are much more sophisticated.

2 Categorical Data

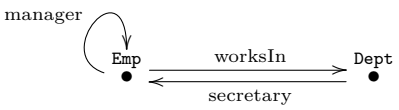
In this section we define the original signatures and instances of [26], as well as "typed signatures" and "typed instances", which are our extension of [26] to attributes. The basic idea is that signatures are stylized ER diagrams that denote categories, and our database instances can be represented as instances of such ER diagrams, and vice versa (up to natural isomorphism; see Proposition C.1.6).

2.1 Signatures

The functorial data model [26] uses directed labeled multi-graphs and path equalities for signatures. A *path* p is defined inductively as:

$$p ::= node \mid p.edge$$

A *signature* S is a finite presentation of a category. That is, a signature S is a triple $S := (N, E, C)$ where N is a finite set of nodes, E is a finite set of labeled directed edges , and C a finite set of path equivalences. E.g,



(1)

$$\texttt{Emp.manager.worksIn} = \texttt{Emp.worksIn}$$
$$\texttt{Dept.secretary.worksIn} = \texttt{Dept}$$

Here we see a signature S with two vertices and three arrows, and two path equivalence statements. This information generates a category $[\![S]\!]$: the free category on the graph, modulo the equivalence relation induced by the path equivalences. The category $[\![S]\!]$ is the *schema* for S, and database instances over $[\![S]\!]$ are functors $[\![S]\!] \to \mathbf{Set}$. Every path $p\colon X \to Y$ in a signature S denotes a morphism $[\![p]\!]\colon [\![X]\!] \to [\![Y]\!]$ in $[\![S]\!]$. Given two paths p_1, p_2 in a signature S, we say that they are *equivalent*, written $p_1 \cong p_2$ if $[\![p_1]\!]$ and $[\![p_2]\!]$ are the same morphism in $[\![S]\!]$. Two signatures S and T are *isomorphic*, written $S \cong T$, if they denote isomorphic schema, i.e. if the categories they generate are isomorphic.

2.2 Cyclic Signatures

If a signature contains a loop, it may or may not denote a category with infinitely many morphisms. Hence, some constructions over signatures may not computable. Testing if two paths in a signature are equivalent is known as the *word problem* for categories. The word problem can be semi-decided using the "completion without failure" extension [3] of the Knuth-Bendix algorithm. This algorithm first attempts to construct a strongly normalizing re-write system based on the path equalities; if it succeeds, it yields a linear time decision procedure for the word problem [15]. If a signature denotes a finite category, the Carmody-Walters algorithm [7] will compute its denotation. The algorithm computes *left Kan extensions* and can be used for many other purposes in computational category theory [11]. In fact, every Σ functor arises as a left Kan extension, and vice versa.

2.3 Instances

Let S be a signature. A $[\![S]\!]$-instance is a functor from $[\![S]\!]$ to the category of sets. We will represent instances as relational tables using the following binary format:

- To each node N corresponds an "identity" or "entity" table named N, a reflexive table with tuples of the form (x, x). We can specify this using first-order logic:

$$\forall xy.N(x,y) \Rightarrow x = y. \tag{2}$$

  The entries in these tables are called *ID*s or *keys*, and for the purposes of this paper we require them to be globally unique. We call this the *globally unique key assumption*.

- To each edge $e\colon N_1 \to N_2$ corresponds a "link" table e between identity tables N_1 and N_2. The axioms below merely say that every edge $e\colon N_1 \to N_2$ designates a total function $N_1 \to N_2$:

$$\begin{array}{l} \forall xy.\ e(x,y) \Rightarrow N_1(x,x) \\ \forall xy.\ e(x,y) \Rightarrow N_2(y,y) \\ \forall xyz.\ e(x,y) \wedge e(x,z) \Rightarrow y = z \\ \forall x.\ N_1(x,x) \Rightarrow \exists y.e(x,y) \end{array} \tag{3}$$

An example instance of our employees schema is:

Emp	
Emp	**Emp**
101	101
102	102
103	103

Dept	
Dept	**Dept**
q10	q10
x02	x02

manager	
Emp	**Emp**
101	103
102	102
103	103

worksIn	
Emp	**Dept**
101	q10
102	q10
103	x02

secretary	
Dept	**Emp**
x02	102
q10	101

To save space, we will sometimes present instances in a "joined" format:

Emp		
Emp	**manager**	**worksIn**
101	103	q10
102	102	q10
103	103	x02

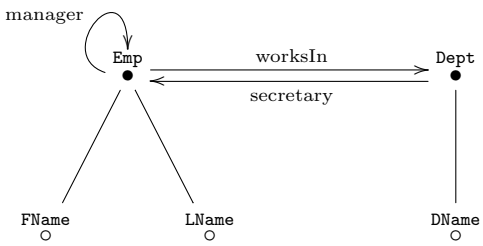
Dept	
Dept	**secretary**
q10	102
x02	101

The natural notion of equality of instances is *isomorphism*. In particular, the actual constants in the above tables should be considered meaningless IDs.

2.4 Attributes

Signatures and instances, as defined above, do not have quite enough structure to be useful in practice. At a practical level, we usually need fixed atomic domains like String and Nat to store actual data. Hence, in this section we extend the functorial data model with *attributes*.

Let S be a signature. A *typing* Γ for S is a mapping from each node N of S to a (possibly empty) set of attribute names and associated base types (Nat, String, etc), written $\Gamma(N)$. We call a pairing of a signature and a typing a *typed sig.nature*. Borrowing from ER-diagram notation, we will write attributes as circles. For example, we might enrich our previous signature with a typing as follows:

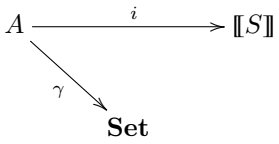


(4)

$$\texttt{Emp.manager.worksIn} = \texttt{Emp.worksIn}$$

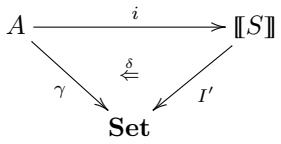
$$\texttt{Dept.secretary.worksIn} = \texttt{Dept}$$

Importantly, path expressions may not refer to attributes; they may only refer to nodes and directed edges. The meaning of Γ is a triple $[\![\Gamma]\!] := (A, i, \gamma)$, where A is a discrete category consisting of the attributes of Γ, i is a functor from A to $[\![S]\!]$ mapping each attribute to its corresponding node, and γ is a functor from A to *Set*, mapping each attribute to its domain (e.g., the set of strings, the set of natural numbers):



2.5 Typed Instances

Let S be a signature and Γ a typing such that $[\![\Gamma]\!] = (A, i, \gamma)$. A *typed instance* I is a pair (I', δ) where $I' : [\![S]\!] \to \mathbf{Set}$ is an (untyped) instance together with a natural transformation $\delta : I' \circ i \Rightarrow \gamma$. Intuitively, δ associates an appropriately typed constant (e.g., a string) to each globally unique ID in I':



We represent the δ-part of a typed instance as a set of binary tables as follows:

- To each node table N and attribute A with type t in $\Gamma(N)$ corresponds a binary "attribute" table mapping the domain of N to values of type t.

In our employees example, we might add the following:

FName	
Emp	**String**
101	Alan
102	Andrey
103	Camille

LName	
Emp	**String**
101	Turing
102	Markov
103	Jordan

DName	
Dept	**Str**
x02	Math
q10	CS

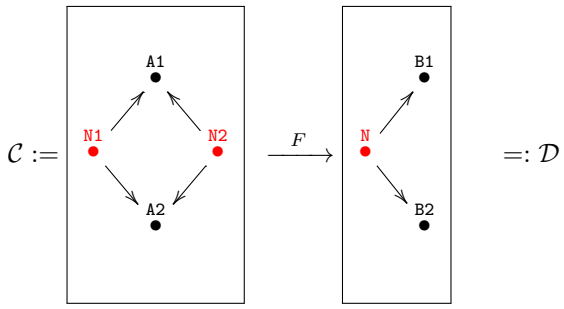
Typed instances form a category, and two instances are *equivalent*, written $\cong$, when they are isomorphic objects in this category. Isomorphism of typed instances captures our expected notion of equality on typed instances, where the "structure parts" are compared for isomorphism and the "attribute parts" are compared for equality under such an isomorphism.

3 Functorial Data Migration

In this section we define the original signature morphisms and data migration functors of [26], as well as "typed signature morphisms" and "typed data migration functors", which are our extension of [26] to attributes. The basic idea is that associated with a sufficiently well-behaved mapping between signatures $F : S \to T$ is a data transformation $\Delta_F : T\text{-}\mathbf{Inst} \to S\text{-}\mathbf{Inst}$ and left and right adjoints $\Sigma_F, \Pi_F : S\text{-}\mathbf{Inst} \to T\text{-}\mathbf{Inst}$. This adjunction appears often in areas outside database theory, under the slogan that "quantification is adjoint to substitution" [16].

3.1 Signature Morphism

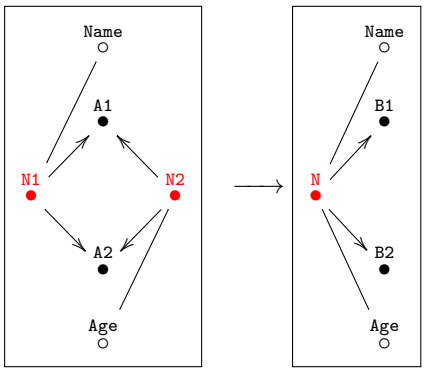
Let $\mathcal{C}$ and $\mathcal{D}$ be signatures. A *signature morphism* $F : \mathcal{C} \to \mathcal{D}$ is a mapping that takes vertices in $\mathcal{C}$ to vertices in $\mathcal{D}$ and arrows in $\mathcal{C}$ to *paths* in $\mathcal{D}$; in so doing, it must respect arrow sources, arrow targets, and path equivalences. In other words, if $p_1 \cong p_2$ is a path equivalence in $\mathcal{C}$, then $F(p_1) \cong F(p_2)$ is a path equivalence in $\mathcal{D}$. Each signature morphism $F : \mathcal{C} \to \mathcal{D}$ determines a unique *schema morphism* $[\![F]\!] : [\![\mathcal{C}]\!] \to [\![\mathcal{D}]\!]$ in the obvious way. Two signature morphisms $F_1 : \mathcal{C} \to \mathcal{D}$ and $F_2 : \mathcal{C} \to \mathcal{D}$ are *equivalent*, written $F_1 \cong F_2$, if they denote isomorphic functors. Below is an example of a signature morphism.



In the above example, the nodes `N1` and `N2` are mapped to `N`, the two morphisms to `A1` are mapped to the morphism to `B1`, and the two morphisms to `A2` are mapped to the morphism to `B2`.

3.2 Typed Signature Morphisms

Intuitively, signature morphisms are extended to typed signatures by providing an additional mapping between attributes. For example, we might have `Name` and `Age` attributes in our source and target typings:

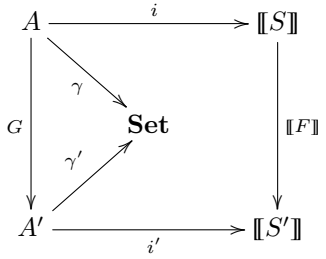


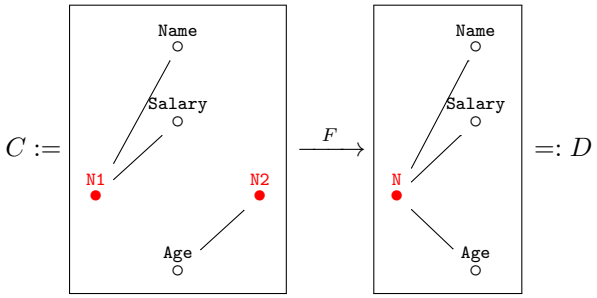
In the above, we map `Name` to `Name` and `Age` to `Age`. More complicated mappings of attributes are also possible, as we will see in the next section.

Formally, let S be a signature and $[\![\Gamma]\!] = (A, i, \gamma)$ a typing for S. Let S' be a signature and $[\![\Gamma']\!] = (A', i', \gamma')$ a typing for S'. A typed signature morphism from (S,Γ) to (S',Γ') consists of a signature morphism $F : S \to S'$ and a functor $G : A \to A'$ such that the following diagram commutes:

Figure 1: Data migration functors induced by a translation $F : \mathcal{C} \to \mathcal{D}$

Name	**Symbol**	**Type**	**Idea of definition**	**Relational partner**
Pullback	Δ_F	$\Delta_F : \mathcal{D}$–**Inst** $\to \mathcal{C}$–**Inst**	Composition with F	Project
Right Pushforward	Π_F	$\Pi_F : \mathcal{C}$–**Inst** $\to \mathcal{D}$–**Inst**	Right adjoint to Δ_F	Join
Left Pushforward	Σ_F	$\Sigma_F : \mathcal{C}$–**Inst** $\to \mathcal{D}$–**Inst**	Left adjoint to Δ_F	Union





3.3 Data Migration Functors

Each signature morphism $F : \mathcal{C} \to \mathcal{D}$ is associated with three data migration functors, Δ_F, Σ_F, and Π_F, which migrate instance data on $\mathcal{D}$ to instance data on $\mathcal{C}$ and vice versa. A summary is given is Figure 1.

Definition 1 (Data Migration Functors)**.** *Let $F : \mathcal{C} \to \mathcal{D}$ be a functor. We will define a functor $\Delta_F : \mathcal{D}$–**Inst** $\to \mathcal{C}$–**Inst**; that is, given a $\mathcal{D}$-instance $I : \mathcal{D} \to$ **Set** we will construct a $\mathcal{C}$-instance $\Delta_F(I)$. This is obtained simply by composing the functors I and F to get*

$$\Delta_F(I) := I \circ F : \mathcal{C} \to \textbf{Set} \qquad \mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{I} \textbf{Set}, \quad \mathcal{C} \xrightarrow{\Delta_F I} \textbf{Set}$$

*Then, $\Sigma_F : \mathcal{C}$–**Inst** $\to \mathcal{D}$–**Inst** is defined as the* left adjoint *to Δ_F, and $\Pi_F : \mathcal{C}$–**Inst** $\to \mathcal{D}$–**Inst** is defined as the* right adjoint *to Δ_F.*

Data migration functors extend to *typed data migration functors* over typed instances in a natural way. We will use typed signatures and instances as we examine each data migration functor in turn below.

3.4 Δ

Consider the following signature morphism $F : C \to D$:

Even though our translation F points forwards (from C to D), our migration functor Δ_F points "backwards" (from D-instances to C-instances). Consider the instance J, on schema D, defined by

$J :=$

N			
ID	**Name**	**Age**	**Salary**
1	Bob	20	$250
2	Sue	20	$300
3	Alice	30	$100

$\Delta_F(J)$ splits up the columns of table `N` according to the translation F, resulting in

$I :=$

N1		
ID	**Name**	**Salary**
1	Bob	$250
2	Sue	$300
3	Alice	$100

N2	
ID	**Age**
a	20
b	20
c	30

Because of the globally unique ID requirement, the IDs of the two tables `N1` and `N2` must be disjoint. Note that Δ never changes the number of IDs associated with a node. For example, table `N1` is ostensibly the "projection" of Age, yet has two rows with age 20.

3.5 Π

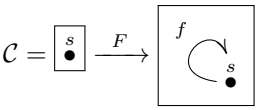
Consider the morphism $F : C \to D$ and C-instance I defined in the previous section. Our migration functor Π_F points in the same direction as F: it takes C-instances to D-instances. In general, Π will take the join of tables. We can Π along any typed signature morphism whose attribute mapping is a bijection. Continuing with our example, we find that $\Pi_F(I)$ will take the cartesian

product of N1 and N2:

N			
ID	**Name**	**Age**	**Salary**
1	Alice	20	$100
2	Alice	20	$100
3	Alice	30	$100
4	Bob	20	$250
5	Bob	20	$250
6	Bob	30	$250
7	Sue	20	$300
8	Sue	20	$300
9	Sue	30	$300

This example illustrates that adjoints are not, in general, inverses. Intuitively, the above instance is a product rather than a join because in there is no path between N1 and N2.

Remark. When the target schema is infinite, on finite inputs Π may create uncountably infinite result instances. Consider the unique signature morphism



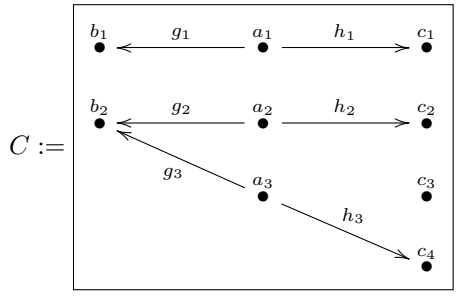
Here $[\![\mathcal{D}]\!]$ has arrows $\{f^n \mid n \in \mathbb{N}\}$ so it is infinite. Given the two-element instance $I\colon \mathcal{C} \to \mathbf{Set}$ with $I(s) = \{\text{Alice}, \text{Bob}\}$, the rowset in the right pushforward $\Pi_F(I)$ is the (uncountable) set of infinite streams in $\{\text{Alice}, \text{Bob}\}$, i.e. $\Pi_F(I)(s) = I(s)^{\mathbb{N}}$.

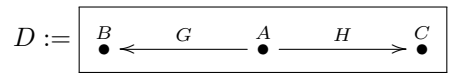
3.6 Σ

For the purposes of this paper we will only define Σ_F for functors F that are *discrete op-fibrations*. Inasmuch as Σ can be thought of as computing unions, functors that are discrete op-fibrations intuitively express the idea that all such unions are over "union compatible" tables.

Definition 2 (Discrete op-fibration)**.** *A functor $F\colon \mathcal{C} \to \mathcal{D}$ is called a* discrete op-fibration *if, for every object $c \in \mathrm{Ob}(\mathcal{C})$ and every arrow $g\colon d \to d'$ in $\mathcal{D}$ with $F(c) = d$, there exists a unique arrow $\bar{g}\colon c \to c'$ in $\mathcal{C}$ such that $F(\bar{g}) = g$.*

Consider the following discrete op-fibration, which maps as to A, bs to B, cs to C, gs to G, and hs to H:





Intuitively, $F\colon \mathcal{C} \to \mathcal{D}$ is a discrete op-fibration if "the columns in each table T of D are exactly matched by the columns in each table in C mapping to T." Since $a_1, a_2, a_3 \mapsto A$, they must have the same column structure and they do: each has two non-ID columns. Similarly each of the b_i and each of the c_i have no non-ID columns, just like their images in D.

To explain the action of Σ_F, consider the following instance:

a1		
ID	**g1**	**h1**
11	7	1

a2		
ID	**g2**	**h2**
16	9	3
15	10	4
14	8	4

a3		
ID	**g3**	**h3**
13	10	17
12	9	18

b1
ID
7
6

b2
ID
10
9
8

c1
ID
2
1

c2
ID
4
3

c3
ID
5

c4
ID
18
17

The result of Σ_F is:

A		
ID	**G**	**H**
16	9	3
15	10	4
14	8	4
13	10	17
12	9	18
11	7	1

B
ID
10
9
8
7
6

C
ID
18
17
5
4
3
2
1

By counting the number of rows it is easy to see that Σ computes union: A has $6 = 1 + 3 + 2$ rows, B has $5 = 3 + 2$ rows, C has $7 = 2 + 2 + 1 + 2$ rows.

We can Σ along any typed signature morphism for which the attribute mapping is also "union compatible in the sense of Codd": string attributes must map to string attributes, integer attributes must map to integer attributes, and so forth. Intuitively, the attribute data will be unioned together in exactly the same way as ID data.

Technically, Σ is a *disjoint* union. However, by requiring our IDs to be globally unique, we can use regular union to implement disjoint union: the globally unique ID assumption ensures that for all distinct tables X, Y in a functorial instance, $|X \cup Y| = |X| + |Y|$

Remark. In this paper we require that signature morphisms used with Σ be discrete op-fibrations. However, it is possible to define Σ for arbitrary, un-restricted signature morphisms, at the following cost:

- Unrestricted Σs may not exist for typed instances.

- An unrestricted variant of FQL will probably not be closed under composition.
- To implement unrestricted Σ we may be required to synthesize new IDs, and termination of this process is semi-decidable.

4 FQL

The goal of this section is to define and study an algebraic query language where every query denotes a composition of data migration functors. Our syntax for queries is designed to build-in the syntactic restrictions discussion in the previous section, and to provide a convenient normal form.

Definition 3 (FQL Query)**.** *A* FQL query Q *from* S *to* T*, denoted* $Q\colon S \rightsquigarrow T$ *is a triple of typed signature morphisms* (F, G, H)*:*

$$S \xleftarrow{F} S' \xrightarrow{G} S'' \xrightarrow{H} T$$

such that

- $[\![S]\!], [\![S']\!], [\![S'']\!]$*, and* $[\![T]\!]$ *are finite*
- G*'s attribute mapping is a bijection*
- H *is a discrete op-fibration with a union compatible attribute mapping*

Semantically, the query $Q\colon S \rightsquigarrow T$ corresponds to a functor $[\![Q]\!]\colon [\![S]\!]$–**Inst** $\to [\![T]\!]$–**Inst** given as follows:

$$[\![Q]\!] := \Sigma_{[\![H]\!]}\Pi_{[\![G]\!]}\Delta_{[\![F]\!]}\colon [\![S]\!]\text{–}\mathbf{Inst} \to [\![T]\!]\text{–}\mathbf{Inst}$$

By choosing two of F, G, and H to be the identity mapping, we can recover Δ, Σ, and Π. However, grouping Δ, Σ, and Π together like this formalizes a query as a disjoint union of a join of a projection. (Interestingly, in the SPCU relational algebra, the order of join and projection are swapped: the normal form is that of unions of projections of joins.)

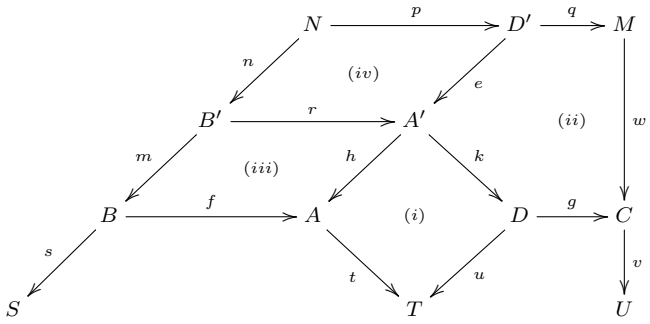
4.1 Composition

Theorem 1 (Closure under composition)**.** *Given any FQL queries* $f\colon S \rightsquigarrow X$ *and* $g\colon X \rightsquigarrow T$*, we can compute an FQL query* $g \cdot f\colon S \rightsquigarrow T$ *such that*

$$[\![g \cdot f]\!] \cong [\![g]\!] \circ [\![f]\!]$$

Sketch of proof. Following the proof in [12, Proposition 1.1.2], it suffices to show the Beck-Chevalley conditions hold for Δ and its adjoints, and that dependent sums distribute over dependent products when the former are indexed by discrete op-fibrations. The proof and algorithm are given in the appendix as Theorem D.4.22 and Corollary D.4.23. □

Detailing the algorithm involves a number of constructions we have not defined, so we only give a brief sketch here. To compose $\Sigma_t\Pi_f\Delta_s$ with $\Sigma_v\Pi_g\Delta_u$ we must construct the following diagram:



First we form the pullback (i). Then we form the typed distributivity diagram (ii) as in Proposition E.2.5. Then we form the typed comma categories (iii) and (iv) as in Proposition E.2.1. The result then follows from Proposition E.2.4, E.2.5, and Corollary E.2.3. The composed query is $\Sigma_{w;v}\Pi_{p;q}\Delta_{n;m;s}$.

5 SQL Generation

In this section we define SQL generation algorithms for Δ, Σ, and Π. Let $F\colon S \to T$ be a signature morphism.

5.1 Δ

Theorem 2. *We can compute a SQL program* $[F]_\Delta\colon T \to S$ *such that for every* T*-instance* $I \in T$–**Inst***, we have* $\Delta_F(I) \cong [F]_\Delta(I)$*.*

Proof. See Construction D.2.1. □

We sketch the algorithm as follows. We are given a T-instance I, presented as a set of binary functions, and are tasked with creating the S-instance $\Delta_F(I)$. We describe the result of $\Delta_F(I)$ on each table in the result instance by examining the schema S:

- for each node N in S, the binary table $\Delta_F(N)$ is the binary table $I_{F(N)}$
- for each attribute A in S, the binary table $\Delta_F(A)$ is the binary table $I_{F(A)}$
- Each edge $E : X \to Y$ in S maps to a path $F(E) : FX \to FY$ in T. We compose the binary edges tables making up the path $F(E)$, and that becomes the binary table $\Delta_F(E)$.

The SQL generation algorithm for Δ sketched above does not maintain the globally unique ID requirement. For example, Δ can copy tables. Hence we must also generate SQL to restore this invariant.

5.2 Σ

Theorem 3. *Suppose F is a discrete op-fibration and has a union compatible attribute mapping. Then we can compute a SQL program $[F]_\Sigma \colon S \to T$ such that for every S-instance $I \in S$–**Inst**, we have $\Sigma_F(I) \cong [F]_\Sigma(I)$.*

Proof. See Construction D.2.10. □

We sketch the algorithm as follows. We are given a S-instance I, presented as a set of binary functions, and are tasked with creating the T-instance $\Sigma_F(I)$. We describe the result of $\Sigma_F(I)$ on each table in the result instance by examining the schema T:

- for each node N in T, the binary table $\Delta_F(N)$ is the union of the binary node tables in I that map to N via F.
- for each attribute A in T, the binary table $\Delta_F(A)$ is the union of the binary attribute tables in I that map to A via F.
- Let $E : X \to Y$ be an edge in T. We know that for each $c \in F^{-1}(X)$ there is at least one path p_c in C such that $F(p_c) \cong e$. Compose p_c to a single binary table, and define $\Sigma_F(E)$ to be the union over all such c. The choice of p_c will not matter.

5.3 Π

Theorem 4. *Suppose $[\![S]\!]$ and $[\![T]\!]$ are finite, and F has a bijective attribute mapping. Then we can compute a SQL program $[F]_\Pi \colon S \to T$ such that for every S-instance $I \in S$–**Inst**, we have $\Pi_F(I) \cong [F]_\Pi(I)$.*

Proof. See Construction D.2.14. □

The algorithm for Π_F is more complicated than for Δ_F and Σ_F. In particular, its construction makes use of comma categories, which we have not yet defined, as well as "limit tables", which are a sort of "join all". We define these now.

Let B be a typed signature and H a typed B-instance. The limit table lim_B is computed as follows. First, take the cartesian product of every binary reflexive node table in B, and naturally join the attribute tables of B. Then, for each edge $e : n_1 \to n_2$ filter the table by $n_1 = n_2$. This filtered table is the limit table lim_B.

Let $S : A \to C$ and $T : B \to C$ be functors. The comma category $(S \downarrow T)$ has for objects triples (α, β, f), with α an object in A, β and object in B, and $f : S(\alpha) \to T(\beta)$ a morphism in C. The morphisms from (α, β, f) to (α', β', f') are all the pairs (g, h) where $g : \alpha \to \alpha'$ and $h : B \to B'$ are morphisms in A and B respectively such that $T(h) \circ f = f' \circ S(g)$.

The algorithm for Π_F proceeds as follows. First, for every object $d \in T$ we consider the comma category $B_d := (d \downarrow F)$ and its projection functor $q_d : (d \downarrow F) \to C$. (Here we treat d as a functor from the empty category). Let $H_d := I \circ q_d : B_d \to$ **Set**, constructed by generating SQL for $\Delta(I)$. We say that the limit table for d is $lim_{B_d} H_d$, as described above. Now we can describe the target tables in T:

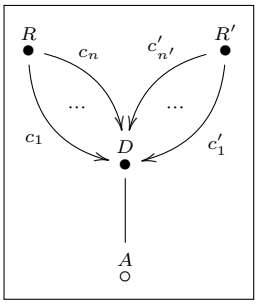
- for each node N in T, generate globally unique IDs for each row in the limit table for N. These GUIDs are $\Pi_F(N)$.
- for each attribute $A : X \to \mathit{type}$ in T, $\Pi_F(A)$ will be a projection from the limit table for X.
- for each edge $E : X \to Y$ in T, $\Pi_F(E)$ will be a map from X to Y obtained by joining the limit tables for X and Y on columns which "factor through" E.

Remark. Our SQL generation algorithms for Δ and Σ work even when $[\![S]\!]$ and $[\![T]\!]$ are infinite, but this is not the case for Π. But if we are willing to take on infinite SQL queries, we have Π in general; see Construction D.2.14. Alternatively, it is possible to target a category besides **Set**, provided that category is complete and co-complete. For example, it is possible to store our data not as relations, but as programs in the Turing-complete language PCF [26] [24].

6 SQL in FQL

Because FQL operates over functorial instances, it is not possible to implement many relational/SQL operations directly in FQL. However, we can always "encode" arbitrary relational databases as functorial instances.

Relational signatures are encoded as "pointed" signatures with a single attribute that can intuitively be thought of as the domain. For example, the signature for a relational schema with two relations $R(c_1, \ldots, c_n)$ and $R'(c'_1, \ldots, c'_{n'})$ has the following form:



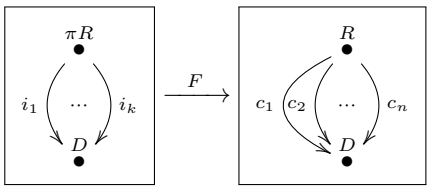
We might expect that the $c_1, \ldots, c_n$ would be attributes of node R, and hence there would be no node D, but that doesn't work: attributes may not be joined on. Instead, we must think of each column of R as a mapping from R's domain to IDs in D, and A as a mapping from IDs in D to constants. We will write $[R]$ for

the encoding of a relational schema R and $[I]$ for the encoding of a relational R-instance I.

6.1 Conjunctive queries (Bags)

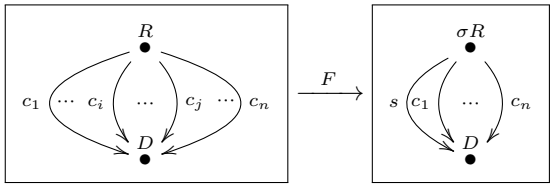
FQL can implement relational conjunctive (SPC/select-from-where) queries *under bag semantics* directly using the above encoding. In what follows we will omit the attribute A from the diagrams. For simplicity, we will assume the minimal number of tables required to illustrate the construction. We may express the (bag) operations $\pi, \sigma, \times$ as follows:

- Let R be a table. We can express $\pi_{i_1,\ldots,i_k} R$ using the pullback Δ_F, where F is the following functor



  This construction is only appropriate for bag semantics because πR will will have the same number of rows as R.

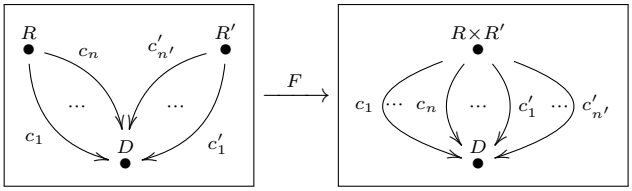
- Let R be a table. We can express $\sigma_{i=j} R$ using the pullback of the right pushforward $\Delta_F \Pi_F$, where F is the following functor:



  Here $F(c_i) = F(c_j) = s$.

  Remark. In fact, pulling back along F creates the duplicated column. If we wanted the more economical query in which the column is not duplicated, we would use Π_F instead of $\Delta_F \Pi_F$.

- Let R and R' be tables. We can express $R \times R'$ as the right pushforward Π_F, where F is the following functor:



Theorem 5 (Conjunctive RA in FQL (bags))**.** *Let R be a relational schema, and I an R-instance. For every conjunctive (SPC) query q under bag semantics we can compute a FQL program $[q]$ such that $[q(I)] \cong [q]([I])$.*

Proof. See Proposition D.3.3. □

6.2 Conjunctive Queries (Sets)

The encoding strategy described above fails for relational conjunctive queries under their typical set-theoretic semantics. For example, consider a simple two column relational table R, its encoded FQL instance $[R]$, and an attempt to project off col1 using Δ:

R	
col1	**col2**
x	y
x	z

$[R]$		
ID	**col1**	**col2**
0	x	y
1	x	z

$\Delta[R]$	
ID	**col1**
0	x
1	x

This answer is incorrect under projection's set-theoretic semantics, because it has the wrong number of rows. However, it is possible to extend FQL with an operation, $relationalize_T : T\text{-}\mathbf{Inst} \to T\text{-}\mathbf{Inst}$, such that FQL+*relationalize* can implement every relational conjunctive query under normal set-theoretic semantics. Intuitively, $relationalize_T$ converts a T-instance to a smaller T-instance by equating IDs that cannot be distinguished. The relationalization of the above instance would be

$relationalize(\Delta[R])$	
ID	**col1**
0	x

which is correct for the set-theoretic semantics. We have the the following theorem:

Theorem 6 (Conjunctive RA in FQL (sets))**.** *Let R be a relational schema, and I an R-instance. For every conjunctive (SPC) query q under set semantics we can compute a FQL program $[q]$ such that $[q(I)] \cong relationalize_T([q]([I]))$.*

Proof. See Proposition D.3.3 □

Provided T is obtained from a relational schema (i.e., has the pointed form described in this section), $relationalize_T$ can easily be implemented using the same SPCU+GUIDgen operations as Δ, Σ, Π.

7 Schema Transformation

In this section we prove that the functorial data model is a *schema transformation framework* in the sense of Alagic and Bernstein [2] [5]. In fact, Alagic and Bernstein's notion is essentially equivalent to that of *institution* [13].

It is easy to describe basic functorial data migration using Alagic and Bernstein's categorical model theory [2]. However, there is a fundamental difference in terminology. In this paper, we use "signature" for a finitely presented category, and "schema" for the potentially infinite category it denotes. Alagic and Bernstein leave "signature" undefined, and use "schema" for a pair of a signature and a set of sentences in some logical formalism. Hence, every schema in our sense can be represented as a potentially infinite schema in their sense (its presentation), and every signature in our sense can be represented as a finite schema in

their sense (its finite presentation). We consider path-equivalences to be part of our signatures, so our instances obey path-equivalences by definition; in their system, path-equivalences are not part of signatures and a separate satisfaction relation $\models_S$ characterizes the path-equivalences that hold in an S-instance. In the following definition, we use "signature" and "schema" in their sense. Let L be a logical formalism such as first-order logic. Then

Definition 4. *A schema transformation framework is a tuple* $(\mathbf{Sig}, Sig_0, Sen, Db, \models)$ *such that*

1. **Sig** *is a category, representing signatures together with their morphisms, and* **Sig** *has an initial object* Sig_0.
2. $Sen : \mathbf{Sig} \to \mathbf{Set}$ *is a functor such that* $Sen(Sig)$ *is the set of all* L*-sentences over the signature* Sig.
3. Db: $\mathbf{Sig} \to \mathbf{Cat}^{op}$ *is a functor sending each signature* Sig *to the category* $Db(Sig)$ *of* Sig*-instances and their morphisms.*
4. *For each signature* Sig*, the satisfaction relation* $\models_{Sig} \subset |Db(Sig)| \times Sen(Sig)$ *is such that for each schema signature morphism* ϕ: $Sig_A \to Sig_B$*, the following integrity requirement holds for each* Sig_B *database* d_B *and each sentence* $e \in Sen(Sig_A)$:

$$d_b \models_{Sig_B} Sen(\phi)(e) \quad \textit{iff} \quad Db(\phi)(d_b) \models_{Sig_A} e \tag{5}$$

Theorem 7. *The functorial data model is a schema transformation framework.*

Proof. We use the following definitions:

1. **Sig** is a the category of finitely presented freely generated categories, and Sig_0 is the empty category.
2. $Sen(Sig)$ is the set of all equations over the signature Sig.
3. Db: $\mathbf{Sig} \to \mathbf{Cat}^{op}$ is Δ (see Section 3.4).
4. $I \models p_1 \cong p_2$ is defined in the intuitive way.

See Proposition B.1.1. □

8 FQL and EDs

In this section we discuss some in-progress work about implementing FQL using traditional schema mapping languages such as embedded dependencies (EDs) [1] and so-tgds [10]. In this section, all our instances are untyped (i.e., contain only IDs; have no attributes).

We begin by noting some technical complications that arise when relating FQL, as described in this paper, to relational schema mapping languages. The first is that untyped instances in FQL correspond to relational instances that are made up entirely of meaningless globally unique IDs. If we think of FQL IDs as labelled nulls [8] or variables [1], then every natural transformation is a homomorphism, and vice versa. However, the correct notion of equivalence to use for untyped FQL instances is isomorphism, not homomorphic equivalence (instances I and J are homomorphically equivalent when there are homomorphisms $h : I \to J$ and $h' : J \to I$, but h and h' need not be inverses).

The second complication is that the constraints required to hold of functorial instances, e.g., (2)(3), are not all embedded dependencies. In particular, the "globally unique ID assumption" must be stated using negation:

$$\forall x, S(x, x) \to \neg T_1(x, x) \wedge \neg T_2(x, x) \wedge \dots$$

However, the unique ID assumption is an artifact of our SQL generation strategy, rather than a requirement of functorial data migration itself. In particular, the unique ID assumptions lets us use relational union to implement disjoint union.

Bearing these two complications in mind, we conjecture the following: every untyped un-restricted Σ can be implemented as the *initial solution* to a set of EDs; every untyped Π can be implemented as the *terminal solution* to a set of EDs; and every untyped Δ can be implemented as the initial *and* terminal solution to a set of EDs. We now explain this terminology.

If φ is a set of EDs and I an untyped FQL instance, a *solution* to (φ, I) is a finite instance U such that $U \models \phi$ and there exists a natural transformation $h : I \to U$. U is *initial* if, for every other solution U' such that $h' : I \to U'$, there exists a unique natural transformation $f : U \to U'$ such that $h' = h; f$. Dually, U is *terminal* if it has a unique natural transformation $h : U' \to U$ for every other solution U', such that the appropriate diagram commutes.

We implement untyped Δ, Σ, Π with EDs as follows. Let $F : C \to D$ be an untyped signature morphism. We define the disjoint union signature $C + D$ by taking the disjoint union of C and D's nodes, arrows, and equations. Then we define the signatures $C \star_\Sigma D, C \star_\Pi D, C \star_\Delta D$ by adding additional paths and equations to $C + D$:

- the arrows of $C \star_\Pi D$ contain, for each each node $c \in C$, an arrow $m_c : F(c) \to c$; the equations of $C \star_\Pi D$ contain, for every arrow $f : c \to c'$ in C, the equation $F(f); m_{c'} = m_c; f$
- the arrows of $C \star_\Sigma D$ contain, for each each node $c \in C$, an arrow $l_c : c \to F(c)$; the equations of

$C \star_\Sigma D$ contain, for every arrow $f : c \to c'$ in C, the equation $l_c; F(f) = f; l_{c'}$

- the arrows of $C \star_\Delta D$ contain the l_c and m_c described above, along with equations $m_c; l_c = id$ and $l_c; m_c = id$.

The set of EDs that implements Δ, Σ, Π is then simply the functorality EDs from (2)(3) and the path equality constraints of $C \star D$, which are easy to express as EDs. Operationally, we start with a C (resp, D) instance I, we compute an initial or terminal $C \star D$ solution IJ, and the desired D (resp, C) instance J will be a subset of the tables of IJ.

We have modified the FQL compiler to emit the EDs described above, and to solve them using the standard chase, the naive/parallel chase, and the core chase [9]. We find that, on every example we've tried, both the parallel chase and the standard chase correctly compute Σ and Δ data migrations for untyped instances. The core chase does not, because the core of a chased solution I will only be homomorphically equivalent to I, not isomorphic to I, as we require.

Running the chase on the EDs generated for Π always results in an empty target instance, because the "existential force" of the EDs for Π is target-to-source. A category-theoretic understanding is that for our purposes, the parallel and standard chases compute initial instances, whereas our Π migrations are terminal instances. We are unaware of any traditional relational algorithm that implements terminal solutions to EDs.

9 Conclusion

We are working to extend FQL with additional operations such as difference, selection by a constant, and aggregation. In addition, we are studying the systems aspects of FQL, such as FQL's equational theory and the data structures and algorithms that would be appropriate for a native, non-SQL implementation of FQL.

More speculatively, for every monad M in the category of sets, the functorial data model admits generalized M-instances [27], which are database instances where every foreign-key reference to a value of type t has been replaced by a value of type $M\ t$. We speculate that this additional structure can be used to extend FQL to handle purely functional implementations of monadic computational effects in the traditional style [23] [19].

Acknowledgements. The authors would like to thank Lucian Popa for answering our many questions about database theory.

References

[1] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] Suad Alagic and Philip A. Bernstein. A model theory for generic schema management. In *DBPL*, 2001.

[3] Leo Bachmair, Nachum Dershowitz, and David A. Plaisted. Completion without failure, 1989.

[4] Michael Barr and Charles Wells, editors. *Category theory for computing science, 2nd ed.* 1995.

[5] Philip A. Bernstein and Sergey Melnik. Model management 2.0: manipulating richer mappings. SIGMOD '07, 2007.

[6] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *EDBT*, 1992.

[7] S. Carmody, M. Leeming, and R.F.C. Walters. The todd-coxeter procedure and left kan extensions. *Journal of Symbolic Computation*, 19(5):459 – 488, 1995.

[8] Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. PODS '08.

[9] Ronald Fagin, Phokion G. Kolaitis, Alan Nash, and Lucian Popa. Towards a theory of schema-mapping optimization. PODS '08.

[10] Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. Composing schema mappings: Second-order dependencies to the rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, December 2005.

[11] Michael Fleming, Ryan Gunther, and Robert Rosebrugh. A database of categories. *J. SYMBOLIC COMPUT*, 35:127–135, 2002.

[12] N. Gambino and J. Kock. Polynomial functors and polynomial monads.

[13] J.A. Goguen and R.M. Burstall. Introducing institutions. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*. 1984.

[14] Laura M. Haas, Mauricio A. Hernández, Howard Ho, Lucian Popa, and Mary Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD '05:*.

[15] Jieh Hsiang and Michael Rusinowitch. On word problems in equational theories. In *Automata, Languages and Programming*, volume 267 of *LNCS*. 1987.

[16] Bart Jacobs. *Categorical logic and type theory*. PhD thesis, Mathematics, Amsterdam, Lausanne, New York, 1999.

[17] Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium. Vol. 1*. 2002.

[18] J. Lambek and P. J. Scott. *Introduction to higher order categorical logic*. 1986.

[19] S. Kazem Lellahi and Val Tannen. A calculus for collections and aggregates. In *CTCS '97*, 1997.

[20] Sergey Melnik. *Generic Model Management: Concepts And Algorithms (Lecture Notes in Computer Science)*. 2004.

[21] Sergey Melnik, Philip A. Bernstein, Alon Halevy, and Erhard Rahm. Supporting executable mappings in model management. SIGMOD '05.

[22] Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. Rondo: a programming platform for generic model management. In *SIGMOD '03*.

[23] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[24] Gordon D. Plotkin. Lcf considered as a programming language. *Theor. Comput. Sci.*, 5(3):223–255, 1977.

[25] David I. Spivak. Database queries and constraints via lifting problems. 2012.

[26] David I. Spivak. Functorial data migration. *Inf. Comput.*, 217:31–51, August 2012.

[27] David I. Spivak. Kleisli database instances, August 2012. http://arxiv.org/abs/1209.1011.

[28] Limsoon Wong. *Querying nested collections*. PhD thesis, Philadelphia, PA, USA, 1994. AAI9503855.

A Definition of full, untyped FQL

The category of finitely presented categories and mappings is bi-cartesian closed, and for every finitely-presented category T, the category of T instances and their morphisms is a topos (bi-cartesian closed category with a subobject-classifier). Hence, FQL has the following structure.

Let $\mathcal{T}$ indicate finitely presented categories, $\mathcal{F}_{T_1,T_2} : T_1 \to T_2$ finitely presented functors, $\mathcal{I}_T$ finitely presented T-instances (functors from T to the category of sets), and $\mathcal{E}_{I^1_T,I^2_T} : I^1 \Rightarrow I^2$ finitely presented natural transformations (database homomorphisms from T-instances I^1_T to I^2_T). The syntax of FQL types T, mappings F, instances I, and transformations (database homomorphisms) E is given by the following grammar:

$$T ::= 0 \mid 1 \mid T + T \mid T \times T \mid T^T \mid \mathcal{T}$$

$$F ::= id_T \mid F; F \mid proj^1_{T,T} \mid proj^2_{T,T} \mid inj^1_{T,T} \mid inj^2_{T,T} \mid F \otimes F \mid F \oplus F \mid ev_{T,T} \mid \Lambda F \mid \mathcal{F}_{T,T}$$
$$\mid dist^1_{T,T,T} \mid dist^2_{T,T,T} \mid tt_T \mid ff_T$$

$$I ::= 0_T \mid 1_T \mid I + I \mid I \times I \mid I^I \mid \mathcal{I}_T \mid \Omega_T \mid \Delta_F I \mid \Sigma_F I \mid \Pi_F I$$

$$E ::= id_I \mid E; E \mid proj^1_{I,I} \mid proj^2_{I,I} \mid inj^1_{I,I} \mid inj^2_{I,I} \mid E \otimes E \mid E \oplus E \mid ev_{I,I} \mid \Lambda E \mid \mathcal{E}_{I^1,I^2}$$
$$\mid dist^1_{I,I,I} \mid dist^2_{I,I,I} \mid tt_I \mid ff_I \mid eq_I \mid \top_T$$

A.1 Isomorphisms of schemas and instances

The following isomorphisms can always be constructed with the appropriate terms:

$$T_1 \times (T_2 \times T_3) \cong (T_1 \times T_2) \times T_3 \qquad T_1 \times T_2 \cong T_2 \times T_1 \qquad T \times 1 \cong 1 \qquad 1^T \cong 1$$

$$T^1 \cong T \qquad (T_1 \times T_2)^{T_3} \cong T_1^{T_3} \times T_2^{T_3} \qquad (T_1^{T_2})^{T_3} \cong T_1^{T_2 \times T_3}$$

$$T_1 + (T_2 + T_3) \cong (T_1 + T_2) + T_3 \qquad T_1 + T_2 \cong T_2 + T_1 \qquad T \times 0 \cong 0 \qquad T + 0 \cong T$$

$$T^0 \cong 1 \qquad T_1 \times (T_2 + T_3) \cong (T_1 \times T_2) + (T_1 \times T_3) \qquad T_1^{T_2+T_3} \cong T_1^{T_2} \times T_1^{T_3}$$

A.2 Mappings

$$\frac{}{id_T : T \to T} \qquad \frac{F : T_1 \to T_2 \qquad G : T_2 \to T_3}{F;G : T_1 \to T_3} \qquad \frac{}{tt_T : T \to 1}$$

$$\frac{}{proj^1_{T_1,T_2} : T_1 \times T_2 \to T_1} \qquad \frac{}{proj^2_{T_1,T_2} : T_1 \times T_2 \to T_2} \qquad \frac{F : T_1 \to T_2 \qquad G : T_1 \to T_3}{F \otimes G : T_1 \to T_2 \times T_3}$$

$$\frac{}{ff_T : 0 \to T} \qquad \frac{}{inj^1_{T_1,T_2} : T_1 \to T_1 + T_2} \qquad \frac{}{inj^2_{T_1,T_2} : T_2 \to T_1 + T_2}$$

$$\frac{F : T_2 \to T_1 \qquad G : T_3 \to T_1}{F \oplus G : T_2 + T_3 \to T_1} \qquad \frac{}{ev_{T_1,T_2} : T_1^{T_2} \times T_2 \to T_1} \qquad \frac{F : T_1 \times T_2 \to T_3}{\Lambda F : T_1 \to T_3^{T_2}}$$

$$\frac{}{\mathcal{F}_{T_1,T_2} : T_1 \to T_2} \qquad \frac{}{dist^1_{T_1,T_2,T_3} : T_1 \times (T_2 + T_3) \to (T_1 \times T_2) + (T_1 \times T_3)}$$

$$\frac{}{dist^2_{T_1,T_2,T_3}(T_1 \times T_2) + (T_1 \times T_3) \to T_1 \times (T_2 + T_3)}$$

$$id; f = f \qquad f; id = f \qquad f; (g; h) = (f; g); h \qquad \Lambda ev = id \qquad \Lambda f \otimes a \ ; \ ev = id \otimes a; f$$

$$f \otimes g; proj^1 = f \qquad f \otimes g; proj^2 = g \qquad f; proj^1 \otimes f; proj^2 = f \qquad \frac{f : T \to 1}{f = tt}$$

$$\frac{f : 0 \to T}{f = ff} \qquad inj^1; f \oplus g = f \qquad inj^2; f \oplus g = g \qquad inj^1; f \oplus inj^2; f = f$$

$$dist^1; dist^2 = id \qquad dist^2; dist^1 = id$$

A.3 Instances

We omit isomorphisms for data migrations Δ, Σ, Π and sub-object classifier Ω.

$$\frac{}{0_T : T - inst} \qquad \frac{}{1_T : T - inst} \qquad \frac{I : T - inst \qquad J : T - inst}{I + J : T - inst}$$

$$\frac{I : T - inst \qquad J : T - inst}{I \times J : T - inst} \qquad \frac{I : T - inst \qquad J : T - inst}{I^J : T - inst} \qquad \frac{}{\Omega_T : T - inst}$$

$$\frac{}{\mathcal{I}_T : T - inst} \qquad \frac{F : T_1 \to T_2 \qquad I : T_2 - inst}{\Delta_F I : T_1 - inst} \qquad \frac{F : T_1 \to T_2 \qquad I : T_1 - inst}{\Sigma_F I : T_2 - inst}$$

$$\frac{F : T_1 \to T_2 \qquad I : T_1 - inst}{\Pi_F : T_2 - inst}$$

A.4 Transformations

We omit the equational theory for eq and $\top$.

$$\frac{}{id_I : I \Rightarrow I} \qquad \frac{I^1, I^2, I^3 : T - inst \qquad E : I^1 \Rightarrow I^2 \qquad E' : I^2 \Rightarrow I^3}{E; E' : I^1 \Rightarrow I^3} \qquad \frac{I : T - inst}{tt_I : I \Rightarrow 1_T}$$

$$\frac{I^1, I^2 : T - inst}{proj^1_{I^1,I^2} : I^1 \times I^2 \Rightarrow I^1} \qquad \frac{I^1, I^2 : T - inst}{proj^2_{I^1,I^2} : I^1 \times I^2 \Rightarrow I^2}$$

$$\frac{I^1, I^2 : T - inst \qquad E : I^1 \Rightarrow I^2 \qquad E' : I^1 \Rightarrow I^3}{E \otimes E' : I^1 \Rightarrow I^2 \times I^3} \qquad \frac{I : T - inst}{ff_I : 0_T \Rightarrow I}$$

$$\frac{I^1, I^2 : T - inst}{inj^1_{I^1,I^2} : I^1 \Rightarrow I^1 + I^2} \qquad \frac{I^1, I^2 : T - inst}{inj^2_{I^1,I^2} : I^2 \Rightarrow I^1 + I^2}$$

$$\frac{I^1, I^2 : T - inst \qquad E : I^2 \Rightarrow I^1 \qquad E' : I^3 \Rightarrow I^1}{E \oplus E' : I^2 + I^3 \Rightarrow I^1} \qquad \frac{I^1, I^2 : T - inst}{ev_{I^1,I^2} : {I^1}^{I^2} \times I^2 \Rightarrow I^1}$$

$$\frac{I^1, I^2, I^3 : T - inst \qquad E : I^1 \times I^2 \Rightarrow I^3}{\Lambda E : I^1 \Rightarrow {I^3}^{I^2}} \qquad \frac{I^1, I^2 : T - inst}{\mathcal{E}_{I^1,I^2} : I^1 \Rightarrow I^2}$$

$$\frac{I^1, I^2, I^3 : T - inst}{dist^1_{I^1,I^2,I^3} : I^1 \times (I^2 + I^3) \Rightarrow (I^1 \times I^2) + (I^1 \times I^3)}$$

$$\frac{I^1, I^2, I^3 : T - inst}{dist^2_{I^1,I^2,I^3} (I^1 \times I^2) + (I^1 \times I^3) \Rightarrow I^1 \times (I^2 + I^3)} \qquad \frac{I : T - inst}{eq_I : I \times I \Rightarrow \Omega_T}$$

$$\frac{I : T - inst}{\top_T : 1_T \Rightarrow \Omega_T}$$

A.5 λ-calculus formulation

FQL mappings and transformations may be programmed using a λ-calculus as follows. λ-terms are:

$$e ::= x \mid \lambda x : t.e \mid ee \mid () \mid (e, e) \mid e.1 \mid e.2 \mid e = e \mid inl_t\ e \mid inr_t\ e \mid \bot_t \mid case\ e\ of\ \lambda x : t.e\ or\ \lambda x : t.e$$

A context is a list of bindings of variables to types:

$$\Gamma ::= - \mid \Gamma, x : t$$

The three-place typing relation $\Gamma \vdash e : t$ is inductively defined as

$$\frac{}{\Gamma, x : t \vdash x : t} \qquad \frac{\Gamma \vdash x : t}{\Gamma, y : s \vdash x : t} \qquad \frac{\Gamma, x : s \vdash e : t}{\Gamma \vdash \lambda x : s.e : s \to t} \qquad \frac{\Gamma \vdash f : s \to t \qquad \Gamma \vdash e : s}{\Gamma \vdash fe : t}$$

$$\frac{}{\Gamma \vdash () : 1} \qquad \frac{\Gamma \vdash e : 0}{\Gamma \vdash \bot_t\ e : t} \qquad \frac{\Gamma \vdash e : s \times t}{\Gamma \vdash e.1 : s} \qquad \frac{\Gamma \vdash e : s \times t}{\Gamma \vdash e.2 : t} \qquad \frac{\Gamma \vdash e : s \qquad \Gamma \vdash f : t}{\Gamma \vdash (e, f) : s \times t}$$

$$\frac{\Gamma \vdash e : t \qquad f : t}{\Gamma \vdash e = f : \Omega} \qquad \frac{\Gamma \vdash e : s}{\Gamma \vdash inl_t\ e : s + t} \qquad \frac{\Gamma \vdash e : s}{\Gamma \vdash inr_t\ e : t + s}$$

$$\frac{\Gamma \vdash e : s + t \qquad \Gamma, x : s \vdash f : u \qquad \Gamma, y : t \vdash g : u}{\Gamma \vdash case\ e\ of\ \lambda x : s.f\ or\ \lambda y : t.g\ : u}$$

A typing derivation $\Gamma \vdash e : t$ translates to a combinator $[\![\Gamma]\!] \to t$, where

$$[\![-]\!] = 1 \qquad [\![\Gamma, x : t]\!] := [\![\Gamma]\!] \times t$$

The translation $[\![]\!]$ is:

$$[\![\Gamma, x : t \vdash x : t]\!] := proj^2 \qquad [\![\Gamma, y : s \vdash x : t]\!] := proj^1; [\![\Gamma \vdash x : t]\!] \qquad [\![\Gamma \vdash ()]\!] := tt_{[\![\Gamma]\!]}$$

$$[\![\Gamma \vdash (e, f) : s \times t]\!] := [\![\Gamma \vdash e : s]\!] \otimes [\![\Gamma \vdash f : t]\!] \qquad [\![\Gamma \vdash e.1 : t]\!] := [\![\Gamma \vdash e : s \times t]\!]; proj^1$$

$$[\![\Gamma \vdash e.2 : t]\!] := [\![\Gamma \vdash e : s \times t]\!]; proj^2 \qquad [\![\Gamma \vdash \bot\ e : t]\!] := [\![\Gamma \vdash e : 0]\!]; ff$$

$$[\![\Gamma \vdash inl_t\ e : s + t]\!] := [\![\Gamma \vdash e : s]\!]; inj^1 \qquad [\![\Gamma \vdash inr_t\ e : t + s]\!] := [\![\Gamma \vdash e : s]\!]; inj^2$$

$$[\![\Gamma \vdash case\ e\ of\ \lambda x.g\ else\ \lambda y.g]\!] := (id \otimes [\![\Gamma \vdash e : s + t]\!]); dist^1; ([\![\Gamma, x : s \vdash f : u]\!] \oplus [\![\Gamma, y : t \vdash g : u]\!])$$

$$[\![\Gamma \vdash \lambda x : s.e : s \to t]\!] := \Lambda [\![\Gamma, x : s \vdash e : t]\!]$$

$$[\![\Gamma \vdash fe : t]\!] := \langle [\![\Gamma \vdash f : s \to t]\!], [\![\Gamma \vdash e : s]\!] \rangle; ev$$

$$[\![\Gamma \vdash e = f : \Omega]\!] := \langle [\![\Gamma \vdash e : t]\!], [\![\Gamma \vdash f : t]\!] \rangle; eq$$

B FDM as Schema Transformation

Theorem B.1.1. *The functorial data model is a schema transformation framework.*

Proof. We use the following definitions:

1. **Sig** is a the category of finitely presented freely generated categories, and Sig_0 is the empty category.
2. $Sen(Sig)$ is the set of all equations over the signature Sig.
3. Db: **Sig** $\to$ **Cat**^{op} is Δ (see Section 3.4).

4. $I \models p_1 \cong p_2$ is defined in the intuitive way.

The result follows from Lemmas 1, 2, 3, and 4 below. □

Lemma 1 (Requirement 1)**.** *Signatures form a category with an initial object.*

Proof. Proved as Lemma B.1.2. □

Lemma 2 (Requirement 2)**.** *We have a functor* $Sen\colon \mathbf{Sig} \to \mathbf{Set}$ *such that* $Sen(Sig)$ *is the set of all equations between generators over the signature* Sig*.*

Proof. Proved as Lemma B.1.3. □

Lemma 3 (Requirement 3)**.** *The assignment* $Sig \mapsto Db(Sig)$ *is functorial. That is, for each morphism of signatures* $F\colon Sig \to Sig'$ *there is an induced morphism* $Db(Sig) \to Db(Sig')$ *in* $\mathbf{Cat}^{\mathrm{op}}$*, i.e. a functor* $Db(F)\colon Db(Sig') \to Db(Sig)$*. We write* $Db\colon \mathbf{Sig} \to \mathbf{Cat}^{\mathrm{op}}$*, or as is slightly easier to parse,*

$$Db\colon \mathbf{Sig}^{\mathrm{op}} \to \mathbf{Cat}.$$

Proof. Proved as Lemma B.1.4. □

Lemma 4 (Requirement 4)**.** *For each schema signature morphism* $\phi\colon Sig_A \to Sig_B$*, the following statement holds for each* Sig_B *instance* d_B *and each path equation* $e \in Sen(Sig_A)$*:*

$$d_B \models_{Sig_B} Sen(\phi)(e) \qquad \textit{iff} \qquad Db(\phi)(d_B) \models_{Sig_A} e$$

Proof. In categorical terms, the above says that given a functor $\phi\colon A \to B$ and a functor $d_B\colon B \to \mathbf{Set}$, the composition $I \circ \phi\colon A \to \mathbf{Set}$ is a functor. See B.1.5 for the full proof. □

Moreover, our schemas have a further useful schema-join property property:

Theorem 8. *There exists a schema join of any two schemas over any subschema (in particular over the initial schema).*

Proof. Given a diagram of schemas $B \leftarrow A \to C$, the schema join (defined as the colimit in **Cat**) always exists. □

In this section we have only proved that the *functorial data model* is a schema transformation framework. However, it follows that the Δ fragment of FQL is a schema transformation framework. It may be the case that larger fragments of FQL are schema transformation frameworks; see the remarks around Definition D.5.5.

The following lemmas prove the four requirements for a schema transformation framework, as in Definition 4 and discussed in Section 7.

Lemma B.1.2 (Requirement 1)**.** *Signatures form a category with an initial object.*

Proof. The empty category is initial in the category of finitely presented freely generated categories. This corresponds to an ambient universal domain. Multiple atomic types like strings, integers, and booleans can be supported through a simple slice category construction. □

Lemma B.1.3 (Requirement 2)**.** *We have a functor* $Sen\colon \mathbf{Sig} \to \mathbf{Set}$ *such that* $Sen(Sig)$ *is the set of all equations over the signature* Sig*.*

Proof. For each $Sig \in \mathbf{Sig}$, we take the set of all finite path expressions $X.p_1.\ldots.p_n$, close them under composition, and then enumerate the set of equations between such terms. So the object part of $Sen(Sig)$ is the set of all path equations. For each $\phi\colon Sig_1 \to Sig_2$, we associate a function $Sig(\phi)\colon \mathbf{Set} \to \mathbf{Set} = \lambda eqs.\phi(eqs)$ that "applies" ϕ in the obvious manner, yielding a transformed set of equations. □

Lemma B.1.4 (Requirement 3)**.** *The assignment $Sig \mapsto Db(Sig)$ is functorial. That is, for each morphism of signatures $F\colon Sig \to Sig'$ there is an induced morphism $Db(Sig) \to Db(Sig')$ in $\mathbf{Cat}^{\mathrm{op}}$, i.e. a functor $Db(F)\colon Db(Sig') \to Db(Sig)$. We write $Db\colon \mathbf{Sig} \to \mathbf{Cat}^{\mathrm{op}}$, or as is slightly easier to parse,*

$$Db\colon \mathbf{Sig}^{\mathrm{op}} \to \mathbf{Cat}.$$

Proof. Let $F\colon Sig \to Sig'$ be a functor between free categories. Given an object in $Db(Sig')$, i.e. a functor $d\colon Sig' \to \mathbf{Set}$ define $Db(F) = d \circ F\colon Sig \to \mathbf{Set}$. Given a morphism in $Db(Sig')$, i.e. a natural transformation, compose it with the identity natural transformation on F to get a morphism in $Db(Sig)$. One checks easily that with these definitions, Db is indeed a functor. □

Lemma B.1.5 (Requirement 4)**.** *For each schema signature morphism $\phi\colon Sig_A \to Sig_B$, the following statement holds for each Sig_B instance d_B and each path equation $e \in Sen(Sig_A)$:*

$$d_B \models_{Sig_B} Sen(\phi)(e) \qquad \textit{iff} \qquad Db(\phi)(d_B) \models_{Sig_A} e$$

Proof. We are given an instance d_B on B. By definition, for any table X in A, we have $Db(\phi)(d_B)(X) = Db(\phi(X))$ and for any path $p\colon X \to Y$ in A and row $x \in Db(\phi(X))$, we have

$$Db(\phi)(d_B)(p)(x) = d_B(\phi(p))(x) \tag{6}$$

Suppose e equates paths $p, q\colon X \to Y$ for tables X, Y in A. By definition $d_B \models_{Sig_B} Sen(\phi)(e)$ iff for all $x \in d_B(\phi(X))$ we have $d_B(\phi(p))(x) = d_B(\phi(q))(x)$. By (6), this is the case iff for all $x \in Db(\phi)(d_B)$ we have $Db(\phi)(d_B)(p)(x) = Db(\phi)(d_B)(q)(x)$, which is by definition what is needed, $Db(\phi)(d_B) \models_{Sig_A} e$.

□

C Basic proofs

In this section, we fill in those leftover proofs from the main text that do not require much category theory.

Theorem C.1.6. *Let S be a typed signature, and let $[S]$ be its image as a relational schema. Then every S-instance is an $[S]$-instance, and every $[S]$-instance is an S-instance, up to natural isomorphism.*

Proof. A functor $[S] \to \mathbf{Set}$ assigns to each object a set and to each arrow a function, such that composition is preserved. There is a one-to-one correspondence between sets N and identity tables as in (2) and a one-to-one correspondence between functions $N_1 \to N_2$ and link tables as in (3), up to natural isomorphism. Finally, these sets and functions preserve composition if and only if they satisfy the path equivalence axioms. □

The condition "up to natural isomorphism" is necessary because FQL instances require globally unique IDs. For example, consider a category C with two objects X, Y and an arrow $f : X \to Y$. The functor F such that $F(X) := \{1, 2\}$, $F(Y) := \{2, 3\}$, $F(f) = \{(1, 2), (2, 3)\}$ cannot be directly represented as an FQL instance because 2 appears in the table for $F(X)$ and the table for $F(Y)$. However, the naturally isomorphic functor $F'(X) = \{1, 2\}, F'(Y) = \{a, b\}, F'(f) = \{(1, a), (2, b)\}$ can be represented in FQL.

D Categorical constructions for data migration

Here we give many standard definitions from category theory and a few less-than-standard (or original) results. The proofs often assume more knowledge than the definitions do. Let us also note our use of the term *essential*, which basically means "up to isomorphism". Thus an object X having a certain property is *essentially unique* if every other object having that property is isomorphic to X; an object X is in the *essential image* of some functor if it is isomorphic to an object in the image of that functor; etc.

D.1 Basic constructions

Definition D.1.1 (Fiber products of sets)**.** Suppose given the diagram of sets and functions as to the left in (7).

$$\begin{array}{ccc} & & B \\ & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \qquad\qquad \begin{array}{ccc} A \times_C B & \xrightarrow{f'} & B \\ {\scriptstyle g'}\downarrow & \searrow^{h} & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \tag{7}$$

Its *fiber product* is the commutative diagram to the right in (7), where we define

$$A \times_C B := \{(a, c, b) \mid f(a) = c = g(b)\}$$

and f', g', and $h = g \circ f' = f \circ g'$ are the obvious projections (e.g. $f'(a, c, b) = b$). We sometimes refer to just $(A \times_C B, f', g', h)$ or even to $A \times_C B$ as the fiber product.

Example D.1.2 (Chain signatures)*.* Let $n \in \mathbb{N}$ be a natural number. The *chain signature on n arrows*, denoted $\overrightarrow{n}$, [1] is the graph

$$\overset{0}{\bullet} \longrightarrow \overset{1}{\bullet} \longrightarrow \cdots \longrightarrow \overset{n}{\bullet}$$

with no path equivalences. One can check that $\overrightarrow{n}$ has $\binom{n+2}{2}$-many paths.

The chain signature $\overrightarrow{0}$ is the terminal object in the category of signatures.

A signature morphism $c\colon \overrightarrow{0} \to C$ can be identified with an object in C. A signature morphism $f\colon \overrightarrow{1} \to C$ can be identified with a morphism in C. It is determined up to path equivalence by a pair (n, p) where $n \in \mathbb{N}$ is a natural number and $p\colon \overrightarrow{n} \to C$ is a morphism of graphs.

[1] The chain signature $\overrightarrow{n}$ is often denoted $[n]$ in the literature, but we did not want to further overload the bracket [-] notation

Definition D.1.3 (Fiber product of signatures)**.** Suppose given the diagram of signatures and signatures mappings as to the left in (8).

$$\begin{array}{ccc} & & B \\ & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \qquad\qquad \begin{array}{ccc} A \times_C B & \xrightarrow{f'} & B \\ {\scriptstyle g'}\downarrow & \searrow^{h} & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \tag{8}$$

On objects and arrows we have the following diagrams of sets:

$$\begin{array}{ccc} & & \mathrm{Ob}(B) \\ & & \downarrow{\scriptstyle \mathrm{Ob}(g)} \\ \mathrm{Ob}(A) & \xrightarrow[\mathrm{Ob}(f)]{} & \mathrm{Ob}(C) \end{array} \qquad\qquad \begin{array}{ccc} & & \mathrm{Arr}(B) \\ & & \downarrow{\scriptstyle \mathrm{Arr}(g)} \\ \mathrm{Arr}(A) & \xrightarrow[\mathrm{Arr}(f)]{} & \mathrm{Mor}(C) \end{array} \tag{9}$$

We define the *fiber product* of the left-hand diagram in (8) to be the right-hand commutative diagram in (8), where we the objects and arrows of $A \times_C B$ are given by the fiber product of the left and right diagram of sets in (9), respectively. A pair of paths in $A \times_C B$ are equivalent if they map to equivalent paths in both A and B (and therefore in C). Note that we have given $A \times_C B$ as a signature, i.e. we have provided a finite presentation of it.

Example D.1.4 (Pre-image of an object or morphism). Let $F\colon C \to D$ be a functor. Given an object $d \in \mathrm{Ob}(D)$ (respectively a morphism $d \in \mathrm{Mor}(D)$), we can regard it as a functor $d\colon \overrightarrow{0} \to D$ (respectively a functor $d\colon \overrightarrow{1} \to D$), as in Example D.1.2. The pre-image of d is the subcategory of C that maps entirely into d. This is the fiber product of the diagram $C \xrightarrow{F} D \xleftarrow{d} \overrightarrow{0}$ (respectively the diagram $C \xrightarrow{F} D \xleftarrow{d} \overrightarrow{1}$).

A signature is assumed to have finitely many arrows, but because it may have loops, C may have infinitely many non-equivalent paths. We make the following definitions to deal with this.

Definition D.1.5. A signature C is called *finite* if there is a finite set S of paths such that every path in C is equivalent to some $s \in S$. An equivalent condition is that the schema $\llbracket C \rrbracket$ has finitely many morphisms. (Note that if C is finite then C has finitely many objects too, since each object has an associated trivial path.)

Recall (from the top of Section 2) that every signature is assumed to have a finite set of objects and arrows. The above notion is a semantic one: a signature C is called finite if $\llbracket C \rrbracket$ is a finite category.

Definition D.1.6. A signature C is called *acyclic* if it has no non-trivial loops. In other words, for all objects $c \in \mathrm{Ob}(C)$ every path $p\colon c \to c$ is a trivial path.

Lemma D.1.7 (Acyclicity implies finiteness)**.** *If C is an acyclic signature then it is finite.*

Proof. All of our signatures are assumed to include only finitely many objects and arrows. If C is acyclic then no edge can appear twice in the same path. Suppose that C has n arrows. Then the number of paths in C is bounded by the number of lists in n letters of length at most n. This is finite.

□

Lemma D.1.8 (Acyclicity and finiteness are preserved under formation of fiber product)**.** *Given a diagram $A \xrightarrow{F} C \xleftarrow{G} B$ of signatures such that each is acyclic (respectively, finite), the fiber product signature $A \times_C B$ is acyclic (respectively, finite).*

Proof. The fiber product of finite sets is clearly finite, and the fiber product of categories is computed by twice taking the fiber products of sets (see Definition D.1.3). We now show that acyclicity is preserved.

Let $\mathcal{L}oop$ denote the signature

$$\mathcal{L}oop := \boxed{\begin{array}{c} f \\ \circlearrowright \\ s \bullet \end{array}} \tag{10}$$

A category C is acyclic if and only if every functor $L\colon \mathcal{L}oop \to C$ factors through the unique functor $\mathcal{L}oop \to \overrightarrow{0}$, i.e. if and only if every loop is a trivial path. The result follows by the universal property for fiber products.

□

Definition D.1.9 (Comma category)**.** To a diagram of categories $A \xrightarrow{f} C \xleftarrow{g} B$ we can associate the *comma category of f over g*, denoted $(f \downarrow g)$, defined as follows.

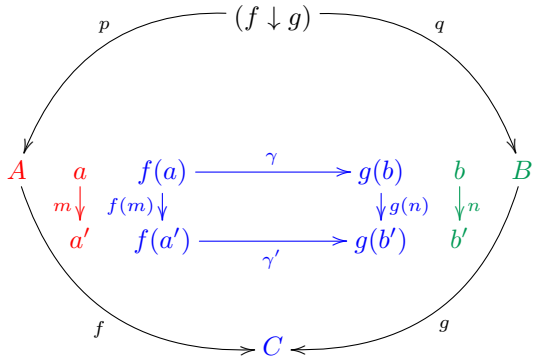
$$\mathrm{Ob}(f \downarrow g) := \{(a, b, \gamma) \mid a \in \mathrm{Ob}(A), b \in \mathrm{Ob}(B), \gamma\colon f(a) \to g(b) \text{ in } C\}$$

$$\mathrm{Hom}_{(f\downarrow g)}((a,b,\gamma),(a',b',\gamma')) := \{(m,n) \mid m\colon a \to a' \text{ in } A, n\colon b \to b' \text{ in } B, \gamma' \circ f(m) = g(n) \circ \gamma\}$$

There are obvious *projection* functors $(f \downarrow g) \xrightarrow{p} A$ and $(f \downarrow g) \xrightarrow{q} B$ and a natural transformation $\alpha\colon f \circ p \to g \circ q$, and we summarize this in a canonical natural transformation diagram

$$\begin{array}{ccc} (f \downarrow g) & \xrightarrow{q} & B \\ {\scriptstyle p}\downarrow & \overset{\alpha}{\swarrow} & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array} \tag{11}$$

Below is a heuristic picture that may be helpful. The outside square consists of categories and functors (note that it does not commute!). The inside square represents a morphism in $(f \downarrow g)$ from object (a, b, γ) to object (a', b', γ'); this diagram is required to commute in C.



Example D.1.10 (Slice category and coslice category). Let $f\colon A \to B$ be a functor, and let $b \in \mathrm{Ob}(B)$ be an object, which we represent as a functor $b\colon \overrightarrow{0} \to B$. The *slice category of f over b* is defined as $(f \downarrow b)$. The *coslice category of b over F* is defined as $(b \downarrow f)$.

For example let $\{*\}$ denote a one element set, and let $\mathrm{id}_{\mathbf{Set}}$ be the identity functor on **Set**. Then the coslice category $(\{*\} \downarrow \mathrm{id}_{\mathbf{Set}})$ is the category of pointed sets, which we denote by $\mathbf{Set}_*$. The slice category $(\mathrm{id}_{\mathbf{Set}} \downarrow \{*\})$ is isomorphic to **Set**.

Lemma D.1.11 (Comma categories as fiber products)**.** *Given a diagram $A \xrightarrow{f} C \xleftarrow{g} B$, the comma category $(f \downarrow g)$ can be obtained as the fiber product in the diagram*

$$\begin{array}{ccc} (f \downarrow g) & \xrightarrow{\alpha} & C^{\overrightarrow{1}} \\ {\scriptstyle (p,q)}\downarrow & & \downarrow{\scriptstyle (dom,cod)} \\ A \times B & \xrightarrow[(f,g)]{} & C \times C \end{array}$$

where $C^{\overrightarrow{1}}$ is the category of functors $\overrightarrow{1} \to C$.

Proof. Straightforward. □

Lemma D.1.12 (Acyclicity and finiteness are preserved under formation of comma category)**.** *Let A, B, and C be acyclic (resp. finite) categories, and suppose we have functors $A \xrightarrow{f} C \xleftarrow{g} B$. Then the comma category $(f \downarrow g)$ is acyclic (resp. finite).*

Proof. Since the fiber product of finite sets is finite, Lemma D.1.11 implies that the formation of comma categories preserves finiteness. Suppose that A and B are acyclic (in fact this is enough to imply that $(f \downarrow g)$ is acyclic). Given a functor $L\colon \mathcal{L}oop \to (f \downarrow g)$, this implies that composing with the projections to A and B yields trivial loops. In other words L consists of a diagram in C of the form

$$\begin{array}{ccc} f(a) & \xrightarrow{\gamma} & g(b) \\ \| & & \| \\ f(a) & \xrightarrow[\gamma']{} & g(b) \end{array}$$

But this implies that $\gamma = \gamma'$, so L is a trivial loop too, competing the proof. □

D.2 Data migration functors

Construction D.2.1 (Δ)**.** Let $F\colon C \to D$ be a functor. We will define a functor $\Delta_F\colon D\text{–}\mathbf{Inst} \to C\text{–}\mathbf{Inst}$; that is, given a D-instance $I\colon D \to \mathbf{Set}$ we will naturally construct a C-instance $\Delta_F(I)$. Mathematically, this is obtained simply by composing the functors to get

$$\Delta_F(I) := I \circ F\colon C \to \mathbf{Set} \qquad \begin{array}{ccccc} C & \xrightarrow{F} & D & \xrightarrow{I} & \mathbf{Set} \\ & \searrow & & \nearrow & \\ & & {\scriptstyle \Delta_F I} & & \end{array}$$

To understand this on the level of binary tables, we take F and I as given and proceed to define the C-instance $J := \Delta_F I$ as follows. Given an object $c \in \mathrm{Ob}(C)$, it is sent to an object $d := F(c) \in \mathrm{Ob}(D)$, and $I(d)$ is an entity table. Assign $J(c) = I(d)$. Given an arrow $g\colon c \to c'$ in C, it is sent to a path $p = F(g)$ in D. We can compose this to a binary table $[p]$. Assign $J(g) = [p]$. It is easy to check that these assignments preserve the axioms.

Lemma D.2.2 (Finiteness is preserved under Δ)**.** *Suppose that C and D are finite categories and $F\colon C \to D$ is a functor. If $I\colon D \to$ **Set** is a finite D-instance then $\Delta_F I$ is a finite C-instance.*

Proof. This follows by construction: every object and arrow in C is assigned some finite join of the tables associated to objects and arrows in D, and finite joins of finite tables are finite.

□

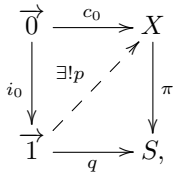
Recall the definition of discrete op-fibrations from Definition 2.

Lemma D.2.3 (Discrete op-fibrations)**.** *Let $F\colon C \to D$ be a signature morphism. The following are conditions on F are equivalent:*

1. *The functor $[F]\colon [C] \to [D]$ is a discrete op-fibration.*
2. *For every choice of object c_0 in C and path $q\colon d_0 \to d_n$ in D with $F(c_0) = d_0$, there exists a path $p \in C$ such that $F(p) \sim q$, and p is unique up to path equivalence.*
3. *For every choice of object c_0 in C and edge $q\colon d_0 \to d_1$ in D with $F(c_0) = d_0$, there exists a path $p \in C$ such that $F(p) \sim q$, and p is unique up to path equivalence.*

Proof. It is clear that condition (2) implies condition (3), and the converse is true because concatenations of equivalent paths are equivalent. It suffices to show that (1) and (2) are equivalent.

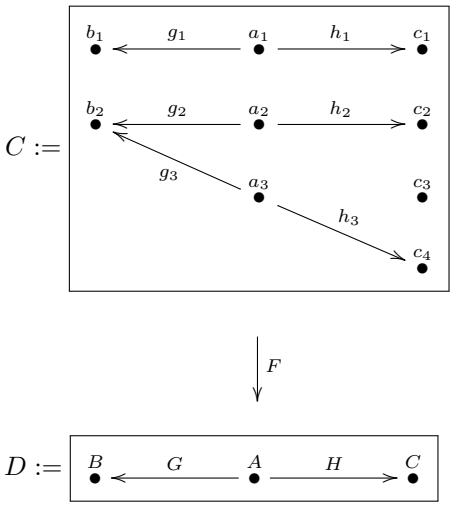
It is shown in [25] that a functor $\pi\colon X \to S$ between categories X and S is a discrete op-fibration if and only if for each solid-arrow square of the form



where $i_0\colon \overrightarrow{0} \to \overrightarrow{1}$ sends the unique object of $\overrightarrow{0}$ to the initial object of $\overrightarrow{1}$, there exists a unique functor ℓ such that both triangles commute. We now translate this statement to the language of signatures with $[\![C]\!] = X$ and $[\![D]\!] = S$. A functor $\overrightarrow{1} \to [\![C]\!]$ corresponds to a path in C, and the uniqueness of lift p corresponds to uniqueness of equivalence class. This completes the proof.

□

Example D.2.4 (A discrete op-fibration). Let C, D be signatures, and let $F\colon C \to D$ be as suggested by the picture below.



This is a discrete op-fibration.

Corollary D.2.5 (Pre-image of an object under a discrete op-fibration is discrete)**.** *Let $F\colon C \to D$ be a discrete op-fibration and let $d \in \mathrm{Ob}(D)$ be an object. Then the pre-image $F^{-1}(d)$ is a discrete signature (i.e. every path in $F^{-1}(d)$ is equivalent to a trivial path).*

Proof. For any object $d \in \mathrm{Ob}(D)$ the pre-image $F^{-1}(d)$ is either empty or not. If it is empty, then it is discrete. If $F^{-1}(d)$ is non-empty, let $e\colon c_0 \to c_1$ be an edge in it. Then $F(e) = d = F(c_0)$ so we have an equivalence $e \sim c_0$ by Lemma D.2.3.

□

Example D.2.6. Let $F\colon A \to B$ be a functor. For any object $b \in B$, considered as a functor $b\colon \overrightarrow{0} \to B$, the induced functor $(b \downarrow F) \to B$ is a discrete op-fibration. Indeed, given an object $b \xrightarrow{g} F(a)$ in $(b \downarrow F)$ and a morphism $h\colon a \to a'$ in A, there is a unique map $b \xrightarrow{g} F(a) \xrightarrow{F(h)} F(a')$ over it.

Definition D.2.7. A signature C is said to *have non-redundant edges* if it satisfies the following condition for every edge e and path p in C: If $e \sim p$, then p has length 1 and $e = p$.

Proposition D.2.8. *Every acyclic signature is equivalent to a signature with non-redundant edges.*

Proof. The important observation is that if C is acyclic then for any edge e and path p in C, if e is an edge in p and $e \sim p$ then $e = p$. Thus, we know that if C is acyclic and $e \sim p$ then e is not in p. So enumerate the edges of C as $E_C := \{e_1, \ldots, e_n\}$. If e_1 is equivalent to a path in $E_C - \{e_1\}$ then there is an equivalence of signatures $C \xrightarrow{\cong} C - \{e_1\} =: C_1$; if not, let $C_1 := C$. Proceed by induction to remove each e_i that is equivalent to a path in C_{i-1}, and at the end no edge will be redundant.

□

Corollary D.2.9 (Discrete op-fibrations and preservation of path length)**.** *If C and D have non-redundant edges and $F\colon C \to D$ is a discrete op-fibration, then for every choice of object c_0 in C and path $q = d_0.e_1'.e_2'.\ldots.e_n'$ of length n in D with $F(c_0) = d_0$, there exists a unique path $p = c_0.e_1.e_2.\ldots.e_n$ of length n in C such that $F(e_i) = e_i'$ for each $1 \leq i \leq n$, so in particular $F(p) = q$.*

Proof. Let c_0, d_0, and q be as in the hypothesis. We proceed by induction on n, the length of q. In the base case $n = 0$ then Corollary D.2.5 implies that every edge in $F^{-1}(d_0)$ is equivalent to the trivial path c_0, so the result follows by the non-redundancy of edges in C. Suppose the result holds for some $n \in \mathbb{N}$. To prove the result for $n+1$ it suffices to consider the final edge of q, i.e. we assume that $q = e'$ is simply an edge. By Lemma D.2.3 there exists a path $p \in C$ such that $F(p) \sim e'$, so by non-redundancy in D we know that $F(p)$ has length 1. This implies that for precisely one edge e_i in p we have $F(e_i) = e'$, and for all other edges e_j in p we have $F(e_j)$ is a trivial path. But by the base case this implies that p has length 1, completing the proof.

□

Construction D.2.10 (Σ for discrete opfibrations)**.** Suppose that $F\colon C \to D$ is a discrete op-fibration. We can succinctly define $\Sigma_F\colon C\text{–}\mathbf{Inst} \to D\text{–}\mathbf{Inst}$ to be the left adjoint to Δ_F, however the formula has a simple description which we give now. Suppose we are given F and an instance $I\colon C \to \mathbf{Set}$, considered as a collection of binary tables, one for each object and each edge in C. We are tasked with finding a D-instance, $J := \Sigma_F I\colon D \to \mathbf{Set}$.

We first define J on an arbitrary object $d \in \mathrm{Ob}(D)$. By Corollary D.2.5, the pre-image $F^{-1}(d)$ is discrete in C; that is, it is equivalent to a finite collection of object tables. We define $J(d)$ to be the disjoint union

$$J(d) := \coprod_{c \in F^{-1}(d)} I(c).$$

Similarly, let $e\colon d \to d'$ be an arbitrary arrow in D. By Lemma D.2.3 we know that for each $c \in F^{-1}(d)$ there is a unique equivalence class of paths p_c in C such that $F(p_c) \sim e$. Choose one, and compose it to a single binary table — all other choices will result in the same result. Then define $J(e)$ to be the disjoint union

$$J(e) := \coprod_{c \in F^{-1}(d)} I(p_c).$$

Lemma D.2.11 (Finiteness is preserved under Σ)**.** *Suppose that C and D are finite categories and $F\colon C \to D$ is a functor. If $I\colon C \to \mathbf{Set}$ is a finite C-instance then $\Sigma_F I$ is a finite D instance.*

Proof. For each object or arrow d in D, the table $\Sigma_F I(d)$ is a finite disjoint union of composition-joins of tables in C. The finite join of finite tables is finite, and the finite union of finite tables is finite.

□

Remark D.2.12 (Σ exists more generally and performs quotients and skolemization)*.* For any functor $F\colon C \to D$ the functor $\Delta_F\colon D\text{–}\mathbf{Inst} \to C\text{–}\mathbf{Inst}$ has a left adjoint, which we can denote by $\Sigma_F\colon C\text{–}\mathbf{Inst} \to D\text{–}\mathbf{Inst}$ because it agrees with the Σ_F constructed in D.2.10 in the case that F is a discrete op-fibration. Certain queries are possible if we

can use Σ_F in this more general case—namely, quotienting by equivalence relations and the introduction of labeled nulls (a.k.a. Skolem variables). We do not consider it much in this paper for a few reasons. First, quotients and skolem variables are not part of the relational algebra. Second, the set of queries that include such quotients and skolem variables are not obviously closed under composition (see Section 9).

Construction D.2.13 (Limit as a kind of "join all")**.** Let B be a signature and let H be a B-instance. The functor $[H]\colon [B] \to \mathbf{Set}$ has a limit $\lim_B H \in \mathrm{Ob}(\mathbf{Set})$, which can be computed as follows. Forgetting the path equivalence relations, axioms (2) and (3) imply that H consists of a set $\{N_1, \ldots, N_m\}$ of node tables, a set $\{e_1, \ldots, e_n\}$ of edge tables, and functions $s, t\colon \{1, \ldots, n\} \to \{1, \ldots, m\}$ such that for each $1 \leq i \leq n$ the table e_i constitutes a function $e_i\colon N_{s(i)} \to N_{t(i)}$. For each i define X_i as follows:

$$\begin{aligned}
X_0 &:= \pi_{2,4,\ldots,2m}(N_1 \times \cdots \times N_m) \\
X_1 &:= \pi_{1,2,\ldots m}\sigma_{s(1)=m+1}\sigma_{t(1)=m+2}(X_0 \times e_1) \\
&\vdots \\
X_i &:= \pi_{1,2,\ldots m}\sigma_{s(i)=m+1}\sigma_{t(i)=m+2}(X_{i-1} \times e_i) \\
&\vdots \\
X_n &:= \pi_{1,2,\ldots m}\sigma_{s(n)=m+1}\sigma_{t(n)=m+2}(X_{n-1} \times e_n)
\end{aligned}$$

Then $\overline{X} := X_n$ is a table with m columns, and its set $|\overline{X}|$ of rows (which can be constructed if one wishes by concatenating the fields in each row) is the limit, $|\overline{X}| \cong \lim_B H$.

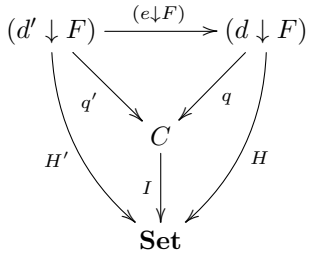
Construction D.2.14 (Π)**.** Let $F\colon C \to D$ be a signature morphism. We can succinctly define $\Pi_F\colon C\text{–}\mathbf{Inst} \to D\text{–}\mathbf{Inst}$ to be the right adjoint to Δ_F, however the formula has an algorithmic description which we give now. Suppose we are given F and an instance $I\colon C \to \mathbf{Set}$, considered as a collection of binary tables, one for each object and each edge in C. We are tasked with finding a D-instance, $J := \Pi_F I\colon D \to \mathbf{Set}$.

We first define J on an arbitrary object $d\colon \overrightarrow{0} \to D$ (see Example D.1.2). Consider the comma category $B_d := (d \downarrow F)$ and its projection $q_d\colon (d \downarrow F) \to C$. Note that in case C and D are finite, so is B_d. Let $H_d := I \circ q_d\colon B_d \to \mathbf{Set}$. We define $J(d) := \lim_{B_d} H_d$ as in Construction D.2.13.

Now let $e\colon d \to d'$ be an arbitrary arrow in D. For ease of notation, rewrite

$$B := B_d, \quad B' := B_{d'}, \quad q := q_d, \quad q' := q_{d'}, \quad H := H_d, \text{ and } H' := H_{d'}.$$

We have the following diagram of categories



A unique natural map $J(d) = \lim_B H \to \lim_B H' = J(d')$ is determined by the universal property for limits, but we give an idea of its construction. There is a map from the set

of nodes in B' to the set of nodes in B and for each node N in B' the corresponding node tables in H and H' agree. Let X_i and X'_i be defined respectively for (B, H) and (B', H') as in Construction D.2.13. It follows that X'_0 is a projection (or column duplication) on X_0. The set of edges in B' also map to the set of edges in B and for each edge e in B' the corresponding edge tables in H and H' agree. Thus the select statements done to obtain $J(d') = \overline{X'}$ contains the set of select statements performed to obtain $J(d') = \overline{X}$. Therefore, the map from $J(d)$ to $J(d')$ is given as the inclusion of a subset followed by a projection.

If C or D is not finite, then the right pushforward Π_F of a finite instance $I \in C$–**Set** may have infinite, even uncountable results.

Example D.2.15. Consider the unique signature morphism

$$\mathcal{C} = \boxed{s \bullet} \xrightarrow{F} \boxed{f \circlearrowright s \bullet} =: \mathcal{D}.$$

Here $[\![\mathcal{D}]\!]$ has arrows $\{f^n \mid n \in \mathbb{N}\}$ so it is infinite. In this case $(d \downarrow F)$ is the discrete category with a countably infinite set of objects $\mathrm{Ob}(d \downarrow F) \cong \mathbb{N}$.

Given the two-element instance $I \colon \mathcal{C} \to \mathbf{Set}$ with $I(s) = \{\mathrm{Alice}, \mathrm{Bob}\}$, the rowset in the right pushforward $\Pi_F(I)$ is the (uncountable) set of infinite streams in $\{\mathrm{Alice}, \mathrm{Bob}\}$, i.e.

$$\Pi_F(I)(s) = I(s)^{\mathbb{N}}.$$

Proposition D.2.16. *Let $H \colon B \to \mathbf{Set}$ be a functor, and let $q \colon B \to \overrightarrow{0}$ be the terminal functor. Noting that there is an isomorphism of categories $\overrightarrow{0}$–**Set** $\cong$ **Set**, we have a bijection $\lim_B H \cong \Pi_q H$.*

Proof. Obvious by construction of Π.

□

Proposition D.2.17 (Behavior of Δ, Σ, Π under natural transformations)**.** *Let C and D be categories, let $F, G \colon C \to D$ be functors, and let $\alpha \colon F \to G$ be a natural transformation as depicted in the following diagram:*

$$C \;\underset{G}{\overset{F}{\Downarrow \alpha}}\; D$$

Then α induces natural transformations

$$\Delta_\alpha \colon \Delta_F \to \Delta_G, \qquad \Sigma_\alpha \colon \Sigma_G \to \Sigma_F, \qquad \text{and} \qquad \Pi_\alpha \colon \Pi_G \to \Pi_F.$$

Proof. For any instance $J \colon D \to \mathbf{Set}$ and any object $c \in \mathrm{Ob}(C)$ we have $\alpha_c \colon J \circ F(c) \to J \circ G(c)$, and the naturality of α implies that we can gather these into a natural transformation $\Delta_\alpha(J) \colon \Delta_F(J) \to \Delta_G(J)$. One checks easily that this assignment is natural in J, so we have $\Delta_\alpha \colon \Delta_F \to \Delta_G$ as desired.

Now suppose that $I \colon C \to \mathbf{Set}$ is an instance on C. Then for any $J \in D$–**Set** we have natural maps

$$\mathrm{Hom}(J, \Pi_G I) \cong \mathrm{Hom}(\Delta_G J, I) \xrightarrow{\Delta_\alpha} \mathrm{Hom}(\Delta_F J, I) \cong \mathrm{Hom}(J, \Pi_F I)$$

so by the Yoneda lemma, we have a natural map $\Pi_G \to \Pi_F$, as desired. We also have natural maps

$$\mathrm{Hom}(\Sigma_F I, J) \cong \mathrm{Hom}(I, \Delta_F J) \xrightarrow{\Delta_\alpha} \mathrm{Hom}(I, \Delta_G J) \cong \mathrm{Hom}(\Sigma_G I, J)$$

so by the Yoneda lemma, we have a natural map $\Sigma_G \to \Sigma_F$, as desired.

□

D.3 Relation to SQL queries

Suppose we are working in a domain $DOM \in \mathrm{Ob}(\mathbf{Set})$.

Definition D.3.1. A *set-theoretic SQL query* q is an expression of the form

SELECT DISTINCT	P
FROM	$(c_{1,1}, \ldots, c_{1,k_1}), \ldots, (c_{n,1}, \ldots, c_{n,k_n})$
WHERE	W

where $n, k_1, \ldots, k_n \in \mathbb{N}$ are natural numbers, C is a set of the form $C = \coprod_i \{c_{i,1}, \ldots, c_{i,k_i}\}$, W is a set of pairs $W \subseteq C \times C$, and $P\colon P_0 \to C$ is a function, for some set P_0.

A *bag-theoretic SQL query* q' is an expression of the form

SELECT	P
FROM	$(c_{1,1}, \ldots, c_{1,k_1}), \ldots, (c_{n,1}, \ldots, c_{n,k_n})$
WHERE	W

where $n, k_1, \ldots, k_n, C, W, P, P_0$ are as above. We call $(n, k_1, \ldots, k_n, C, W, P, P_0)$ an *agnostic SQL query*.

Suppose that for each $1 \le i \le n$ we are given a relation $R_i \subseteq \prod_{1 \le j \le k_i} DOM$.

The *evaluation of* q (respectively, the *evaluation of* q') on $R_1, \ldots, R_n$ is a relation $Q(R_1, \ldots, R_n) \subseteq \prod_{i \in P_0} R_i$ (respectively, a function $Q'(R_1, \ldots, R_n)\colon B' \to \prod_{i \in P_0} R_i$ for some set B'), defined as follows. Let

$$B' = \{r \in R_1 \times \cdots \times R_n \mid r.c_1 = r.c_2 \text{ for all } (c_1, c_2) \in W\}.$$

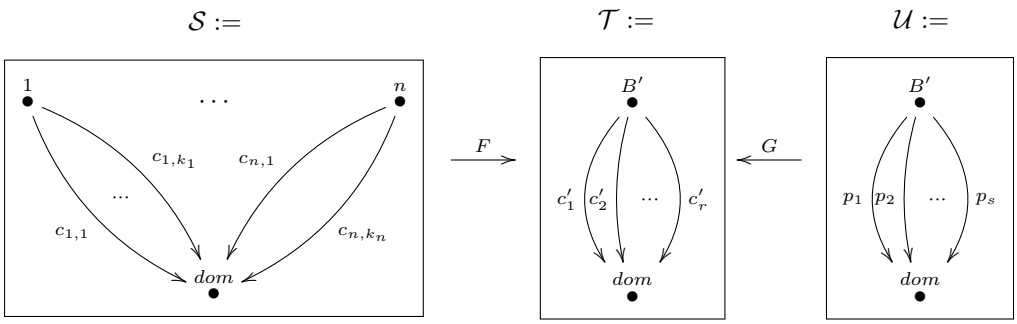
We compose an inclusion with a projection to get a function

$$B' \subseteq R_1 \times \cdots \times R_n \xrightarrow{P} \prod_{i \in P_0} R_i.$$

and define $Q'(R_1, \ldots, R_n)$ to be this function. Its image is the desired relation $Q(R_1, \ldots, R_n) \subseteq \prod_{i \in P_0} R_i$.

Definition D.3.2. Let $q = (n, k_1, \ldots, k_n, C, W, P, P_0)$ be an agnostic SQL query as in Definition D.3.1. The set W generates an equivalence relation $\sim$ on C; let $C' := C/\sim$ be the quotient and $\ell\colon C \to C'$ the induced function. Suppose that C' has $r = |C'|$ many elements and that P_0 has $s = |P_0|$ many elements.

We define the *categorical setup for* q, denoted $CS(q) = \mathcal{S} \xrightarrow{F} \mathcal{T} \xleftarrow{G} \mathcal{U}$ as follows.



Here, the functor F sends *dom* to *dom*, sends the other objects to B, and acts on morphisms by $\ell\colon C \to C'$; the functor G sends *dom* to *dom*, B' to B', and $P_0 \to C'$ by the composite $P_0 \xrightarrow{P} C \xrightarrow{\ell} C'$.

We define the *categorical analogue of* q, denoted $CA(q)$, to be the FQL query $\Delta_G \Pi_F$.[2]

Proposition D.3.3. *Let $q' = (n, k_1, \dots, k_n, C, W, P, P_0)$ be an agnostic SQL query, let $Q(R_1, \dots, R_n)$ be its set-theoretic evaluation, and let $Q'(R_1, \dots, R_n)$ be its bag-theoretic evaluation as in Definition D.3.1. Let $\mathcal{S} \xrightarrow{F} \mathcal{T} \xleftarrow{G} \mathcal{U}$ be the categorical setup for q. Suppose that for each $1 \le i \le n$ we are given a relation $R_i \subseteq \prod_{1 \le j \le k_i} DOM$. Let $I\colon \mathcal{S} \to$ **Set** be the corresponding functor (i.e., for each $1 \le i \le n$ let $I(i) = R_i$, let $I(dom) = DOM$, and let $c_{i,j}\colon R_i \to DOM$ be the projection).*

Then the categorical analogue $CA(q)(I) = \Delta_G \Pi_F(I)$ is equal to the bag-theoretic evaluation $Q'(R_1, \dots, R_n)$, and the relationalization $REL(CA(q)(I))$ (see Definiton E.1.6) is equal to the set-theoretic evaluation $Q(R_1, \dots, R_n)$.

Proof. The first claim follows from Construction D.2.14. The second claim follows directly, since both sides are simply images.

□

D.4 Query composition

In this section and those that follow we were greatly inspired by, and follow closely, the work of [12]. While their work does not apply directly, the adaptation to our context is fairly straightforward.

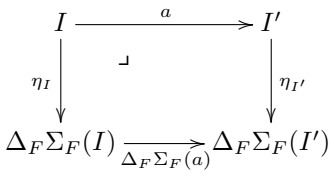
Lemma D.4.1 (Unit and counit for Σ are Cartesian)**.** *Let $F\colon C \to D$ be a discrete op-fibration, and let*

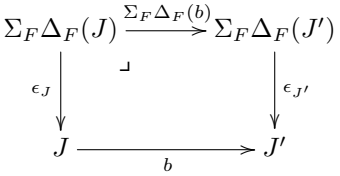
$$\eta\colon \mathrm{id}_{C\text{-}\mathbf{Inst}} \to \Delta_F \Sigma_F \qquad \textit{and} \qquad \epsilon\colon \Sigma_F \Delta_F \to \mathrm{id}_{D\text{-}\mathbf{Inst}}$$

be (respectively) the unit and counit of the (Σ_F, Δ_F) adjunction. Each is a Cartesian natural transformation. In other words, for any morphism $a\colon I \to I'$ of C-instances and for any morphism $b\colon J \to J'$ of D-instances, each of the induced naturality squares (left

[2]The functor $\Delta_G \Pi_F$ is technically not in the correct $\Sigma\Pi\Delta$ order, but it is equivalent to a query in that order by Theorem D.4.22.

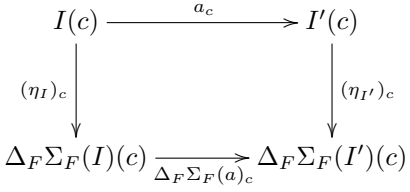
below for the unit and right below for the counit)

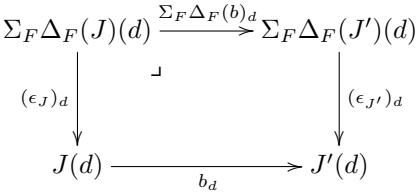




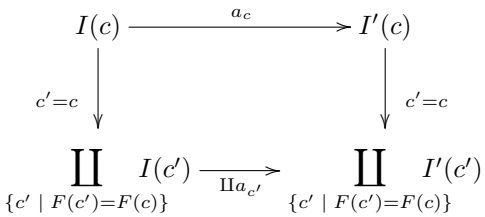
is a pullback in C*–***Inst** *and* D*–***Inst** *respectively.*

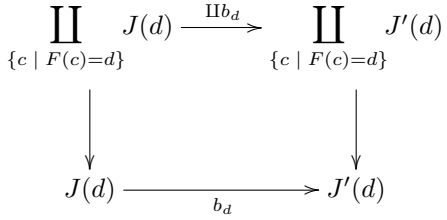
Proof. It suffices to check that for an arbitrary object $c \in \mathrm{Ob}(C)$ and $d \in \mathrm{Ob}(D)$ respectively, the induced commutative diagram in **Set**





is a pullback. Unpacking definitions, these are:





Roughly, the fact that these are pullbacks squares follows from the way coproducts work (e.g. the disjointness property) in **Set**, or more precisely it follows from the fact that **Set** is a positive coherent category [17, p. 34].

□

Definition D.4.2 (Grothendieck construction)**.** Let C be a signature and $I\colon C \to \mathbf{Set}$ an instance. The *Grothendieck category of elements for* I *over* C consists of a pair $(\int_C I, \pi_I)$, where $\int_C I$ is a signature and $\pi_I\colon \int_C I \to C$ is a signature morphism, constructed as follows. The set of nodes in $\int_C I$ is $\{(c,x) \mid c \in \mathrm{Ob}(C), x \in I(c)\}$. The set of edges in $\int_C I$ from node (c,x) to node (c',x') is $\{e\colon c \to c' \mid I(e)(x) = x'\}$. The signature morphism $\pi_I\colon \int_C I \to C$ is obvious: send (c,x) to c and send e to e. Two paths are equivalent in $\int_C I$ if and only if their images under π_I are equivalent. We sometimes denote $\int_C$ simply as $\int$.

Lemma D.4.3. *Let* $I\colon C \to \mathbf{Set}$ *be an instance. Then* $\pi_I\colon \int I \to C$ *is a discrete op-fibration.*

Proof. This follows by Lemma D.2.3.

□

Lemma D.4.4 (Acyclicity and finiteness are preserved under the Grothendieck construction)**.** *Suppose that C is an acyclic (respectively, a finite) signature and that $I\colon C \to \mathbf{Set}$ is a finite instance. Then $\int I$ is an acyclic (respectively, a finite) signature.*

Proof. Both are obvious by construction.

□

Definition D.4.5 (DeGrothendieckification)**.** Let $\pi\colon X \to C$ be a discrete op-fibration. Let $\{*\}^X$ denote a terminal object in X–**Inst**, i.e. any instance in which every node table and edge table consists of precisely one row. Define *the deGrothendieckification of* π, denoted $\partial\pi\colon C \to \mathbf{Set}$ to be $\partial\pi := \Sigma_\pi\left(\{*\}^X\right) \in C$–**Inst**.

One checks that for a discrete op-fibration $\pi\colon X \to C$ and object $c \in \mathrm{Ob}(C)$ we have the formula

$$\partial\pi(c) = \pi^{-1}(c),$$

so we can say that deGrothendieckification is given by pre-image.

Lemma D.4.6 (Finiteness is preserved under DeGrothendiekification)**.** *If X is a finite signature and $\pi\colon X \to C$ is any discrete op-fibration, then $\partial\pi$ is a finite C-instance.*

Proof. Obvious.

□

Proposition D.4.7. *Given a signature C, let $Dopf_C \subseteq \mathbf{Cat}_{/C}$ denote the full category spanned by the discrete op-fibrations over C. Then $\int\colon C$–**Inst** $\to Dopf_C$ and $\partial\colon Dopf_C \to C$–**Inst** are functorial, ∂ is left adjoint to $\int$, and they are mutually inverse equivalences of categories.*

Proof. See [25, Lemma 2.3.4, Proposition 3.2.5].

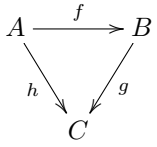
□

Corollary D.4.8. *Suppose that C and D are categories, that $F, G\colon C \to D$ are discrete op-fibrations, and $\alpha\colon F \to G$ is a natural transformation. Then there exists a functor $p\colon C \to C$ such that $F \circ p = G$, and $\Sigma_\alpha = \partial(p)$, where $\Sigma_\alpha\colon \Sigma_G \to \Sigma_F$ is the natural transformation given in Proposition D.2.17.*

Proof. By Proposition D.2.17 the natural transformation $\alpha\colon F \to G$ induces a natural transformation $\Sigma_G \to \Sigma_F$, and applying deGrothendieckification supplies a map $\partial G \to \partial F$. By Proposition D.4.7 this induces a map $p\colon C \to C$ over D with the above properties.

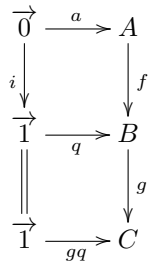
□

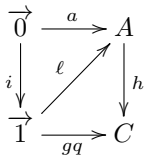
Proposition D.4.9. *Given a commutative diagram*

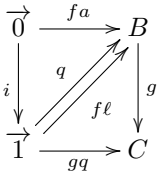


in which g and h are discrete op-fibrations, it follows that f is also a discrete op-fibration.

Proof. Consider the diagram to the left below



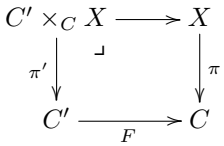




Then since h is a discrete op-fibration there exists a unique $\ell\colon \overrightarrow{1} \to A$ making the middle diagram commute. But now we have two lifts, q and $f\ell$, and since g is a discrete op-fibration, it follows that $q = f\ell$, completing the proof.

□

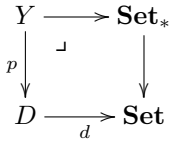
Proposition D.4.10 (Discrete op-fibrations are stable under pullback)**.** *Let $F\colon C' \to C$ be a functor and $\pi\colon X \to C$ be a discrete op-fibration. Then given the pullback square*



the map π' is a discrete op-fibration, and there is a natural isomorphism

$$\pi' \cong \int_{C'} \Delta_F(\partial\pi).$$

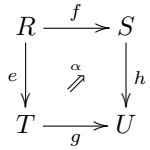
Proof. A functor $p\colon Y \to D$ is a discrete op-fibration if and only there exists a functor $d\colon D \to \mathbf{Set}$ such that the diagram



is a pullback square, where $\mathbf{Set}_*$ is the category of pointed sets as in Example D.1.10. The result follows from the pasting lemma for fiber products, and Lemma D.4.3.

□

Lemma D.4.11 (Comparison morphisms for squares)**.** *Suppose given the following diagram of categories:*



*Then there are natural transformations of functors S–**Inst** $\to T$–**Inst**:*

$$\Sigma_f\Delta_e \longrightarrow \Delta_h\Sigma_g \qquad \text{and} \qquad \Delta_g\Pi_h \longrightarrow \Pi_e\Delta_f.$$

*If $hf = ge$ and $\alpha = \mathrm{id}$ (i.e. if the diagram commutes) then, by symmetry, there are natural transformations of functors T–**Inst** $\to S$–**Inst**:*

$$\Sigma_e \Delta_f \longrightarrow \Delta_g \Sigma_h \qquad \text{and} \qquad \Delta_h \Pi_g \longrightarrow \Pi_f \Delta_e.$$

Proof. These arise from units and counits, together with Proposition D.2.17.

$$\Sigma_f \Delta_e \xrightarrow{\eta_g} \Sigma_f \Delta_e \Delta_g \Sigma_g \xrightarrow{\Delta_\alpha} \Sigma_f \Delta_f \Delta_h \Sigma_g \xrightarrow{\epsilon_f} \Delta_h \Sigma_g$$
$$\Delta_g \Pi_h \xrightarrow{\eta_e} \Pi_e \Delta_e \Delta_g \Pi_h \xrightarrow{\Delta_\alpha} \Pi_e \Delta_f \Delta_h \Pi_h \xrightarrow{\epsilon_h} \Pi_e \Delta_f.$$

The second claim is symmetric to the first if $\alpha = \mathrm{id}$. □

Definition D.4.12. Suppose given a diagram of the form

$$\begin{array}{ccc} R & \xrightarrow{f} & S \\ {\scriptstyle e}\downarrow & \overset{\alpha}{\Rightarrow} & \downarrow{\scriptstyle h} \\ T & \xrightarrow[g]{} & U \end{array}$$

We say it is *exact* if the comparison morphism $\Sigma_f \Delta_e \to \Delta_h \Sigma_g$ is an isomorphism. Note that this is the case if and only if the comparison morphism $\Delta_g \Pi_h \to \Pi_e \Delta_f$ is an isomorphism.

Proposition D.4.13 (Comma Beck-Chevalley for Σ, Δ)**.** *Let $F\colon C \to D$ and $G\colon E \to D$ be functors, and consider the canonical natural transformation diagram (see Definition D.1.9)*

$$\begin{array}{ccc} (F \downarrow G) & \xrightarrow{q} & E \\ {\scriptstyle p}\downarrow & \overset{\alpha}{\Rightarrow} & \downarrow{\scriptstyle G} \\ C & \xrightarrow[F]{} & D \end{array} \tag{12}$$

Let Σ_F be the generalized left push-forward as defined in Remark D.2.12. Then the comparison morphism (Lemma D.4.11) is an isomorphism

$$\Sigma_q \Delta_p \xrightarrow{\cong} \Delta_G \Sigma_F$$

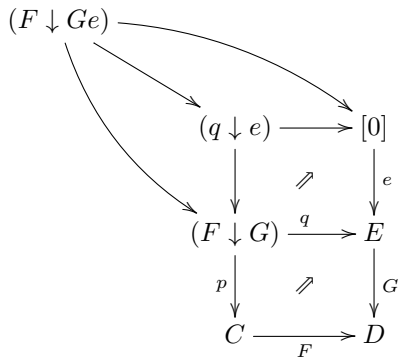
*of functors C–**Inst** $\to E$–**Inst**. In other words, (12) is exact.*

Proof. The map is given by the composition

$$\Sigma_q \Delta_p \xrightarrow{\eta_F} \Sigma_q \Delta_p \Delta_F \Sigma_F \xrightarrow{\alpha} \Sigma_q \Delta_q \Delta_G \Sigma_F \xrightarrow{\epsilon_q} \Delta_G \Sigma_F$$

To prove it is an isomorphism, one checks it on each object $e \in \mathrm{Ob}(E)$ by proving that

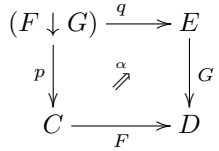
the diagonal map $(F \downarrow Ge) \to (q \downarrow e)$ in the diagram



is a final functor, and this is easily checked.

□

Corollary D.4.14 (Comma Beck-Chevalley for Δ, Π)**.** *Let $F\colon C \to D$ and $G\colon E \to D$ be functors, and consider the canonical natural transformation diagram (11)*



Then the comparison morphism (Lemma D.4.11) is an isomorphism

$$\Delta_F \Pi_G \xrightarrow{\cong} \Pi_p \Delta_q$$

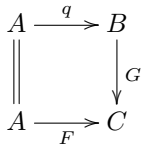
*of functors E–**Inst** $\to C$–**Inst**.*

Proof. This follows from Proposition D.4.13 by adjointness.

□

Sometimes when we push forward an instance I along a functor $B \xrightarrow{G} C$, the resulting instance $\Sigma_G I$ agrees with I, at least on some subcategory $A \subseteq B$. This is very useful to know, because it gives an easy way to calculate row sets and name rows in the pushforward. The following lemma roughly says that this occurs if for all $b \in \mathrm{Ob}(B)$ and $a \in \mathrm{Ob}(A)$, the sense to which b is to the left of a in C, is the same as the sense to which b is to the left of a in B.

Lemma D.4.15. *Suppose that we are given a diagram of the form*

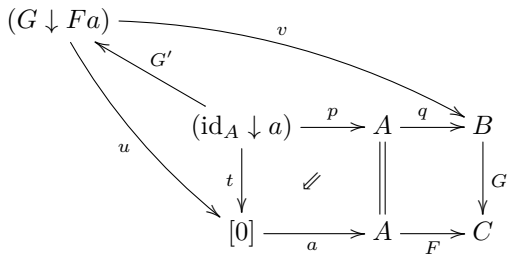


Then the map $\beta\colon \Delta_q \to \Delta_F \Sigma_G$ is an isomorphism if, for all objects $a \in \mathrm{Ob}(A)$ the functor

$$\overline{G}\colon (\mathrm{id}_B \downarrow qa) \to (G \downarrow Fa)$$

is final.

Proof. For typographical convenience, let $[0]$ denote the terminal category, usually denoted $\overrightarrow{0}$. To check that β is an isomorphism, we may choose an arbitrary object $a\colon [0] \to A$ and check that $\Delta_a\beta$ is an isomorphism. Consider the diagram



The map β is the composite

$$\Delta_a\Delta_q \cong \Sigma_t\Delta_p\Delta_q = \Sigma_u\Sigma_{G'}\Delta_{G'}\Delta_v \xrightarrow{\beta'} \Sigma_u\Delta_v \cong \Delta_a\Delta_F\Sigma_G,$$

and this is equivalent to showing that β' is an isomorphism. It suffices to show that $\Sigma_t \to \Sigma_u\Delta_v$ is an isomorphism, but this is equivalent to the assertion that G' is a final functor. One can factor G' as

$$(\mathrm{id}_A \downarrow a) \xrightarrow{\overline{q}} (\mathrm{id}_B \downarrow qa) \xrightarrow{\overline{G}} (G \downarrow Fa)$$

and the first of these is easily checked to be final. Thus the composite is final if and only if the latter map is final, as desired.

□

Proposition D.4.16 (Pullback Beck-Chevalley for Σ, Δ when Σ is along a discrete op-fibration)**.** *Suppose that $F\colon D \to C$ is a functor and $p\colon X \to C$ is a discrete op-fibration, and form the pullback square*

$$\begin{array}{ccc} Y & \xrightarrow{G} & X \\ {\scriptstyle q}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ D & \xrightarrow[F]{} & C \end{array} \tag{13}$$

The comparison morphism (Lemma D.4.11) is a natural isomorphism

$$\Sigma_q\Delta_G \xrightarrow{\cong} \Delta_F\Sigma_p$$

*of functors X–**Inst** $\to$ D–**Inst**. In other words, (13) is exact.*

Proof. Consider the morphism given in Lemma D.4.11. Since η_p and ϵ_q are Cartesian (by Lemma D.4.1), it suffices to check the claim on the terminal object of X–**Inst**, where it follows by Proposition D.4.10.

□

Corollary D.4.17 (Pullback Beck-Chevalley for Δ, Π when Δ is along a discrete op-fibration)**.** *Suppose that $F\colon D \to C$ is a functor and $p\colon X \to C$ is a discrete op-fibration,*

and form the pullback square

$$\begin{array}{ccc} Y & \xrightarrow{G} & X \\ {\scriptstyle q}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ D & \xrightarrow[F]{} & C \end{array} \tag{14}$$

The comparison morphism (Lemma D.4.11) is an isomorphism

$$\Delta_p \Pi_F \xrightarrow{\cong} \Pi_G \Delta_q$$

*of functors D–**Inst** $\to$ X–**Inst**. In other words, (14) is exact.*

Proof. This follows from Proposition D.4.16 by adjointness.

□

Proposition D.4.18 (Distributive law)**.** *Let $u\colon C \to B$ be a discrete op-fibration and $f\colon B \to A$ any functor. Construct the* distributivity diagram

$$\begin{array}{ccccc} & & N & \xrightarrow{g} & M \\ & {\scriptstyle e}\swarrow & \lrcorner & & \\ C & & {\scriptstyle w}\downarrow & & \downarrow{\scriptstyle v} \\ & {\scriptstyle u}\searrow & & & \\ & & B & \xrightarrow[f]{} & A \end{array} \tag{15}$$

as follows. Form $v\colon M \to A$ by $v := \int_A \Pi_f(\partial u)$, form $w\colon N \to B$ by $w := \int_B \Delta_f \Pi_f \partial u$. Finally let $e\colon N \to C$ be given by $\int_B \epsilon_f(\partial u)$, where $\epsilon_f\colon \Delta_f \Pi_f \to \mathrm{id}_B$ is the counit. Note that $N \cong B \times_A M$ by Proposition D.4.10 and that one has $w = u \circ e$ by construction.

Then there is a natural isomorphism

$$\Sigma_v \Pi_g \Delta_e \xrightarrow{\cong} \Pi_f \Sigma_u \tag{16}$$

*of functors C–**Inst** $\to$ A–**Inst**.*

Proof. We have that u, v, and w are discrete op-fibrations. By Corollary D.4.17 we have the following chain of natural transformations

$$\Sigma_v \Pi_g \Delta_e \xrightarrow{\eta_u} \Sigma_v \Pi_g \Delta_e \Delta_u \Sigma_u = \Sigma_v \Pi_g \Delta_w \Sigma_u \xrightarrow{\cong} \Sigma_v \Delta_v \Pi_f \Sigma_u \xrightarrow{\epsilon_v} \Pi_f \Sigma_u.$$

Every natural transformation in the chain is Cartesian, so it suffices to check that the composite is an isomorphism when applied to the terminal object $\{*\}^C$ in C–**Inst**. But there the composition is simply the identity transformation on $\Pi_f(\partial u)$, proving the result.

□

Remark D.4.19. The distributive law above takes on a much simpler form when we realize that any discrete opfibration over B is the category of elements of some functor $I\colon B \to$ **Set**. We have an equivalence of categories $\int(I)$–**Set** $\cong B$–**Set**$_{/I}$. Thinking about the distributive law in these terms is quite helpful.

Let $f\colon B \to A$ be a functor and consider the functor

$$B\text{–}\mathbf{Set}_{/I} \xrightarrow{\text{“}\Pi_f\text{”}} A\text{–}\mathbf{Set}_{/\Pi_f I} \quad [3]$$

given by $(J \to I) \mapsto (\Pi_f J \to \Pi_f I)$. Let $u\colon \int I \to B$ and $v\colon \int(\Pi_f I) \to A$ be the canonical projections. Form the distributivity diagram

$$\begin{array}{ccc} \int \Delta_f \Pi_f I & \xrightarrow{g} & \int \Pi_f I \\ {\scriptstyle e}\downarrow & \lrcorner & \\ \int I & & \downarrow{\scriptstyle v} \\ {\scriptstyle u}\downarrow & & \\ B & \xrightarrow[f]{} & A \end{array}$$

Then the following diagram of categories commutes:

$$\begin{array}{ccc} B\text{–}\mathbf{Set}_{/I} & \xrightarrow{\text{“}\Pi_f\text{”}} & A\text{–}\mathbf{Set}_{/\Pi_f I} \\ {\scriptstyle\cong}\downarrow & & \downarrow{\scriptstyle\cong} \\ (\int I)\text{–}\mathbf{Set} & \xrightarrow[\Pi_g \Delta_e]{} & (\int \Pi_f I)\text{–}\mathbf{Set} \\ {\scriptstyle \Sigma_u}\downarrow & & \downarrow{\scriptstyle \Sigma_v} \\ B\text{–}\mathbf{Set} & \xrightarrow[\Pi_f]{} & A\text{–}\mathbf{Set} \end{array}$$

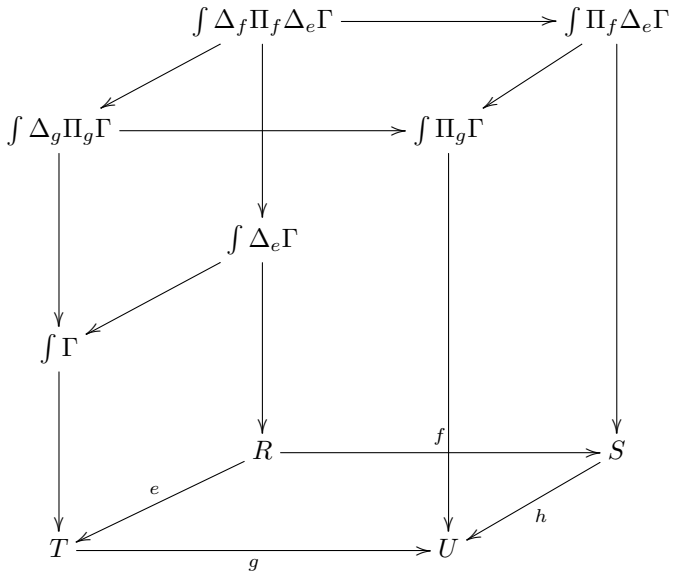
where the vertical composites are the “forgetful” functors. In this diagram, the big rectangle clearly commutes. The distributive law precisely says that the bottom square commutes. Deducing that the top square commutes, we realize that the rather opaque looking $\Pi_g \Delta_e$ is quite a simple functor.

Lemma D.4.20. *Suppose given a pullback square.*

$$\begin{array}{ccc} R & \xrightarrow{f} & S \\ {\scriptstyle e}\downarrow & \lrcorner & \downarrow{\scriptstyle h} \\ T & \xrightarrow[g]{} & U \end{array}$$

[3] The quotes around “Π_f” indicate that this is not actually a right Kan extension, but that “Π_f” is a suggestive name for the functor we indicate.

For any functor $\Gamma\colon T \to \mathbf{Set}$*, every square in the following diagram is a pullback:*

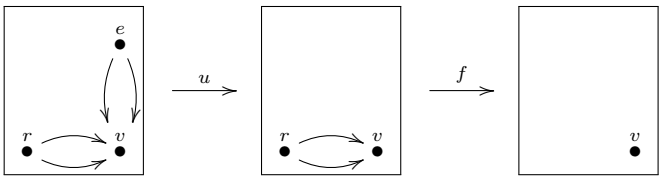


Proof. While the format of this diagram contains a couple copies of the distributivity diagram, this result has nothing to do with that one. Indeed it follows from Proposition D.4.10 and Corollary D.4.17. The bottom square is a pullback by hypothesis. The lower left-hand square, the front square, and the back square are each pullbacks by the proposition. It now follows that the top square is a pullback. The right-hand square is a pullback by a combination of the proposition and the corollary. It now follows that the upper left-hand square is a pullback.

□

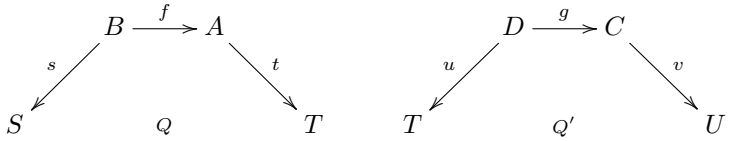
Example D.4.21 (Why Σ-restrictedness appears to be necessary for composability of queries). Without the condition that u be a discrete op-fibration, Proposition D.4.18 does not hold. Indeed, let $C = \mathcal{L}oop$ as in (10) and let $B = A = \overrightarrow{0}$ be the terminal category. One finds that $N = M = \overrightarrow{0}$ too in (15), and e is the unique functor. If one considers C-sets as special kinds of graphs (discrete dynamical systems) then we can say that $\Pi_f \Sigma_u$ will extract the set of connected components for a C-set, whereas $\Sigma_v \Pi_g \Delta_e$ will extract its set of vertices.

We may still wish to ask "whether a Π can be pushed past a Σ", i.e. whether for any $C \xrightarrow{u} B \xrightarrow{f} A$, some appropriate v, g, e exist such that an isomorphism as in (16) holds. The following example may help give intuition. Let $C \xrightarrow{u} B \xrightarrow{f} A$ be given as follows



where $u(e) = v$ and $f(r) = v$. The goal is to find some $C \xleftarrow{e} N \xrightarrow{g} M \xrightarrow{v} A$ such that isomorphism (16) holds. This does not appear possible if M and N are assumed finitely presentable.

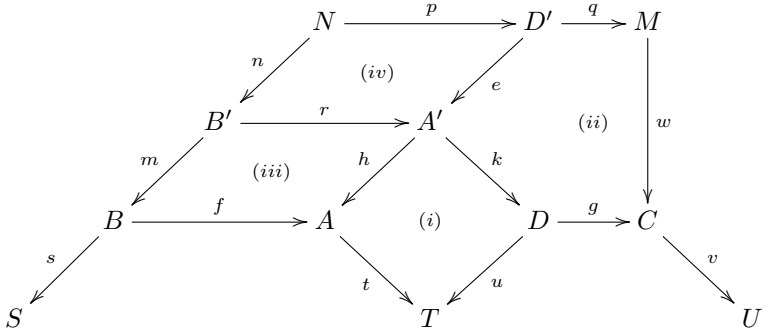
Theorem D.4.22 (Query composition)**.** *Suppose that one has Σ-restricted data migration queries $Q\colon S \rightsquigarrow T$ and $Q'\colon T \rightsquigarrow U$ as follows:*



(17)

Then there exists a Σ-restricted data migration query $Q''\colon S \rightsquigarrow U$ such that $[\![Q'']\!] \cong [\![Q']\!] \circ [\![Q]\!]$.

Proof. We follow the proof in [12], as we have been throughout this section. We will form Q'' by constructing the following diagram, which we will explain step by step:



(18)

First, form (i) by taking the pullback; note that k is a discrete op-fibration by Proposition D.4.10. Second, form (ii) as a distributivity diagram (see Proposition D.4.18) and note that w is a discrete op-fibration. Third, form (iii) and (iv) with $B' = (f \downarrow h)$ and $N = (r \downarrow e)$.

By Proposition D.4.16 we have $\Delta_u \Sigma_t \cong \Sigma_k \Delta_h$. By Proposition D.4.18 we have $\Pi_g \Sigma_k \cong \Sigma_w \Pi_q \Delta_e$. By Corollary D.4.14 we have both $\Delta_h \Pi_f \cong \Pi_r \Delta_m$ and $\Delta_e \Pi_r \cong \Pi_p \Delta_n$. Pulling this all together, we have an isomorphism

$$\begin{aligned}\Sigma_v \Pi_g \Delta_u \Sigma_t \Pi_f \Delta_s \cong \Sigma_v \Pi_g \Sigma_k \Delta_h \Pi_f \Delta_s &\cong \Sigma_v \Sigma_w \Pi_q \Delta_e \Delta_h \Pi_f \Delta_s \\ &\cong \Sigma_v \Sigma_w \Pi_q \Delta_e \Pi_r \Delta_m \Delta_s \\ &\cong \Sigma_v \Sigma_w \Pi_q \Pi_p \Delta_n \Delta_m \Delta_s,\end{aligned}$$

which proves that $[\![Q'']\!] \cong [\![Q']\!] \circ [\![Q]\!]$, where Q'' is the data migration functor given by the triple of morphisms $Q' := (s \circ m \circ n, q \circ p, v \circ w)$, i.e. $[\![Q]\!] = \Sigma_{vw} \Pi_{qp} \Delta_{smn}$. This completes the proof.

□

Corollary D.4.23. *Suppose that $Q\colon S \rightsquigarrow T$ and $Q'\colon T \rightsquigarrow U$ are queries as in (17) and that both are (Δ, Σ)-restricted. Then there exists a (Δ, Σ)-restricted data migration query $Q''\colon S \rightsquigarrow U$ such that $[\![Q'']\!] \cong [\![Q']\!] \circ [\![Q]\!]$.*

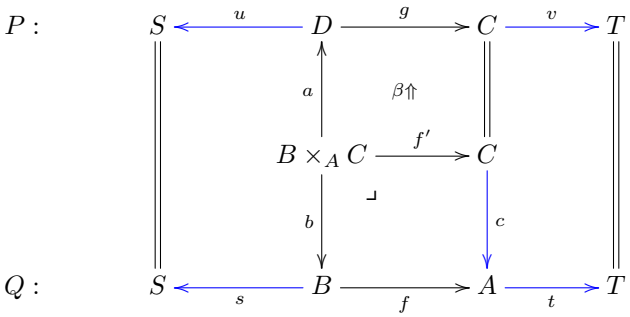
Proof. We form a diagram similar to (18), as follows. First, form (i) by taking the pullback; note that k is a discrete op-fibration by Proposition D.4.10. Second, form (ii) as a distributivity diagram (see Proposition D.4.18) and note that w is a discrete op-fibration. Third, form (iii) and (iv) as pullbacks. Note that h, m, and n will be discrete opfibrations. Following the proof of Theorem D.4.22, the result follows by Corollary D.4.17 in place of Corollary D.4.14.

□

D.5 Syntactic characterization of query morphism and query equivalence

We continue to closely follow [12].

Definition D.5.1 (Morphism of data migration queries)**.** A *morphism of* (Δ, Σ)*-restricted data migration queries from* $P = (u, g, v)$ *to* $Q = (s, f, t)$, denoted $(c, a, \beta)\colon P \to Q$, consists of two functors c, a and a natural transformation β, fitting into a diagram of the form



(19)

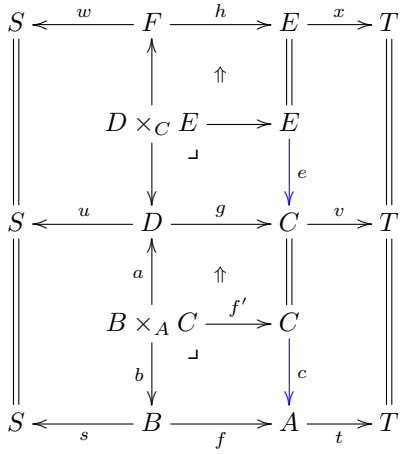
where blue arrows (u, v, s, t, c) are required to be discrete op-fibrations and every square commutes except for the top middle square in which $\beta\colon f' \to ga$ is a natural transformation.

Note that for any diagram of the form (19), the maps a and b will automatically be discrete op-fibrations too.

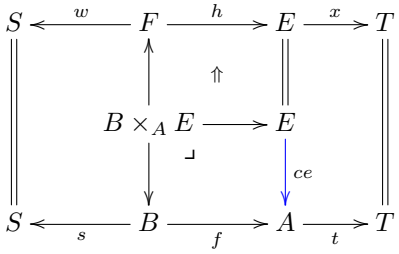
Remark D.5.2. Note that a morphism of queries $P \to Q$ involves a "backward" morphisms of schemas, $a\colon B \times_A C \to D$ as in (19). In fact, this reversal happens in the conjunctive fragment only (see Lemma D.5.9). This may not be surprising because a is a morphism of indexing categories, or "bound variables", and thus the directionality is in line with classical database theory.

Definition D.5.3 (Composition of data migration queries)**.** Suppose given data migration queries $P, Q, R\colon S \rightsquigarrow T$ and morphisms $P \to Q$ and $Q \to R$ as in Definition 19. Then they can be composed to give a query morphism $P \to R$, as follows.

We begin the following diagram



By Proposition D.2.17 there is a canonical map $f' \times_c e \to (ga) \times_C e$, and this allows us to construct a composite



Remark D.5.4. Note that this composition is associative *up to isomorphism*, but not on the nose. Thus we cannot speak of the category of data migration queries $P \rightsquigarrow Q$, but only the $(\infty, 1)$-category of such.

The following definition is slightly abbreviated, i.e. not fully spelled out, but hopefully it is clear to anyone who is following so far.

Definition D.5.5 (Category of data migration queries)**.** We define the *category of* (Δ, Σ)*-restricted data migration queries from* S *to* T, denoted $\mathrm{RQry}(S, T)$, to be the category whose objects are (Δ, Σ)-restricted data migration queries $S \rightsquigarrow T$ and whose morphisms are equivalence classes of diagrams as in (19), where two such diagrams are equivalent if there are equivalences of categories between corresponding objects in their middle rows and the functors commute appropriately.

Lemma D.5.6. *Given a morphism of* (Δ, Σ)*-restricted data migration queries* $P \to Q$*, there is an induced natural transformation* $[\![P]\!] \to [\![Q]\!]$*.*

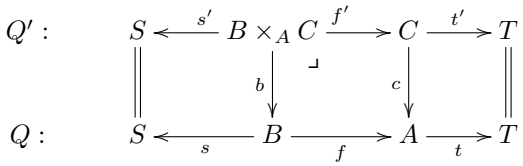
Proof. By Proposition D.2.17 and Corollary D.4.17 we have the natural natural transformations below:

$$\begin{aligned} \Sigma_v \Pi_g \Delta_u = \Sigma_t \Sigma_c \Pi_g \Delta_u &\xrightarrow{\eta_a} \Sigma_t \Sigma_c \Pi_g \Pi_a \Delta_a \Delta_u \\ &\xrightarrow{\beta} \Sigma_t \Sigma_c \Pi_{f'} \Delta_a \Delta_u \\ &= \Sigma_t \Sigma_c \Pi_{f'} \Delta_b \Delta_s \\ &\cong \Sigma_t \Sigma_c \Delta_c \Pi_f \Delta_s \xrightarrow{e_c} \Sigma_t \Pi_f \Delta_s \end{aligned} \tag{20}$$

□

Lemma D.5.7. *Let* $Q \colon S \rightsquigarrow T$ *be a* (Δ, Σ)*-restricted data migration query, and let* $P \colon S\text{–}\mathbf{Inst} \to T\text{–}\mathbf{Inst}$ *be any functor. If there exists a Cartesian natural transformation* $\phi \colon P \to [\![Q]\!]$*, then there is an essentially unique* (Δ, Σ)*-restricted data migration query isomorphic to* P*.*

Proof. Suppose that Q is represented by the bottom row in the diagram below



We form the rest of the diagram as follows. Let $*$ be the terminal object in S–**Inst**, so in particular $\int_T Q(*) \cong t$. Let $t' = \int_T P(*)$ and define $c\colon t' \to t$ over T to be $\int_T \phi(*)$. Let b be the pullback of c and $s' = s \circ b$. The Diagram (21) now formed, let $Q'\colon S \rightsquigarrow T$ be the top row. Note that c, and therefore b and s, are discrete op-fibrations, so in particular Q' is a (Δ, Σ)-restricted data migration query. Note also that there was essentially no choice in the definition of Q'.

The map $[\![Q']\!] \to [\![Q]\!]$ is given by

$$\Sigma_{t'}\Pi_{f'}\Delta_{s'} = \Sigma_t\Sigma_c\Pi_{f'}\Delta_b\Delta_s \xrightarrow{\cong} \Sigma_t\Sigma_c\Delta_c\Pi_f\Delta_s \xrightarrow{\eta_c} \Sigma_t\Pi_f\Delta_s,$$

so it is Cartesian by Lemma D.4.1. But if both P and Q' are Cartesian over Q and if they agree on the terminal object (as they do by construction), then they are isomorphic. Thus $P \cong Q'$ is a (Δ, Σ)-restricted data migration query.

□

Lemma D.5.8 (Yoneda Functorialization)**.** *Let C be a category and let $b\colon C \to$ **Set** be a functor with $s\colon B \to C$ its Grothendieck category of elements. Then there is a natural isomorphism of functors*

$$\mathrm{Hom}_{C\text{–}\mathbf{Set}}(b, -) \cong \Pi_t\Pi_s\Delta_s$$

*where $t\colon I \to \overrightarrow{0}$ is the unique functor. Moreover, for any $b'\colon C \to$ **Set** with $s'\colon B' \to C$ its category of elements, there is a bijection*

$$\mathrm{Hom}_{\mathbf{Cat}/C}(B, B') \cong \mathrm{Hom}_{C\text{–}\mathbf{Set}}(b, b') \cong \mathrm{Hom}(\Pi_t\Pi_{s'}\Delta_{s'}, \Pi_t\Pi_s\Delta_s).$$

Proof. The second claim follows from the first by Proposition D.4.7 and the usual Yoneda-style argument. For the first, we have

$$\begin{aligned}\Pi_u\Pi_s\Delta_s(-) \cong \mathrm{Hom}_{\mathbf{Set}}(*, \Pi_u\Pi_s\Delta_s(-)) \cong \mathrm{Hom}_{B\text{–}\mathbf{Set}}(*^B, \Delta_s(-)) &\cong \mathrm{Hom}_{C\text{–}\mathbf{Set}}(\Sigma_s(*^B), -)\\ &\cong \mathrm{Hom}_{C\text{–}\mathbf{Set}}(b, -).\end{aligned}$$

□

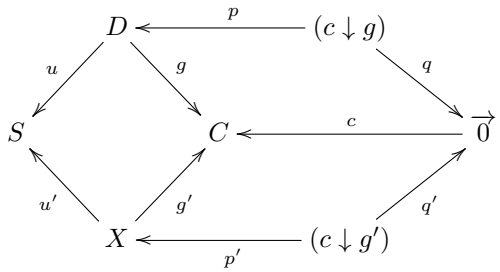
Lemma D.5.9. *Suppose given the diagram to the left, where u and u' are discrete op-fibrations:*

$$\begin{array}{ccccc} S & \xleftarrow{u} & D & \xrightarrow{g} & C \\ \| & & & & \| \\ S & \xleftarrow[u']{} & X & \xrightarrow[g']{} & C \end{array} \qquad\qquad \begin{array}{ccccc} S & \xleftarrow{u} & D & \xrightarrow{g} & C \\ \| & & \uparrow{\scriptstyle a} & \beta\Uparrow & \| \\ S & \xleftarrow[u']{} & X & \xrightarrow[g']{} & C \end{array}$$

and a natural transformation $j\colon \Pi_g\Delta_u \to \Pi_{g'}\Delta_{u'}$. Then there exists an essentially unique diagram as to the right, such that j is the composition (see Proposition D.2.17),

$$\Pi_g\Delta_u \xrightarrow{\eta_a} \Pi_g\Pi_a\Delta_a\Delta_u \xrightarrow{\beta} \Pi_{g'}\Delta_a\Delta_u \cong \Pi_{g'}\Delta_{u'}. \tag{22}$$

Proof. Choose an object $c\colon \overrightarrow{0} \to C$ in C. Form the diagram



Let $s = u \circ p$ and $s' = u' \circ p'$, and let $t\colon S \to \overrightarrow{0}$ denote the unique functor. It is easy to show that p and p', and hence s and s' are discrete op-fibrations. We have a natural transformation

$$\Pi_t\Pi_s\Delta_s = \Pi_q\Delta_s = \Pi_q\Delta_p\Delta_u \cong \Delta_c\Pi_g\Delta_u \xrightarrow{j} \Delta_c\Pi_{g'}\Delta_{u'} \cong \Pi_{q'}\Delta_{p'}\Delta_{u'} = \Pi_{q'}\Delta_{s'} = \Pi_t\Pi_{s'}\Delta_{s'}.$$

By Lemma D.5.8 this is equivalent to giving a map $j_c\colon (c \downarrow g') \to (c \downarrow g)$ over S. Moreover, given a map $f\colon c_0 \to c_1$ in C, the natural transformation $\Delta_f\colon \Delta_{c_0} \to \Delta_{c_1}$ (see Proposition D.2.17) induces a commutative diagram of functors $S \to \mathbf{Set}$, as to the left

$$\begin{array}{ccc} \Delta_{c_1}\Pi_{g'}\Delta_{u'} & \longleftarrow & \Delta_{c_1}\Pi_g\Delta_u \\ \uparrow & & \uparrow \\ \Delta_{c_0}\Pi_{g'}\Delta_{u'} & \longleftarrow & \Delta_{c_0}\Pi_g\Delta_u \end{array} \qquad \begin{array}{ccc} (c_1 \downarrow g') & \xrightarrow{j_{c_1}} & (c_1 \downarrow g) \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle f} \\ (c_0 \downarrow g') & \xrightarrow[j_{c_0}]{} & (c_0 \downarrow g) \end{array}$$

which induces a commutative diagram of categories over S as to the right.

We are now in a position to construct a unique functor $a\colon X \to D$ with $u' = u \circ a$ and natural transformation $\beta\colon g' \to g \circ a$ agreeing with each j_c. Given an object $x \in \mathrm{Ob}(X)$, we apply the map $j_{g'x}\colon (g'x \downarrow g') \to (g'x \downarrow g)$ to $(x, \mathrm{id}_{g'x})$ to get some $(d, g'x \xrightarrow{b} gd)$. We assign $a(x) := d$ and $\beta_x := b$, and note that $u'(x) = u(d)$ as above. A similar argument applies to morphisms, so a and β are constructed. To see that j is the composite given in (22), one checks it on objects and arrows in C using the formula from Construction D.2.14 in conjunction with Proposition D.2.16.

□

Theorem D.5.10. *Suppose that $P, Q\colon S \rightsquigarrow T$ are (Δ, Σ)-restricted data migration queries, and $X\colon [\![P]\!] \to [\![Q]\!]$ is a natural transformation of functors. Then there exists an essentially unique morphism of (Δ, Σ)-restricted data migration queries $x\colon P \to Q$ (i.e. a diagram of the form (19)) such that $[\![x]\!] \cong X$. In other words, the functor* $\mathrm{RQry}(S, T) \to \mathrm{Fun}(S, T)$ *is fully faithful.*

Proof. Let $*$ denote the terminal object in S–**Inst**. We define a functor $\overline{P}\colon S$–**Inst** $\to$ T–**Inst** as follows. For $I\colon S \to \mathbf{Set}$, define $\overline{P}(I)$ as the fiber product

$$\begin{array}{ccc} \overline{P}(I) & \longrightarrow & [\![Q]\!](I) \\ \downarrow & \lrcorner & \downarrow \\ [\![P]\!](*) & \xrightarrow[X(*)]{} & [\![Q]\!](*). \end{array}$$

There is a induced natural transformation $i\colon [\![P]\!] \to \overline{P}$ and an induced Cartesian natural transformation $X'\colon \overline{P} \to [\![Q]\!]$, such that $X = X' \circ i$.

Let $v\colon C \to T$ and $t\colon A \to T$ denote the category of elements $v := \int_T [\![P]\!](*) = \int_T \overline{P}(*)$ and $t := \int_T [\![Q]\!](*)$ respectively, and let $c := \int_T X(*)\colon v \to t$ over T. If P and Q are the top and bottom rows in the diagram below, then Lemma D.5.7 and in particular Diagram (21) implies that there is an essentially unique way to fill out the middle row, and its maps down to the bottom row, as follows:

$$\begin{array}{lccccccc}
P: & S & \xleftarrow{u} & D & \xrightarrow{g} & C & \xrightarrow{v} & T \\
 & \| & & & & \| & & \| \\
P': & S & \xleftarrow{s'} & B \times_A C & \xrightarrow{f'} & C & \xrightarrow{v} & T \\
 & \| & & \downarrow{\scriptstyle b} & \lrcorner & \downarrow{\scriptstyle c} & & \| \\
Q: & S & \xleftarrow[s]{} & B & \xrightarrow[f]{} & A & \xrightarrow[t]{} & T
\end{array}$$

with $s' = s \circ b$ and $[\![P']\!] = \overline{P}$.

Lemma D.4.1 implies that $i\colon \Sigma_v \Pi_g \Delta_u \to \Sigma_v \Pi_{f'} \Delta_{s'}$ induces a natural transformation $j\colon \Pi_g \Delta_u \to \Pi_{f'} \Delta_{s'}$ with $\Sigma_v j = i$. The result follows by Lemma D.5.9.

□

E Typed signatures

E.1 Definitions and data migration

Definition E.1.1. A *typed signature* $\overline{\mathcal{C}}$ is a sequence $\overline{\mathcal{C}} := (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ where $\mathcal{C}$ is a signature, $\mathcal{C}_0$ is a discrete category, [4] $i\colon \mathcal{C}_0 \to \mathcal{C}$ is a functor, and $\Gamma\colon \mathcal{C}_0 \to \mathbf{Set}$ is a functor,

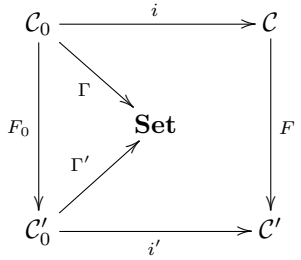
$$\begin{array}{ccc}
\mathcal{C}_0 & \xrightarrow{\quad i \quad} & \mathcal{C} \\
 & \searrow^{\Gamma} & \\
 & & \mathbf{Set}
\end{array}$$

The signature $\mathcal{C}$ is called the *structure part* of $\overline{\mathcal{C}}$, and the rest is called the *typing setup*.

Suppose $(\mathcal{C}', \mathcal{C}_0', i', \Gamma')$ is another typed signature. A *typed signature morphism from* $\overline{\mathcal{C}}$ *to* $\overline{\mathcal{C}'}$, denoted $\overline{F} = (F, F_0)\colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$, consists of a functor $F\colon \mathcal{C} \to \mathcal{C}'$ and a functor

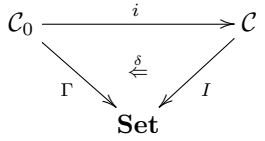
[4] In fact, one does not need to assume that $\mathcal{C}_0$ is a discrete category for the following results to hold. Still, it is conceptually simpler. To get a feeling for what could be done if $\mathcal{C}_0$ were not discrete, consider the possibility of a table having two data columns, an attribute A of type string and an attribute B of type integer, where the integer in B was the *length* of the string in A.

$F_0 \colon \mathcal{C}_0 \to \mathcal{C}'_0$ such that the following diagram commutes:

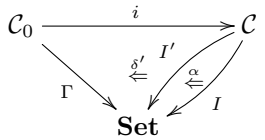


The category of typed signatures is denoted **TSig**.

A $\overline{\mathcal{C}}$*-instance* $\overline{I}$ is a pair $\overline{I} := (I, \delta)$ where $I \colon \mathcal{C} \to \mathbf{Set}$ is a functor, called the *structure part of* $\overline{I}$ together with a natural transformation $\delta \colon I \circ i \to \Gamma$, called the *data part of* $\overline{I}$.



Suppose $\overline{I'} := (I', \delta')$ is another $\overline{\mathcal{C}}$-instance. An *instance morphism from* $\overline{I}$ *to* $\overline{I'}$, denoted $\alpha \colon \overline{I} \to \overline{I'}$, is a natural transformation $\alpha \colon I \to I'$ such that $\delta' \circ \alpha = \delta$.



The category of $\overline{\mathcal{C}}$-instances is denoted $\overline{\mathcal{C}}$–**Inst**.

Proposition E.1.2. *Let* $\overline{\mathcal{C}}$ *be a typed signature. Then the category* $\overline{\mathcal{C}}$–**Inst** *is a topos.*

Proof. $\overline{\mathcal{C}}$–**Inst** is equivalent to the slice topos $\mathcal{C}$–$\mathbf{Set}/\Pi_i\Gamma$.

□

Construction E.1.3. Let $\overline{\mathcal{C}} := (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ be a typed signature. We set up the tables for it as follows. For every arrow in $\mathcal{C}$ we make a binary table; we call these *arrow tables*. For every object $c \in \mathrm{Ob}(\mathcal{C})$, let $N = i^{-1}(c) \subseteq \mathcal{C}_0$. We make an $1 + |N|$ column table, where $|N|$ is the cardinality of N; we call these *node tables*. For each node table, one column is the primary key column and the other N columns are called *data columns*. The *data type* for each $n \in N$ is the set $\Gamma(n)$.

Let (I, δ) be an instance of $\overline{\mathcal{C}}$. For each object $c \in \mathrm{Ob}(\mathcal{C})$ we fill in the primary key column of the node table with the set $I(c) \in \mathrm{Ob}(\mathbf{Set})$. For each data column $n \in N$ we have a function $\delta_n \colon I(c) \to \Gamma(n)$, which we use to fill in the data in that column. For every arrow $f \colon c \to c'$ in $\mathcal{C}$ we fill in the primary key column of the arrow table with $I(c)$, and we fill in the other column with the function $I(f) \colon I(c) \to I(c')$.

Example E.1.4. If $\mathcal{C}_0 = \emptyset$ then for any category presentation $\mathcal{C}$ there is a unique functor $i \colon \mathcal{C}_0 \to \mathcal{C}$ and a unique functor $\Gamma \colon \mathcal{C}_0 \to \mathbf{Set}$, so there is a unique schema $\overline{\mathcal{C}} = (\mathcal{C}, \emptyset, i, \Gamma)$ with structure $\mathcal{C}$ and empty domain setup.

Given a functor $I \colon \mathcal{C} \to \mathbf{Set}$, there is a unique instance $\overline{I} := (I, !) \in \overline{\mathcal{C}}$–**Inst** with that structure part. The data part of $\overline{I}$ is empty.

Example E.1.5. Let $\mathcal{C} = \boxed{\bullet^X}$ be a terminal category, and let $\mathcal{C}_0 = \{\text{First}, \text{Last}\}$. There is a unique $i\colon \mathcal{C}_0 \to \mathcal{C}$. Let

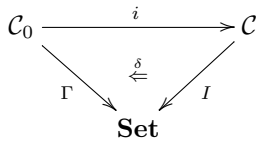
$$\Gamma(\text{First}) = \Gamma(\text{Last}) = \mathit{Strings}.$$

Thus we have our schema $\overline{\mathcal{C}}$.

An instance on $\overline{\mathcal{C}}$ consists of a functor $I\colon \mathcal{C} \to \mathbf{Set}$ and a natural transformation $\delta\colon I \circ i \to \Gamma$. Let $I(X) = \{1, 2\}$, let $\delta_{\text{First}}(1) = \texttt{David}$, let $\delta_{\text{First}}(2) = \texttt{Ryan}$, let $\delta_{\text{Last}}(1) = \texttt{Spivak}$, let $\delta_{\text{Last}}(2) = \texttt{Wisnesky}$. We display this as

X		
ID	**First**	**Last**
1	David	Spivak
2	Ryan	Wisnesky

Definition E.1.6. Suppose given a typed signature $\overline{\mathcal{C}}$ and a typed instance $\overline{I}$,

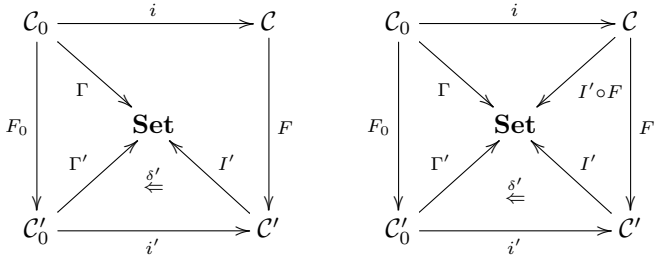


We have a morphism $\delta\colon I \to \Pi_i\Gamma$. We say our instance $\overline{I}$ is *relational* if δ is a monomorphism. We define the *relationalization of* $\overline{I}$ denoted $REL(\overline{I})$ to be the image $\text{im}(\delta) \subseteq \Pi_i\Gamma$.

Proposition E.1.7. *Let* $\overline{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ *and* $\overline{\mathcal{C}'} = (\mathcal{C}', \mathcal{C}'_0, i', \Gamma')$ *be typed signatures, and let* $\overline{F} = (F, F_0)\colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$ *be a typed signature morphism. Then the pullback functor* $\Delta_F\colon \mathcal{C}'\text{–}\mathbf{Inst} \to \mathcal{C}\text{–}\mathbf{Inst}$ *of untyped instances extends to a functor of typed instances*

$$\Delta_{\overline{F}}\colon \overline{\mathcal{C}'}\text{–}\mathbf{Inst} \to \overline{\mathcal{C}}\text{–}\mathbf{Inst}.$$

Proof. An object $(I', \delta') \in \text{Ob}(\overline{\mathcal{C}'}\text{–}\mathbf{Inst})$ is drawn to the left; simply compose with F to get the diagram on the right

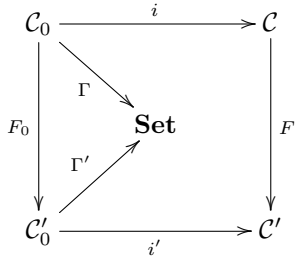


More formally, by Lemma D.4.11 we have a natural transformation $\Delta_F\Pi_{i'} \to \Pi_i\Delta_{F_0}$, so we set $I = \Delta_F I'$ and we set δ' to be the composite

$$\Delta_i I = \Delta_i\Delta_F I' = \Delta_{F_0}\Delta_{i'} I' \xrightarrow{\Delta_{F_0}\delta'} \Delta_{F_0}\Gamma' = \Gamma$$

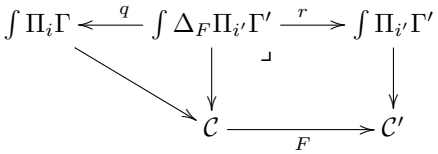
□

Remark E.1.8. Suppose given the following diagram:



(23)

A $\overline{\mathcal{C}'}$-instance is a functor $\int \Pi_{i'}\Gamma' \to \mathbf{Set}$, whereas a $\overline{\mathcal{C}}$ instance is a functor $\int \Pi_i \Gamma \to \mathbf{Set}$. We can reinterpret the typed-Δ functor using the following commutative diagram



The functor $\Delta_{\overline{F}}$ is given by the composition $\Sigma_q \Delta_r$.

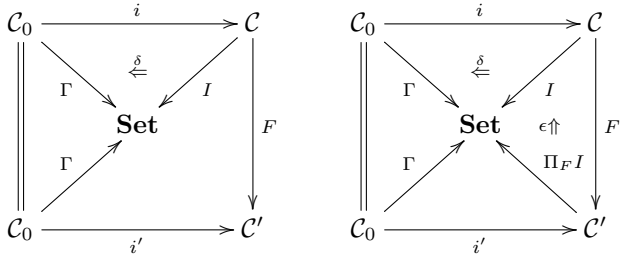
Note that if (the exterior square of) Diagram 23 is a pullback square, then q is an isomorphism.

Definition E.1.9. Let $\overline{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ and $\overline{\mathcal{C}'} = (\mathcal{C}', \mathcal{C}_0, i', \Gamma)$ be typed signatures, and let $\overline{F} = (F, F_0) \colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$ be a typed signature morphism. We say that $\overline{F}$ is Π-*ready* if $F_0 = \mathrm{id}_{\mathcal{C}_0}$.

Proposition E.1.10. *Let $\overline{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ and $\overline{\mathcal{C}'} = (\mathcal{C}', \mathcal{C}_0, i', \Gamma)$ be typed signatures, and let $\overline{F} = (F, F_0) \colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$ be a typed signature morphism that is Π-ready. Then the right pushforward functor $\Pi_F \colon \mathcal{C}$–**Inst** $\to \mathcal{C}'$–**Inst** of untyped instances extends to a functor of typed instances*

$$\Pi_{\overline{F}} \colon \overline{\mathcal{C}}\text{–}\mathbf{Inst} \to \overline{\mathcal{C}'}\text{–}\mathbf{Inst}.$$

Proof. Given the diagram to the left, we form $\Pi_F(I) \colon \mathcal{C}' \to \mathbf{Set}$ to get the diagram on the right:



The morphism $\epsilon \colon \Delta_F \Pi_F I \to I$ is the counit map. We set $\delta' \colon \Delta_{i'} \Pi_F I \to \Gamma$ to be the composite

$$\Delta_{i'} \Pi_F I = \Delta_i \Delta_F \Pi_F I \xrightarrow{\epsilon} \Delta_i I \xrightarrow{\delta} \Gamma.$$

□

Remark E.1.11. A $\overline{\mathcal{C}}$-instance is a functor $\int \Pi_i \Gamma \to \mathbf{Set}$, whereas a $\overline{\mathcal{C}'}$ instance is a functor $\int \Pi_{i'} \Gamma = \int \Pi_F \Pi_i \Gamma \to \mathbf{Set}$. We can reinterpret the typed-Π functor using the following diagram (see Remark D.4.19).

$$\begin{array}{ccc} \int \Delta_F \Pi_F \Pi_i \Gamma & \xrightarrow{g} & \int \Pi_F \Pi_i \Gamma \\ \downarrow{\scriptstyle e} & \lrcorner & \\ \int \Pi_i \Gamma & & \downarrow{\scriptstyle v} \\ \downarrow{\scriptstyle u} & & \\ \mathcal{C} & \xrightarrow[F]{} & \mathcal{C}' \end{array}$$

The functor $\Pi_{\overline{F}}$ is given by

$$\Pi_g \Delta_e \colon (\textstyle\int \Pi_i \Gamma)\text{–}\mathbf{Set} \to (\int \Pi_F \Pi_i \Gamma)\text{–}\mathbf{Set}.$$

Definition E.1.12. Let $\overline{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ and $\overline{\mathcal{C}'} = (\mathcal{C}', \mathcal{C}_0, i', \Gamma)$ be typed signatures, and let $\overline{F} = (F, F_0) \colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$ be a typed signature morphism. We say that $\overline{F}$ is Σ-*ready* if

- p is a discrete opfibration, and
- $C_0 = p^{-1}(C_0')$, i.e. the following is a fiber product of categories,

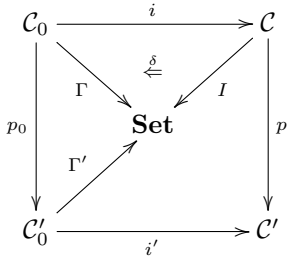
$$\begin{array}{ccc} \mathcal{C}_0 & \xrightarrow{i} & \mathcal{C} \\ \downarrow{\scriptstyle p_0} & \lrcorner & \downarrow{\scriptstyle p} \\ \mathcal{C}_0' & \xrightarrow[i']{} & \mathcal{C}' \end{array}$$

.

Proposition E.1.13. *Let $\overline{\mathcal{C}} = (\mathcal{C}, \mathcal{C}_0, i, \Gamma)$ and $\overline{\mathcal{C}'} = (\mathcal{C}', \mathcal{C}_0', i', \Gamma')$ be typed signatures, and let $\overline{p} = (p, p_0) \colon \overline{\mathcal{C}} \to \overline{\mathcal{C}'}$ be a typed signature morphism that is Σ-ready. Then the left pushforward functor $\Sigma_p \colon \mathcal{C}\text{–}\mathbf{Inst} \to \mathcal{C}'\text{–}\mathbf{Inst}$ of untyped instances extends to a functor of typed instances*

$$\Sigma_{\overline{p}} \colon \overline{\mathcal{C}}\text{–}\mathbf{Inst} \to \overline{\mathcal{C}'}\text{–}\mathbf{Inst}.$$

Proof. An instance $\delta \colon \Delta_i I \to \Gamma$ is drawn below



Let $I' = \Sigma_p I$; we need a morphism $\Delta_{i'} I' \to \Gamma'$. Since $\Gamma \cong \Delta_{p_0} \Gamma'$, we indeed have by Proposition D.4.16

$$\Delta_{i'} \Sigma_p I \xrightarrow{\cong} \Sigma_{p_0} \Delta_i I \xrightarrow{\Sigma_{p_0} \delta} \Sigma_{p_0} \Delta_{p_0} \Gamma' \xrightarrow{\epsilon} \Gamma'.$$

□

Remark E.1.14. A $\overline{\mathcal{C}}$-instance is a functor $\int \Pi_i \Gamma \to \mathbf{Set}$, whereas a $\overline{\mathcal{C}'}$ instance is a functor $\int \Pi_{i'} \Gamma' \to \mathbf{Set}$. We can reinterpret the typed-Σ functor using the following diagram.

$$\begin{array}{ccc} \int \Pi_i \Gamma & \xrightarrow{p'} & \int \Pi_{i'} \Gamma' \\ \downarrow & \lrcorner & \downarrow \\ \mathcal{C} & \xrightarrow[p]{} & \mathcal{C}' \end{array} \tag{24}$$

The above diagram is a fiber product diagram because of the isomorphism (see Corollary D.4.17):

$$\Pi_i \Gamma = \Pi_i \Delta_{p_0} \Gamma' \cong \Delta_p \Pi_{i'} \Gamma'.$$

At this point we can interpret our typed pushforward using the natural isomorphism

$$\Sigma_{\overline{p}} \cong \Sigma_{p'}.$$

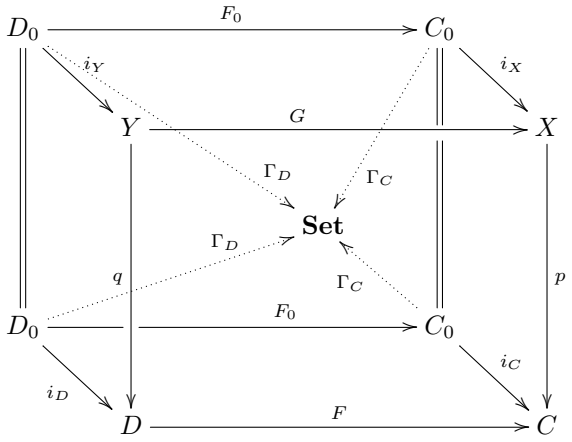
Definition E.1.15. Let $\overline{C} = (S, S_0, i_S, \Gamma_S)$ and $\overline{T} = (T, T_0, i_T, \Gamma_T)$ be typed signatures. A *typed FQL query* Q from $\overline{S}$ to $\overline{T}$, denoted $Q \colon \overline{S} \rightsquigarrow \overline{T}$ is a triple of typed signature morphisms $(\overline{F}, \overline{G}, \overline{H})$:

$$\overline{S} \xleftarrow{\overline{F}} \overline{S'} \xrightarrow{\overline{G}} \overline{S''} \xrightarrow{\overline{H}} \overline{T}$$

such that $\overline{G}$ is Π-ready and $\overline{H}$ is Σ-ready.

E.2 Typed query composition

Proposition E.2.1 (Comparison morphism for typed Δ, Π)**.** *Suppose given the following diagram of typed signatures,*



(25)

such that all diagrams commute, except that the front square for which we have a natural transformation $\alpha \colon Fq \to pG$:

$$\begin{array}{ccc} Y & \xrightarrow{G} & X \\ {\scriptstyle q}\downarrow & \overset{\alpha}{\Rightarrow} & \downarrow{\scriptstyle p} \\ D & \xrightarrow[F]{} & C \end{array}$$

Then the comparison transformation for untyped instances $\Delta_F \Pi_p \to \Pi_q \Delta_G$*, from Lemma D.4.11, extends to a* typed comparison morphism *of typed queries* $\overline{X}$–**Inst** $\to$ $\overline{D}$–**Inst**,

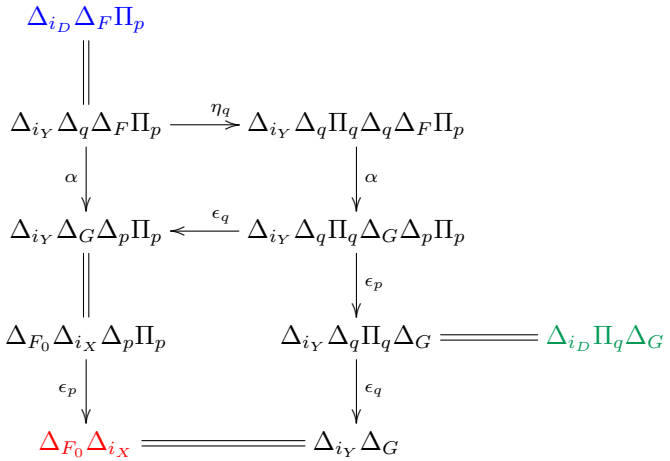
$$\Delta_{\overline{F}} \Pi_{\overline{p}} \to \Pi_{\overline{q}} \Delta_{\overline{G}}.$$

Proof. Suppose given an $\overline{X}$-instance $(I, \delta \colon \Delta_{i_X} I \to \Gamma_C$. The formulas in Propositions E.1.7 and E.1.10 become

$$\Delta_{i_D} \Delta_F \Pi_p I = \Delta_{F_0} \Delta_{i_C} \Pi_p I = \Delta_{F_0} \Delta_{i_X} \Delta_p \Pi_p I \xrightarrow{\epsilon_p} \Delta_{F_0} \Delta_{i_X} I \xrightarrow{\delta} \Delta_{F_0} \Gamma_C = \Gamma_D \tag{26}$$

$$\Delta_{i_D} \Pi_q \Delta_G I = \Delta_{i_Y} \Delta_q \Pi_q \Delta_G I \xrightarrow{\epsilon_q} \Delta_{i_Y} \Delta_G I = \Delta_{F_0} \Delta_{i_X} I \xrightarrow{\delta} \Delta_{F_0} \Gamma_C = \Gamma_D \tag{27}$$

We need to show that the comparison morphism above commutes with these maps to Γ_D. To see this, consider the following commutative diagram:



The left-hand composite $\Delta_{i_D} \Delta_F \Pi_p \to \Delta_{F_0} \Delta_{i_X}$ is the first part of (26), the lower-right zig-zag $\Delta_{i_D} \Pi_q \Delta_G \to \Delta_{F_0} \Delta_{i_X}$ is the first part of (27), and the map from top to lower-right $\Delta_{i_D} \Delta_F \Pi_p \to \Delta_{i_D} \Pi_q \Delta_G$ is Δ_{i_D} applied to the comparison morphism. The commutativity of the diagram (which follows by the triangle identities and the associativity law) is what we were trying to prove.

□

Corollary E.2.2 (Pullback Beck-Chevalley for typed Δ, Π)**.** *Suppose given Diagram (25) such that the front square is a pullback:*

$$\begin{array}{ccc} Y & \xrightarrow{G} & X \\ {\scriptstyle q}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ D & \xrightarrow[F]{} & C \end{array}$$

and p *is a discrete op-fibration. Then the typed comparison morphism is an isomorphism:*

$$\Delta_{\overline{F}} \Pi_{\overline{p}} \xrightarrow{\cong} \Pi_{\overline{q}} \Delta_{\overline{G}}$$

Proof. By Corollary D.4.17, the comparison morphism is an isomorphism,

$$\Delta_F \Pi_p \xrightarrow{\cong} \Pi_q \Delta_G.$$

The result follows from Proposition E.2.1, where $i_Y : D_0 \to Y$ is induced by the fact that Y is a fiber product.

□

Corollary E.2.3 (Comma Beck-Chevalley for typed Δ, Π)**.** *Suppose given Diagram (25) such that the front square is a comma category:*

$$\begin{array}{ccc} Y = (F \downarrow p) & \xrightarrow{G} & X \\ {\scriptstyle q}\downarrow & \overset{\alpha}{\swarrow} & \downarrow{\scriptstyle p} \\ D & \xrightarrow[F]{} & C \end{array}$$

Then the typed comparison morphism is an isomorphism:

$$\Delta_{\overline{F}} \Pi_{\overline{p}} \xrightarrow{\cong} \Pi_{\overline{q}} \Delta_{\overline{G}}$$

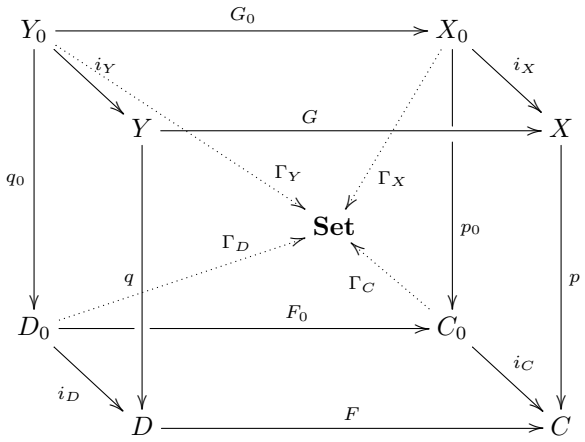
Proof. By Corollary D.4.14, the comparison morphism is an isomorphism,

$$\Delta_F \Pi_p \xrightarrow{\cong} \Pi_q \Delta_G.$$

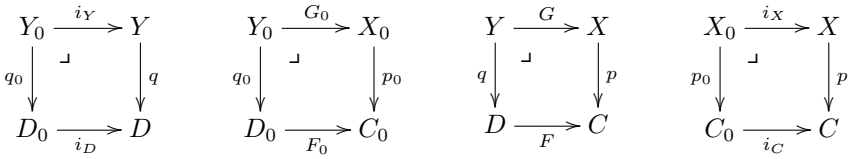
The result follows from Proposition E.2.1, where $i_Y : D_0 \to Y$ is the induced map factoring through the canonical morphism $F \times_C p \to (F \downarrow p)$.

□

Proposition E.2.4 (Pullback Beck-Chevalley for typed Σ, Δ)**.** *Suppose given the following commutative diagram of typed signatures,*



in which $p : X \to C$ is a discrete op-fibration, and the four side squares



are pullbacks. Then there is an isomorphism

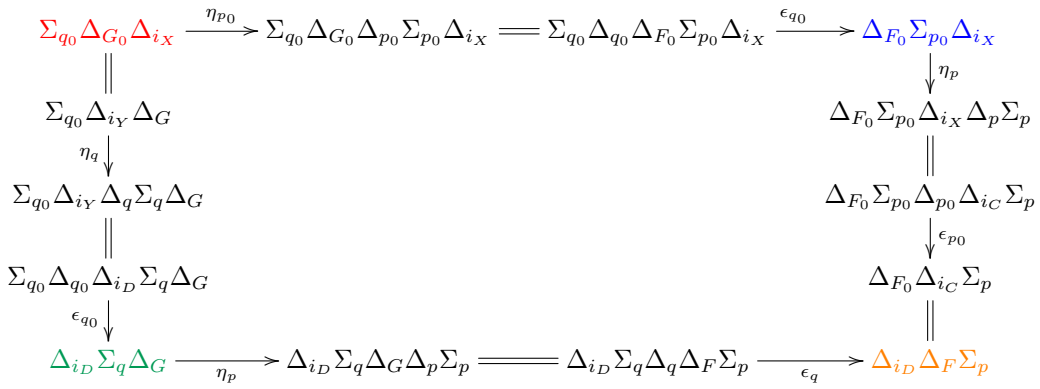
$$\Sigma_{\overline{q}}\Delta_{\overline{G}} \xrightarrow{\cong} \Delta_{\overline{F}}\Sigma_{\overline{p}}$$

of functors $\overline{X}$–**Inst** $\to \overline{D}$–**Inst**.

Proof. Note that the functors q, p_0, and q_0 are discrete opfibrations too. Suppose given an $\overline{X}$-instance $(I, \delta\colon \Delta_{i_X} I \to \Gamma_X$. By Proposition D.4.16, we have a Beck-Chevalley isomorphism $\Sigma_q\Delta_G \xrightarrow{\cong} \Delta_F\Sigma_p$. The formulas in Propositions E.1.7 and E.1.13 become

$$\begin{aligned}\Delta_{i_D}\Sigma_q\Delta_G I \xrightarrow{\cong} \Sigma_{q_0}\Delta_{i_Y}\Delta_G I = \Sigma_{q_0}\Delta_{G_0}\Delta_{i_X} I \qquad (28)\\ \xrightarrow{\delta} \Sigma_{q_0}\Delta_{G_0}\Gamma_X = \Sigma_{q_0}\Gamma_Y\\ = \Sigma_{q_0}\Delta_{q_0}\Gamma_D \xrightarrow{\epsilon_{q_0}} \Gamma_D\\ \Delta_{i_D}\Delta_F\Sigma_p I = \Delta_{F_0}\Delta_{i_C}\Sigma_p I \xrightarrow{\cong} \Delta_{F_0}\Sigma_{p_0}\Delta_{i_X} I \qquad (29)\\ \xrightarrow{\delta} \Delta_{F_0}\Sigma_{p_0}\Gamma_X = \Delta_{F_0}\Sigma_{p_0}\Delta_{p_0}\Gamma_C\\ \xrightarrow{\epsilon_{p_0}} \Delta_{F_0}\Gamma_C = \Gamma_D\end{aligned}$$

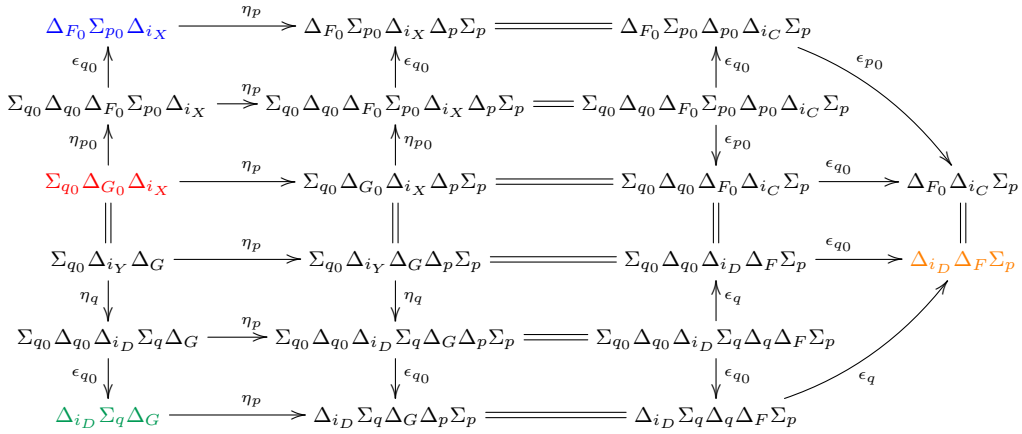
We need to show that the Beck-Chevalley isomorphism commutes with these maps to Γ_D. It suffices to show that that the following diagram commutes:



Indeed, the left-hand composite is the inverse to (28), the right-hand composite is the inverse to (29), the bottom map is the Beck-Chevalley isomorphism, and the top map relates $\Sigma_{q_0}\Delta_{G_0}$ to $\Delta_{F_0}\Sigma_{p_0}$, which completes the comparison between the two maps to Γ_D above.

Proving that the above diagram commutes may be much easier than what follows, which is quite unenlightening. It is mainly an exercise in finding something analogous

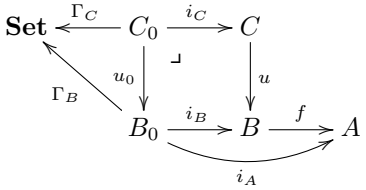
to a least common denominator in the square above. We include it for completeness.



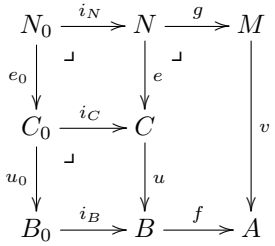
Every square in the above diagram clearly commutes, completing the proof.

□

Proposition E.2.5 (Typed distributivity)**.** *Suppose given the diagram of typed signatures*



such that u is a discrete op-fibration and the square is a pullback. Note that $\overline{u} = (u, u_0)$ is Σ-ready and that $\overline{f} = (f, \mathrm{id}_{B_0})$ is Π-ready. Construct the typed distributivity diagram



as follows. First form the distributivity diagram as in Proposition D.4.18, which is the big rectangle to the right. Now take N_0 to be the pullback of either the top-left square, the big-left rectangle, or the big outer square — all are equivalent. The functors v and e are discrete opfibrations. Note that $\overline{g} = (g, \mathrm{id}_{N_0})$ is Π-ready and $\overline{v} = (v, u_0 \circ e_0)$ is Σ-ready.

Then there is an isomorphism

$$\Sigma_{\overline{v}} \Pi_{\overline{g}} \Delta_{\overline{e}} \xrightarrow{\cong} \Pi_{\overline{f}} \Sigma_{\overline{u}}$$

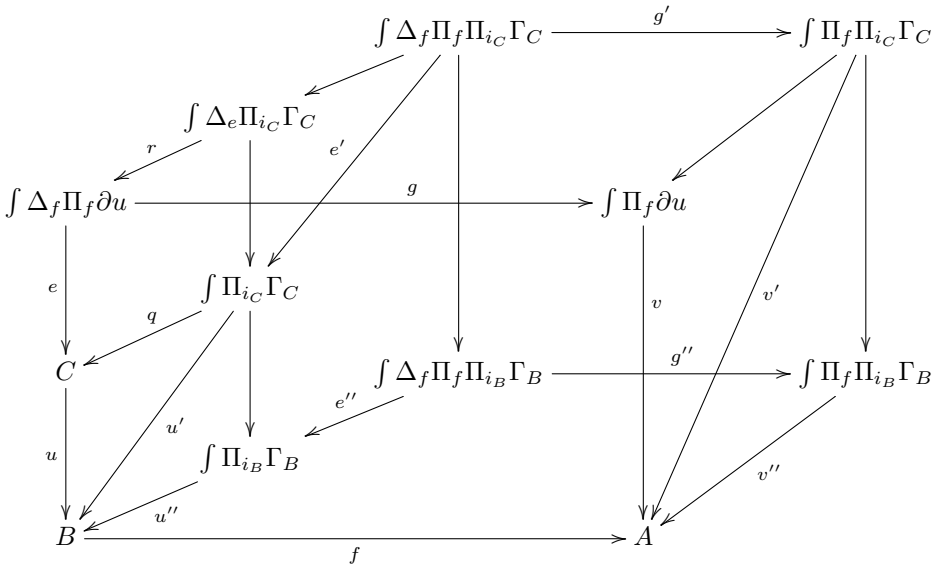
*of functors $\overline{C}$–**Inst** $\to \overline{A}$–**Inst**.*

Proof. By Proposition D.4.18 we have a distributivity isomorphism

$$\Sigma_v \Pi_g \Delta_e \xrightarrow{\cong} \Pi_f \Sigma_u \tag{30}$$

of functors C–**Inst** $\to$ A–**Inst**, but we want an isomorphism of functors $\overline{C}$–**Inst** $\to$ $\overline{A}$–**Inst**.

Suppose given a $\overline{C}$-instance $(I, \delta\colon \Delta_{i_C} I \to \Gamma_C)$. This is equivalent to a natural transformation $\delta\colon I \to \Pi_{i_C}\Gamma_C$ or equivalently a functor $\delta\colon \int \Pi_{i_C}\Gamma_C \to \mathbf{Set}$. The two sides of isomorphism (30) give rise to the same instance $A \to \mathbf{Set}$, but not a priori the same typed instance. A typed A-instance is a functor $\int \Pi_f \Pi_{i_B}\Gamma_B \to \mathbf{Set}$. It may be useful to find $\int \Pi_{i_C}\Gamma_C$ (middle vertex of left-hand square) and $\int \Pi_f \Pi_{i_B}\Gamma_B$ (bottom right back vertex) in the diagram below, which we will presently describe:



(31)

The front square is the (untyped) distributivity diagram of Proposition D.4.18; note that it is a pullback. The bottom square is our interpretation of $\Pi_{\overline{f}}$ from Remark E.1.11; note that it is a pullback. In the left-hand square we begin by form the little square in the front portion (the part that includes q, u, u', u'') as a pullback; see Remark E.1.14. Form the other little square as a pullback along e. Form the diagonal square as another distributivity diagram for Π_f of $\partial u'$. Note that we can complete the right-hand square by the functoriality of Π_f and that it is a pullback because Π_f preserves limits. Finally, form the back square as a pullback. It is now easy to see that all six sides are pullbacks.

Our only task is to show that the top square is a distributivity square. It is already a pullback, so our interest is in the top right corner: we need to show that there is a natural isomorphism

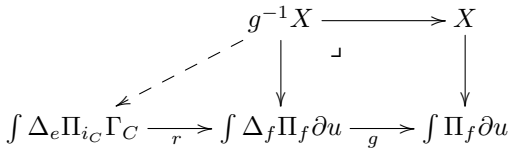
$$\Pi_f \Pi_{i_C} \Gamma_C = \Pi_f \partial u' \cong^? \Pi_g \partial r = \Pi_g \Delta_e \Pi_{i_C} \Gamma_C.$$

The adjunction isomorphism for $\Pi_g \partial r$ says that for any discrete opfibration $X \to \int \Pi_f \partial u$ there is an isomorphism

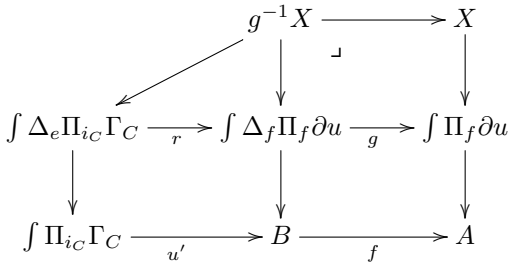
$$\mathrm{Hom}_{/\int \Delta_f \Pi_f \partial u}(g^{-1}X, \textstyle\int \Delta_e \Pi_{i_C} \Gamma_C) \cong \mathrm{Hom}_{/\int \Pi_f \partial u}(X, \textstyle\int \Pi_g \Delta_e \Pi_{i_C} \Gamma_C).$$

We thus need to show that there is a one-to-one correspondence between functors $X \to$

$\int \Pi_f \Pi_{i_C} \Gamma_C$) over $\int \Pi_f \partial u$ and dotted arrows in the diagram



Given a map $X \to \int \Pi_f \Pi_{i_C} \Gamma_C$), we can pull it back along g to get a map $g^{-1}X \to \int \Delta_f \Pi_f \Pi_{i_C} \Gamma_C$ because the top square in (31) is a pullback; this induces the required dotted arrow. Conversely, given a dotted arrow, we obtain the diagram



and because the diagonal diagram in (31) was constructed as a distributivity diagram, this induces a map $X \to \int \Pi_f \Pi_{i_C} \Gamma_C$ as desired.

Now that we have completed the aforementioned task, we are ready to prove the result. Our goal is as follows. We begin with an instance in the middle of the left square, a functor $\delta \colon \int \Pi_{i_C} \Gamma_C \to \mathbf{Set}$. We are interested in $\Pi_{\overline{f}} \Sigma_{\overline{u}} \delta$ and $\Sigma_{\overline{v}} \Pi_{\overline{g}} \Delta_{\overline{e}} \delta$; we consider them in this order.

By Remark E.1.14, we have a natural isomorphism

$$\Pi_{\overline{f}} \Sigma_{\overline{u}} \cong \Pi_f \Sigma_{u'}.$$

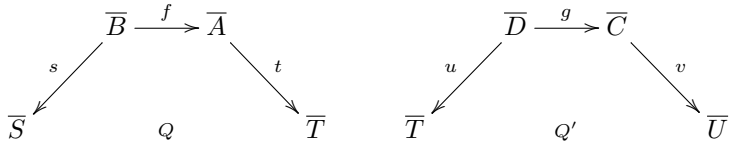
Because e is a discrete opfibration, Remarks E.1.8, E.1.11, and E.1.14, we have a natural isomorphism

$$\Sigma_{\overline{v}} \Pi_{\overline{g}} \Delta_{\overline{e}} \cong \Sigma_{v'} \Pi_{g'} \Delta_{e'}.$$

The result now follows from the usual distributive law, Proposition D.4.18.

□

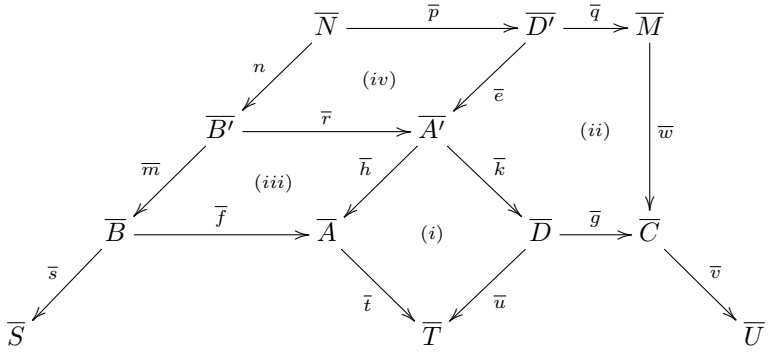
Theorem E.2.6. *Suppose that one has typed FQL queries $Q \colon \overline{S} \rightsquigarrow \overline{T}$ and $Q' \colon \overline{T} \rightsquigarrow \overline{U}$ as follows:*



(32)

Then there exists a typed FQL query $Q'' \colon \overline{S} \rightsquigarrow \overline{U}$ such that $[\![Q'']\!] \cong [\![Q']\!] \circ [\![Q]\!]$.

Proof. We will form Q'' by constructing the following diagram, which we will explain step by step:



(33)

We form the pullback (i); note that since $\overline{t}$ was Σ-ready, so is $\overline{k}$. Now form the typed distributivity diagram (ii) as in Proposition E.2.5; note that $\overline{w}$ is Σ-ready and $\overline{q}$ is Π-ready. Now form the typed comma categories (iii) and (iv) as in Proposition E.2.1. Note that $\overline{r}$ and $\overline{p}$ are Π-ready. The result now follows from Proposition E.2.4, E.2.5, and Corollary E.2.3,

□