Math. Struct. in Comp. Science (2014), vol. 24, iss. 6, e240602, 55 pages. © Cambridge University Press 2013
doi:10.1017/S0960129513000479

Database queries and constraints via lifting problems

DAVID I. SPIVAK[†]

Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139, United States of America
Email: dspivak@mit.edu

Received 16 June 2001; revised 8 October 2002

Previous work has demonstrated that categories are useful and expressive models for databases. In the current paper we build on that model, showing that certain queries and constraints correspond to lifting problems, as found in modern approaches to algebraic topology. In our formulation, each SPARQL graph pattern query corresponds to a category-theoretic lifting problem, whereby the set of solutions to the query is precisely the set of lifts. We interpret constraints within the same formalism, and then investigate some basic properties of queries and constraints. In particular, to any database π, we can associate a certain derived database $\mathbf{Qry}(\pi)$ of queries on π. As an application, we explain how giving users access to certain parts of $\mathbf{Qry}(\pi)$, rather than direct access to π, improves the ability to manage the impact of schema evolution.

1. Introduction

Diskin and Kadish (1994), Johnson (2001), Johnson *et al.* (2002) and many others have presented and investigated a tight connection between database schemas and the category-theoretic notion of sketches. This connection was taken further in Spivak (2012), where the existence of three data migration functors was shown to follow as a simple consequence of using categories rather than sketches to model schemas. In the current paper, we will show that a modern approach to the study of algebraic topology, *viz.* the *lifting problem* approach (Quillen 1967), provides an excellent model for typical queries and constraints (Prud'hommeaux and Seaborne 2008).

A database consists of a schema (a layout of tables in which *foreign key* columns connect one table to another) and an instance (the rows of actual data conforming to the chosen layout). One can picture the analogy between databases and topological spaces as follows. Imagine that a collection of data I and a schema S are each an abstract space, and suppose we have a projection from I to S. That is, we have some kind of continuous map $\pi : I \to S$ from a 'data bundle' I to a 'base space' S. Points in S represent tables, and paths in S represent foreign key columns (or iterates thereof), which point from one table to another. Over every point $s \in S$ in the base space, we can look at the corresponding fibre $\pi^{-1}(s) \subseteq I$ of the data bundle: this will correspond to the set of rows in table s. The map π associating data with the schema is called a *database instance*.

[†] This project was supported by ONR grant N000141010841.

A *query* on a database instance $\pi : I \to S$ is like a system of equations: it includes an organised collection of knowns and unknowns. In our model, a query takes the form of a functor $m : W \to R$ such that W (standing for WHERE-clause) corresponds to the set of knowns, each of which maps to a specific value in the data bundle I and such that the relationship between knowns and unknowns is captured in a schema R. More precisely, in (1), a query on the database instance $\pi : I \to S$ is presented as a commutative diagram to the left, which would be roughly translated into the pseudo-SQL to the right[†]:

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] & S \end{tikzcd} \qquad \begin{array}{ll} \text{SELECT} & * \\ \text{FROM} & R \xrightarrow{n} S \\ \text{WHERE} & R \xleftarrow{m} W \xrightarrow{p} I \end{array} \tag{1}$$

A *result* to the query is any mapping $\ell : R \to I$ making both triangles commute ($\ell \circ m = p$ and $\pi \circ \ell = n$) in the diagram

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] \arrow[ur, "\ell"] & S \end{tikzcd} \tag{2}$$

The map ℓ is called a *lift* of Diagram (1), hence the term *lifting problem*. The idea is that a lift is a way to fill the result schema R with conforming data from the instance π.

We will now give a simple example from algebraic topology to strengthen this image. By connecting databases and topology, we can not only visualise queries in a new way, but it is conceivable that algebraic topologists could use database interfaces to make computers work on lifting problems that arise in their research. Regardless of this, following the topological example, we will ground the discussion using an example database query.

Consider an empty sphere, defined by the equation

$$x^2 + y^2 + z^2 = 1,$$

and call it I. We project it down onto the (x, y)-coordinate plane ($z = 0$), and call that plane $S \cong \mathbb{R}^2$. The sphere I serves as the database instance and the plane S serves as the schema. A query consists of some result schema mapping to the plane S, say a solid disk R, given by

$$z = 0, \quad x^2 + y^2 \leqslant 1,$$

together with some constraints, say on the boundary circle W of the disk, given by

$$z = 0, \quad x^2 + y^2 = 1.$$

Graphically, we have Figure 1.

[†] A more general SQL query, with a specific SELECT statement, will be discussed in Example 4.8.

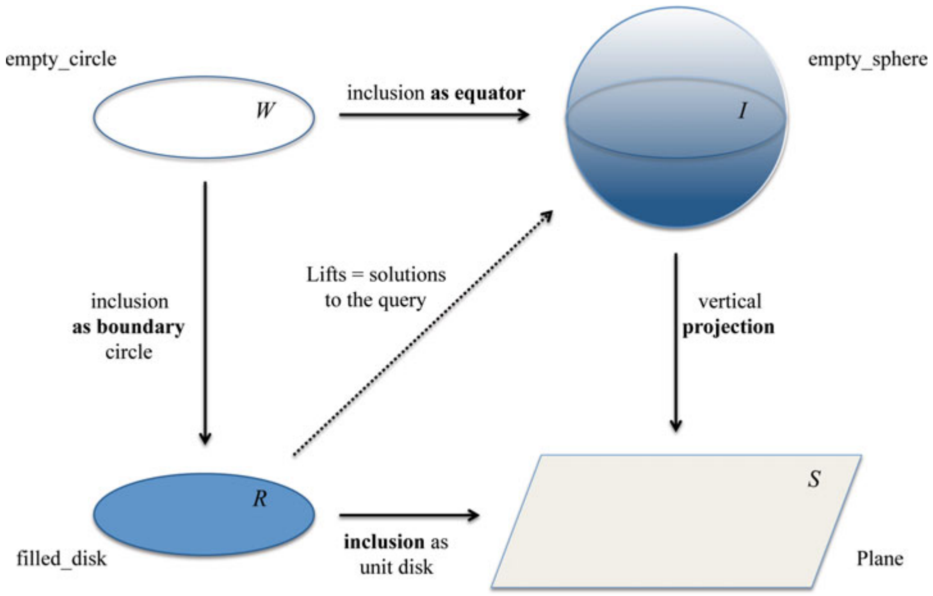


Fig. 1. (Colour online) A topological lifting problem

The results of the lifting query from Figure 1 are the mappings $R \to I$ making the diagram commute. Using (1) as a guide, the query would look something like

```
SELECT *
FROM    filled_disk inclusion
WHERE empty_circle as boundary = empty_circle as equator
```

Topologically, we can check that there are exactly two lifts – the top hemisphere and the bottom hemisphere – so our pseudo-SQL query above would return exactly two results.

1.1. *Main example of a lifting query*

We will now provide an example of a situation in which one may wish to query a database, and then show that this query naturally takes the structure of a lifting problem. We will break a single example into three parts for clarity.

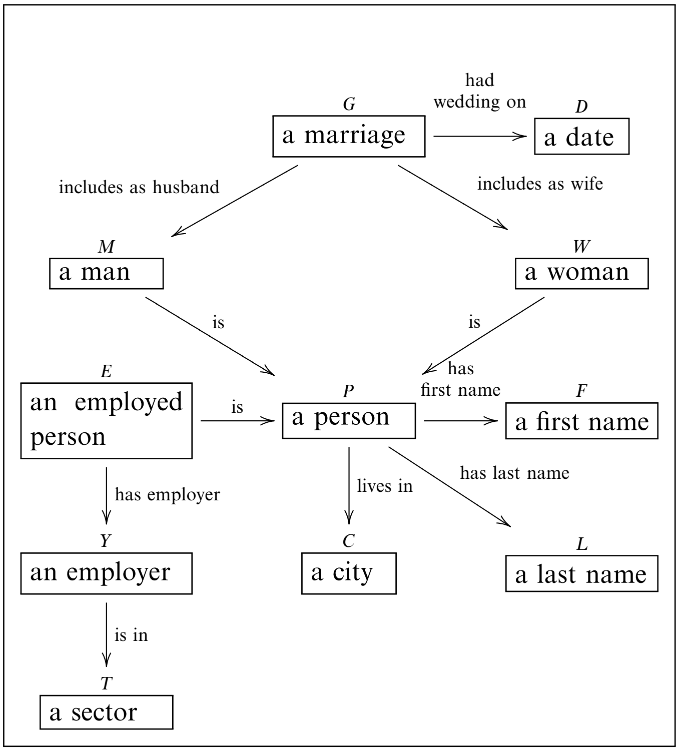
Example 1.1 (main example 1: situation, SPARQL and schema). Suppose we have just come home from a party where we met and really hit it off with a married couple. The husband's name is Bob and the wife's name is Sue, and they live in Cambridge. From the conversation, we know that Bob works at MIT and Sue works in the financial sector. We would like to see them again, but we somehow forgot to ask for their contact information – in particular, we would like to know their last names.

This is a typical database query problem. It can be phrased as the following SPARQL graph pattern query (which we arrange in two columns for readability and to save space):

```
(?marriage includesAsHusband ?b)    (?marriage includesAsWife ?s)
(?b hasFirstName Bob)               (?s hasFirstName Sue)
(?b livesIn Cambridge)              (?s livesIn Cambridge)
(?employedb is ?b)                  (?employeds is ?s)                (3)
(?employedb hasEmployer MIT)        (?employeds hasEmployer ?sueEmp)
                                    (?sueEmp isIn financial)
(?b hasLastName ?bobLast)           (?s hasLastName ?sueLast)
```

The query in (3) might be asked on the following database schema[†]:

$S :=$



(4)

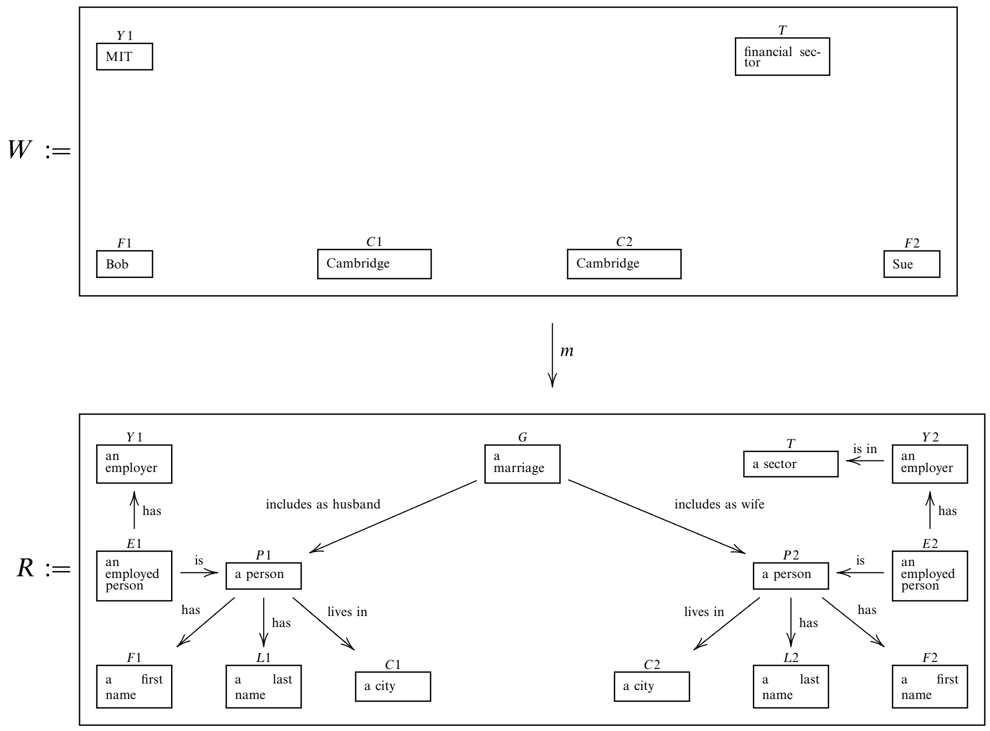
Given that S is instantiated with data $\pi : I \to S$, we can hope to find Bob and Sue, and then determine their last name. In the following two examples (Examples 1.2 and 1.3) we will show that this query corresponds to a lifting problem for π.

Example 1.2 (main example 2: WHERE-clause and result schema). In this example we consider the SPARQL query presented as (3) in Example 1.1, in which we wanted to find information about our new friends Bob and Sue. We will use a lifting problem to state this query, and to do this we will need to come up with a result schema R, a constraint schema (a set of knowns) W and a mapping $m : W \to R$ embedding the known objects

[†] The schema S in (4) deliberately includes a box D and an arrow $G \to D$ that are not part of our query (3).

into the result schema. In this example, we will present m, W and R, then, in Example 1.3, we will explain the lifting diagram for the query and show the results.

In order to find our friends Bob and Sue, we will use the following mapping:



The functor $m: W \to R$ sends each object in W to the object with the same label in R: for example, ⌜MIT⌝[†] in Ob(W) is sent to ⌜an employer⌝ in Ob(R) because they are both labelled $Y1$.

The following exercise will act as an aid to orientation. First count the number of constants in the SPARQL query (3) – there are 6 (such as Bob, Cambridge, and so on), and this is precisely the number of objects in W. Now count the combined number of constants and variables in the SPARQL query – there are 14 (there are 8 variables, such as ?marriage, ?empoyedb, and so on), and this is precisely the number of objects in R. Finally, count the number of triples in the SPARQL query – there are 13, and this is precisely the number of arrows in R. These facts are not coincidences.

Example 1.3 (main example 3: lifting diagram and result set). In Example 1.2 we showed a functor $m: W \to R$ corresponding to the SPARQL query stated in (3). In this example,

[†] We will use corner symbols around words in the running text of the paper when we refer to objects displayed as textboxes in diagrams. For example, we write ⌜MIT⌝ to refer to the object

$$\boxed{\text{MIT}}$$

we will explain how this query can be formulated as a lifting problem of the form

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd} \tag{5}$$

that serves to pose our query to the database instance π. At this point, we can ask for the set of solutions ℓ. So far, we have presented W, m, R and S, and assumed I and π, we will come to the set of ℓ's later, so we just need to present p and n now.

Reference should be made to our presentation of S in Example 1.1 (4) for the following. The functor $n\colon R \to S$ should be obvious from our labelling system (for example, the object E1=⌜an employed person⌝ in category R is mapped to the object E=⌜an employed person⌝ in category S). Note that, as applied to objects, n is neither injective nor surjective in this case:

$$n^{-1}(P) = \{P_1, P_2\}$$
$$n^{-1}(D) = \varnothing.$$

Suppose $\pi : I \to S$ is our data bundle, and assume that it contains enough data for the constants in the query to have unique referents[†]. There is an obvious functor $p\colon W \to I$ that sends each object in category W to its referent in I. For example, we assume that there is an object in I labelled ⌜MIT⌝, which is mapped to by the object Y1=⌜MIT⌝ in W.

Thus, our query from (3) is finally in the form of a lifting problem as in (5). We will show in Example 4.10, after we have built up the requisite theory, that the set of lifts can be collected into a single table, the most useful projection of which would look something like this:

Marriage								
ID	**Husband**				**Wife**			
	ID	**First**	**Last**	**City**	**ID**	**First**	**Last**	**City**
G3801	M881-36	Bob	Graf	Cambridge	W913-55	Sue	Graf	Cambridge

(6)

This concludes our tour of the main example, in which we have shown a typical query formulated as a lifting problem. The mathematical basis for the above ideas will be presented in Section 4.

1.2. *Relation to earlier work*

Aswe have already mentioned, there is a long history of applying a category-theoretic formalism to database theory. For the purposes of our exposition here, we will divide these approaches into two groupings. The first grouping, which includes the work in Tuijn and

[†] Note that the way we use the term 'query' here is not standard – see Sections 1.5.1 and 4.2 for an explanation.

Gyssens (1992) and Kato (1983), considers relational database tables as sets of attributes, using limits to discuss joins. This formulation is similar to that used in Spivak (2009), in which simplicial sets were used as a geometric model for 'sheaves of attributes'. The second grouping, which includes the work in Diskin and Kadish (1994), Johnson (2001) and Johnson *et al.* (2002), uses *sketches* in the sense of Ehresmann (1968). The latter approach is closer to that of Spivak (2012) and the current paper. We will now discuss the similarities to and differences between the sketch and lifting problems approaches.

In the current paper, we will model database schemas as finite category presentations (see Section 2.1), whereas the 'second grouping' of approaches mentioned above models them as sketches. A sketch is a category together with specified limit cones and colimit cones (Barr and Wells 2005). Sketches are more expressive than categories: for example, in the database context, using sketches allows a schema to convey when the set of rows in table T is the product of the sets of rows in tables U and V. This expressivity comes at some cost: while the categorical model in Spivak (2012) has three built-in data migration functors for moving data back and forth between schemas, the most general sketch model has only one, a 'pullback'. If the modelling is confined to the less expressive limit sketches, a left adjoint to this pullback becomes available. The point is that with the capacity to express more in a model, it seems the ability to transmit data to other models is reduced.

Still, it may be useful to find something in between sketches and bare categories, because using categories as models does not allow us to express constraints beyond foreign keys and commutative diagrams. For example, it does not allow for injectivity, or 'is a', constraints. It is here that the current paper fits in. Modern mathematical research, especially algebra and topology, has found surprisingly little use for sketches and sketch morphisms, despite their naturality and simplicity. It is not clear why this should be the case, especially given the success of the sketch model in applications (Barr and Wells 2005); the future of category theory in mathematics may indeed make more use of sketches.

What we can say is that modern mathematical research has become deeply invested in categories and functors. Further, algebraic topology, which was the trailblazer for category theory, has for more than half a century found lifting problems to be a key tool for investigating abstract spaces. In this paper we make the connection between lifting problems and database theory. As mentioned above, we show that there are many constraints that are well phrased as lifting problems, and that queries also fit nicely into this framework.

Sketches are often divided into the following three types: limit sketches; colimit sketches; and mixed sketches (that is, granting the architect the ability to impose limits, colimits, or both). Lifting constraints cover the expressivity of limit sketches fully, and then a bit more – see Section 3.5. In particular, lifting problems can enforce injectivity constraints, as shown in Example 3.13. However, colimit sketches can express things that lifting constraints cannot. For example, with colimit sketches we can express set-theoretic complements, and this cannot be done with lifting problems. The ability to enforce the fact that one subset is the complement of another (also known as negation) comes with well-known problems, such as domain dependence. However, there are certain real-world applications in which colimit sketches are simply unavoidable: for example, this is even the case for something as simple as the data model for 3-packs of toothbrushes.

What we can take away from the current section is that lifting problems can express a different class of constraints from that expressible by sketches. We contend here that it is a valuable class of constraints, and that it corresponds quite well with SPARQL graph pattern queries. We will summarise our argument for the usefulness of lifting constraints in Section 3.3.

1.3. *Purpose of the paper*

The purpose of this paper is to:

— provide an efficient mathematical formulation of common database queries (modelling both SQL and SPARQL styles);

— attach a geometric image to database queries that can be useful in conceptualisation; and

— explore the theory and applications of the derived database schema $\mathbf{Qry}(\pi)$ of queries on a database instance π, and the derived instance of results.

We include several mathematical results that are well known to experts in order to help those interested in using this paper to bridge the gap between database theory and category theory.

1.4. *Plan of the paper*

We begin in Section 2 with a review of the categorical approach to databases – see Spivak (2012) for more details. Roughly speaking, this correspondence goes by the slogan ‘schemas are categories; instances are set-valued functors’. In Section 2.3, we also discuss the Grothendieck construction, which will be crucial for our approach: a database instance can be converted into a *discrete opfibration*, which we will later use extensively to make the parallel with algebraic topology, and lifting problems in particular.

In Section 3, we define constraints on a database in terms of lifting conditions and discuss some constraint implications. We give several examples to show how various common existence and uniqueness constraints (such as the constraint that a given foreign key column is surjective) can be framed in the language of lifting conditions.

In Section 4, we discuss queries as lifting problems, and review the paper’s main example.

In Section 5, we show that the information in a given database instance can be collected into a new, derived database. This derived database of queries and their results can be queried, giving rise to nested queries. We explain how this formulation can be useful for managing the impact of schema evolution.

Finally in Section 6, we briefly discuss some possible directions for future work, including tying our approach into homotopy type theory (in the sense of Awodey and Warren (2009) and Voevodsky (2006)) and other projects.

1.5. *Notation and terminology*

For any natural number $n \in \mathbb{N}$, we use $\underline{n}$ to denote the set $\{1, 2, \ldots, n\}$, and sometimes regard sets as discrete categories without mentioning it. Note that $\underline{0} = \varnothing$. We use $[n]$ to denote the linear order $0 \leqslant 1 \leqslant \ldots \leqslant n$, and sometimes regard orders as categories without mentioning that either. In particular, $\underline{1}$ is the terminal category: it has one object and one morphism (the identity).

Given any category $\mathcal{C}$, we denote the category of all functors $\mathcal{C} \to \mathbf{Set}$ by $\mathcal{C}$–$\mathbf{Set}$. The terminal object in $\mathcal{C}$–$\mathbf{Set}$ sends each object in $\mathcal{C}$ to $\underline{1}$, and we denote it by $\underline{1}^{\mathcal{C}} : \mathcal{C} \to \mathbf{Set}$. For any category $\mathcal{C}$, there is a one-to-one correspondence between the objects in $\mathcal{C}$ and the functors $\underline{1} \to \mathcal{C}$, so we may denote an object $c \in \mathrm{Ob}(\mathcal{C})$ by a functor $\underline{1} \xrightarrow{c} \mathcal{C}$. In particular, we elide the difference between a set and a functor $\underline{1} \to \mathbf{Set}$.

We draw schemas in two different ways, depending on context. When we wish to save space, we draw objects as nodes with simple labels and morphisms as arrows with simple labels. But, when we wish to be more expressive, we draw objects as text boxes and put as much text in them (and on each arrow) as needed for clarity (see Spivak and Kent (2012)). For example, we might draw the indexing category for directed graphs in either of the following two ways:

$$\begin{tikzcd}
\overset{E}{\bullet} \arrow[r, bend left, "s"] \arrow[r, bend right, "t"'] & \overset{V}{\bullet}
\end{tikzcd}
\qquad
\begin{tikzcd}
\boxed{\text{an edge}} \arrow[r, bend left, "\text{has as source}"] \arrow[r, bend right, "\text{has as target}"'] & \boxed{\text{a vertex}}
\end{tikzcd}$$

(See the footnote in Example 1.2 for the representation we use in running text for objects displayed as textboxes in diagrams.)

Given two categories, there are generally many functors from one to the other. However, if the objects and arrows are labelled coherently, there are many fewer functors that roughly respect the labellings. We will usually be explicit when defining functors, but we will also take care that our functors respect labelling to the greatest extent possible.

1.5.1. *'Queries on a database'.*

In the wide-spread terminology for database queries, a 'query' cannot depend on the current instance π of the database, but instead only on the schema S. This is perfectly reasonable for theoretical and practical reasons. In applications, however, we often use what is known as a *cursor*, which is basically a pre-defined query consisting of a join-graph and a set of variables to be bound at run-time. With respect to the diagram

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S
\end{tikzcd}$$

the join-graph is R, the set of variables waiting to be bound is W and the binding itself is $p : W \to I$. The mathematics will be covered more extensively in Section 4.2, but in the rest of this section we hope to get across how our use of the term ‘query’ in the current paper might be connected to common ideas in database systems.

In applications, a query wizard may run the cursor in a two-step query process. First it will query the database to offer the user a drop-down menu of choices in the active domain of each variable. The user will then choose a row to which the variable will be bound (once for each variable). At this point, the program will apply the actual query declared by the cursor. This two-step process corresponds to searching for possible functors $p\colon W \to I$ and then searching for lifts ℓ.

Throughout the current paper, when we refer to queries on a database, we mean queries for which the constant variables have been bound to elements in the active domain of a given instance. However, as we will see in Section 4.2, we can also use the same machinery in cursor-like fashion to pose queries in which variable values have been chosen without regard for whether they are in the active domain or not. In other words, we will show that what can be accomplished by queries in the sense of traditional relational database theory fits easily into our framework. Because it works either way, we use the unusual terminology ‘queries on a database’ since it neither lulls readers into thinking that these gadgets are completely instance independent, nor frightens them into thinking that the instance must be known in advance for the ideas here to work.

2. Elementary theory of categorical databases

2.1. *Review of the categorical description of databases*

The basic mantra for the categorical description of databases is that a database schema is a small category S and an instance is a functor $\delta : S \to \mathbf{Set}$, where **Set** is the category of sets[†]. In recalling these ideas, we will borrow liberally from Spivak (2012), where further details and clarification can be found if required. Anyone familiar with the basic setup and data migration functors can skip to Section 2.3.

Spivak (2012) defined a category **Sch** of categorical schemas and translations and proved an equivalence of categories

$$\mathbf{Sch} \simeq \mathbf{Cat}, \tag{7}$$

where **Cat** is the category of small categories. The difference between **Sch** and **Cat** is that an object of the former is a *chosen presentation* of a category using generators and relations, as described below. Given the equivalence (7), we can and will elide the difference between schemas and small categories.

Roughly speaking, a schema S consists of a graph G together with an equivalence relation on the set of paths of G. Each object $s \in \mathrm{Ob}(S)$ represents a table (or, more precisely, the ID column of a table), and each arrow $s \to t$ emanating from s represents a column of table s, taking values in the ID column of table t. The following example should clarify these ideas.

[†] If preferred, **Set** can be replaced by the category of finite sets or by the category Types for some λ-calculus.

Example 2.1. As a typical database example, consider the bookkeeping necessary to run a department store. We keep track of a set of employees and a set of departments. For each employee e, we keep track of:

E1 : the **first** name of e, which is a `FirstNameString`;
E2 : the **last** name of e, which is a `LastNameString`;
E3 : the **manager** of e, which is an `Employee`; and
E4 : the department that e **works in**, which is a `Department`.

For each department d, we keep track of:

D1 : the **name** of d, which is a `DepartmentNameString`; and
D2 : the **secretary** of d, which is an `Employee`.

Suppose further that we adopt the following two rules:

Rule 1 : For every employee e, the **manager** of e **works in** the same department that e **works in**.
Rule 2 : For every department d, the **secretary** of d **works in** department d.

This is all captured neatly, with nothing left out and nothing added, by the category presented below:

$$S := \boxed{\begin{array}{c} \underline{\texttt{Employee manager worksIn} \simeq \texttt{Employee worksIn}} \\ \underline{\texttt{Department secretary worksIn} \simeq \texttt{Department}} \\ \begin{tikzcd} & \underset{\bullet}{\texttt{Employee}} \arrow[loop above, "\text{manager}"] \arrow[rr, shift left, "\text{worksIn}"] \arrow[ld, "\text{first}"'] \arrow[rd, "\text{last}"] & & \underset{\bullet}{\texttt{Department}} \arrow[ll, shift left, "\text{secretary}"] \arrow[d, "\text{name}"] \\ \underset{\bullet}{\texttt{FirstNameString}} & & \underset{\bullet}{\texttt{LastNameString}} & \underset{\bullet}{\texttt{DepartmentNameString}} \end{tikzcd} \end{array}} \qquad (8)$$

The underlined statements at the top indicate pairs of commutative (that is, equivalent) paths: each path is indicated by its source object followed by the sequence of arrows that composes it. The objects, arrows and equivalences in S correspond to the tables, columns and rules laid out at the beginning of this example.

The collection of data on a schema is typically presented in table form. The following tables show how a database with schema S might look at a particular moment in time.

Employee				
ID	**first**	**last**	**manager**	**worksIn**
101	David	Hilbert	103	q10
102	Bertrand	Russell	102	x02
103	Alan	Turing	103	q10

Department		
ID	**name**	**secretary**
q10	Sales	101
x02	Production	102

FirstNameString
ID
Alan
Alice
Bertrand
Carl
David
⋮

LastNameString
ID
Arden
Hilbert
Jones
Russell
Turing
⋮

DepartmentNameString
ID
Marketing
Production
Sales
⋮

(9)

Every table has an ID column, and in every table each cell references a cell in the ID column of some table. For example, cells in the secretary column of the `Department` table refer to cells in the ID column of the `Employee` table. Finally, we can check that Rules 1 and 2 hold. For example, let e be Employee 101. He works in Department q10 and his manager is Employee 103. Employee 103 works in Department q10 as well, as required. The point is that the data in the tables in (9) conform precisely to the schema S from Diagram (8).

A set of tables that conforms to a schema is called an *instance* of that schema. We will denote the set of tables in (9) by δ (we noted above that δ conforms with, and is thus an instance of, schema S). Mathematically, δ can be modelled as a functor

$$\delta : S \to \mathbf{Set}.$$

To each object $s \in S$, the instance δ assigns a set of row-IDs, and to each arrow $f : s \to t$ in S, it assigns a function, as specified by the cells in the f-column of s.

2.2. *Review of data migration functors*

Once we realise that a database schema can be captured simply as a category S and each instance on S as a set-valued functor $\delta : S \to \mathbf{Set}$, classical category theory provides some ready-made tools for migrating data between different schemas. We begin by defining a schema mapping.

Definition 2.2. Let S and T be schemas (that is, small categories). A *schema mapping* is a functor $F : S \to T$.

Thus, a schema mapping assigns to each table in S a table in T, to each column in S a column in the corresponding table of T, and all of this in such a way that the path equivalence relation is preserved.

Definition 2.3. A schema mapping $F : S \to T$ induces three functors on instance categories, which we call *the data migration functors associated with* F, and which we denote by Σ_F, Δ_F and Π_F, as follows:

$$\begin{tikzcd} S\text{–}\mathbf{Set} \arrow[r, bend left, "\Sigma_F"] \arrow[r, bend right, "\Pi_F"'] & T\text{–}\mathbf{Set}. \arrow[l, "\Delta_F" description] \end{tikzcd}$$

The functor

$$\Delta_F : T\text{–}\mathbf{Set} \to S\text{–}\mathbf{Set}$$

sends an instance

$$\delta : T \to \mathbf{Set}$$

to the instance

$$\delta \circ F : S \to \mathbf{Set}.$$

The functor Σ_F is the left adjoint to Δ_F, and the functor Π_F is the right adjoint to Δ_F. We call:

— Δ_F the *pullback along* F;
— Σ_F the *left pushforward along* F; and
— Π_F the *right pushforward along* F.

The functors Δ_F, Σ_F and Π_F are well known in the category theory literature, where the latter two are often referred to as the *left Kan extension along* F and the *right Kan extension along* F (Mac Lane 1988, Chapter X). In databases, the left pushforward Σ_F will generally correspond to unions and quotients, and the right pushforward Π_F will generally correspond to products and joins. We will explore these ideas a bit further in Section 5 – see Spivak (2012) for further explanation.

2.3. *RDF via the Grothendieck construction*

There is a well-known construction that associates with a functor $\delta : S \to \mathbf{Set}$, a pair $(\int(\delta), \pi_\delta)$, where $\int(\delta) \in \mathbf{Cat}$ is a new category called *the category of elements of* δ, and $\pi_\delta : \int(\delta) \to S$ is a functor. The pair $(\int(\delta), \pi_\delta)$ is often called the *Grothendieck construction* of δ. The objects and morphisms of $\int(\delta)$ are given as follows:

$$\mathrm{Ob}(\textstyle\int(\delta)) := \Big\{(s, x) \mid s \in \mathrm{Ob}(S), x \in \delta(s)\Big\}$$

$$\mathrm{Hom}_{\int(\delta)}((s, x), (s', x')) := \Big\{f : s \to s' \mid \delta(f)(x) = x'\Big\}.$$

The functor $\pi_\delta : \int(\delta) \to S$ is straightforward: it sends the object (s, x) to s and sends the morphism $f : (s, x) \to (s', x')$ to $f : s \to s'$.

We call the pair $(\int(\delta), \pi_\delta)$ the *discrete opfibration associated with* δ. We will see in the next section (Definition 3.6) that π_δ is indeed a kind of fibration of categories. This construction, and, in particular, the category $\int(\delta)$, is also nicely connected with the *resource descriptive framework* (Prud'hommeaux and Seaborne 2008), in which data is captured in *RDF triples*. Indeed, the arrows $\overset{s}{\bullet} \xrightarrow{p} \overset{b}{\bullet}$ of $\int(\delta)$ correspond one-for-one with these RDF

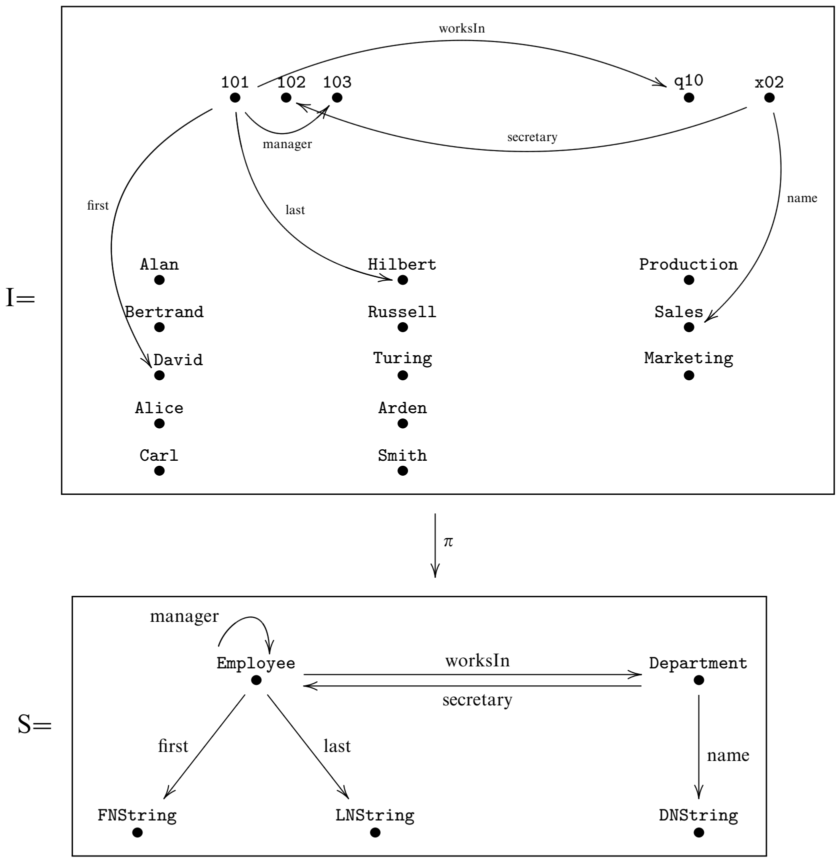


Fig. 2. An example of the Grothendieck construction, or category of elements, of a functor $\delta : S \to \mathbf{Set}$. The functor $\pi : I \to S$ sends objects 101,102,103 in I to the object `Employee`, in S; similarly, it sends the arrow labelled worksIn in $\int(\delta)$ to the arrow labelled worksIn in S, and so on. Note that there are 16 non-ID cells in the tables in (9), which represents our instance δ, but, to aid readability, we have only drawn six arrows in $I = \int(\delta)$ and omitted the other ten – such as the arrow $\overset{\texttt{102}}{\bullet} \xrightarrow{\text{Last}} \overset{\texttt{Russell}}{\bullet}$. The point is that the RDF triple store associated with instance δ is nicely represented using the standard Grothendieck construction. For example, the arrow $\overset{\texttt{101}}{\bullet} \xrightarrow{\text{first}} \overset{\texttt{David}}{\bullet}$ represents the RDF triple (101 :first David).

triples (**s**ubject, **p**redicate, o**bj**ect). In this way, we have shown a ready-made conversion from relational databases to RDF triple stores through the Grothendieck construction. The following example should clarify this discussion.

Example 2.4. Recall the database instance $\delta : S \to \mathbf{Set}$ given by the tables in (9), whose schema S was presented as Diagram (8). Applying the Grothendieck construction to $\delta : S \to \mathbf{Set}$, we get a category $I := \int(\delta)$ and a functor $\pi := \pi_\delta$ as in Figure 2.

In Section 1, when we discussed database instances in terms of mappings π, each from a data bundle I to a base space S, we were referring to exactly the discrete opfibration picture in Figure 2.

In Section 3.2, we will give a definition of discrete opfibrations in terms of lifting constraints (Definition 3.6). However, we will first attempt to understand a discrete opfibration $\pi : I \to S$ by considering its various fibres and their relationships. More precisely, given an object $s \in \mathrm{Ob}(S)$, we consider the fibre $\pi^{-1}(s)$, and given a morphism $f : s \to s'$ in S we consider how the fibres $\pi^{-1}(s)$ and $\pi^{-1}(s')$ are related.

If $\pi : I \to S$ were not assumed to be a discrete opfibration, but just a general functor, all we would know about these various fibres would be that they were categories. But the first distinctive feature of a discrete opfibration is that the fibre $\pi^{-1}(s)$ is a *discrete category*, that is, a set, for each object $s \in S$: in other words, there are no morphisms between different objects in a chosen fibre (see Proposition 3.7). The pre-image $\pi^{-1}(f)$ of $f : s \to s'$ is a set of morphisms from objects in $\pi^{-1}(s)$ to objects in $\pi^{-1}(s')$. When π is a discrete opfibration, there exists a unique morphism in $\pi^{-1}(f)$ emanating from each object in $\pi^{-1}(s)$, so the subcategory

$$\pi^{-1}(f) \subseteq I$$

can be cast as a single function

$$\pi^{-1}(f) : \pi^{-1}(s) \to \pi^{-1}(s').$$

To recap, the discrete opfibration $\pi_\delta : \int(\delta) \to S$ of a set-valued functor $\delta : S \to \mathbf{Set}$ contains the same information as δ does, but from a different perspective. We have

$$\pi_\delta^{-1}(s) \cong \delta(s)$$
$$\pi_\delta^{-1}(f) \cong \delta(f)$$

for any $s, s' \in \mathrm{Ob}(S)$ and $f : s \to s'$.

2.3.1. *Basic behaviour of the Grothendieck construction.* In this section we present some simple results for the Grothendieck construction, all of which are well known.

Proposition 2.5. Let $\delta : S \to \mathbf{Set}$ be a functor. Then the Grothendieck construction $\int(\delta) \xrightarrow{\pi_\delta} S$ of δ can be described as a pullback in the diagram of categories

$$\begin{tikzcd}
\int(\delta) \arrow[r] \arrow[d, "\pi_\delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{Set}_* \arrow[d, "\pi"] \\
S \arrow[r, "\delta"'] & \mathbf{Set},
\end{tikzcd}$$

where $\mathbf{Set}_*$ is the category of pointed sets and π is the functor that sends a pointed set $(X, x \in X)$ to its underlying set X.

Proof. The statement follows directly from the definitions. □

Lemma 2.6. Let S be a category. The functor

$$\int : S\text{–}\mathbf{Set} \to \mathbf{Cat}_{/S}$$

is fully faithful. That is, given two instances, $\delta, \epsilon : S \to \mathbf{Set}$, there is a natural bijection

$$\mathrm{Hom}_{S\text{-}\mathbf{Set}}(\delta, \epsilon) \xrightarrow{\cong} \mathrm{Hom}_{\mathbf{Cat}_{/S}}(\textstyle\int(\delta), \int(\epsilon)).$$

Proof. The statement follows directly from the definitions. □

Proposition 2.7. Let $F : S \to T$ be a functor. Let $\delta : S \to \mathbf{Set}$ and $\epsilon : T \to \mathbf{Set}$ be instances. Suppose we have a commutative diagram

$$\begin{tikzcd} \int(\delta) \arrow[r] \arrow[d, "\pi_\delta"'] & \int(\epsilon) \arrow[d, "\pi_\epsilon"] \\ S \arrow[r, "F"'] & T. \end{tikzcd} \tag{10}$$

Then diagram (10) is a pullback, that is,

$$\int(\delta) \cong S \times_T \int(\epsilon)$$

if and only if

$$\delta \cong \Delta_F \epsilon.$$

Proof. It is easy to check the statement by comparing the set of objects and the set of morphisms in $\int(\delta)$ with the respective sets in $S \times_T \int(\epsilon)$. □

2.3.2. *Examples from algebraic topology.* In algebraic topology (May 1999), we associate with every topological space X, a fundamental groupoid $Gpd(X)$. This is a category whose objects are the points of X and whose set of morphisms between two objects is the set of (equivalence classes of) continuous paths in X from one point to the other. Two paths in X are considered equivalent if one can be deformed to the other (without any part of it leaving X). Composition of morphisms is given by concatenation of paths.

Some of the study of a space X can be reduced to the study of this algebraic object $G = Gpd(X)$, and the latter is well suited for translation to the language of the current paper.

Example 2.8. Suppose G is a groupoid. Then a covering of groupoids in the sense of May (1999, Section 4.3) is precisely the same as a surjective discrete opfibration with schema G.

Let $G = Gpd(S^1)$ denote the fundamental groupoid of the circle with circumference 1. Explicitly, we have

$$\mathrm{Ob}(G) = \{\theta \in \mathbb{R}\}/\sim,$$

where $\theta \sim \theta'$ if $\theta - \theta' \in \mathbb{Z}$. We also have

$$\mathrm{Hom}_G(\theta, \theta') = \{x \in \mathbb{R} \mid x + \theta \sim \theta'\}.$$

Think of G as the category whose objects are positions of a clock hand, and whose morphisms are arbitrary durations of time (rotating the hands from one clock position around to another). Consider the functor $T : G \to \mathbf{Set}$ such that

$$T(\theta) = \{t \in \mathbb{R} \mid t - \theta \in \mathbb{Z}\}$$

and such that for $x \in \mathrm{Hom}_G(\theta, \theta')$ we put

$$T(x)(t) = x + t.$$

So, for a clock position θ, the functor T returns all points in time at which the clock is in position θ.

Applying the Grothendieck construction to T, we get a covering

$$\pi : \int (T) \to G,$$

which corresponds to the universal cover of the circle S^1. It may be thought of as a helix (modelling the time line) mapping down to the circle (modelling the clock).

A much more sophisticated example relating databases to classical questions in algebraic topology can be found in Morava (2012).

3. Constraints via lifting conditions

In this section we introduce the lifting problem approach to database constraints. Roughly speaking, we will apply the same model to database queries in the next section, the idea being that a lifting constraint is a lifting query that is guaranteed to have a result.

3.1. *Basic definitions*

Definition 3.1. Let $S \in \mathbf{Cat}$ be a database schema. A (*lifting*) *constraint on* S is a pair (m, n) of functors

$$W \xrightarrow{m} R \xrightarrow{n} S.$$

A functor $\pi : I \to S$ is said to *satisfy the constraint* (m, n) if, for all solid arrow commutative diagrams of the form

$$\begin{tikzcd} W \arrow[r] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] \arrow[ur, dashed] & S \end{tikzcd} \tag{11}$$

there exists a dotted arrow lift making the diagram commute.

A (*lifting*) *constraint set* is a set

$$\xi := \{W_\alpha \xrightarrow{m_\alpha} R_\alpha \xrightarrow{n_\alpha} S \mid \alpha \in A\},$$

for some set A. A functor $\pi : I \to S$ is said to *satisfy the constraint set* ξ if it satisfies each constraint (m_α, n_α) in ξ.

Given a constraint set ξ on S, we say that a constraint $W \xrightarrow{m} R \xrightarrow{n} S$ is *implied by* ξ if whenever a functor $\pi : I \to S$ satisfies ξ, it also satisfies (m, n).

Remark 3.2. Although not all constraints on databases are lifting constraints (for example, declaring a table to be the union of two others is not expressible by a lifting constraint), lifting constraints are the only type of constraint we will consider in this paper. For this reason, we will often omit the word 'lifting', as suggested by the parentheses in Definition 3.1.

Example 3.3. Consider the schema

$$\mathcal{G} = \boxed{\begin{tikzcd} \overset{E}{\bullet} \arrow[r, bend left, "s"] \arrow[r, bend right, "t"'] & \overset{V}{\bullet} \end{tikzcd}}$$

The category $\mathcal{G}$–**Set** is precisely the category of (directed) graphs. Given a graph $X : \mathcal{G} \to$ **Set**, we have a function

$$X(s) : X(E) \to X(V)$$

assigning to every edge its source vertex. Suppose we want to declare this function to be surjective, meaning that every vertex in X is the source of some edge. We can do this with the lifting constraint

$$\boxed{\overset{V}{\bullet}} \xrightarrow{m} \boxed{\begin{tikzcd} \overset{E}{\bullet} \arrow[r, "s"] & \overset{V}{\bullet} \end{tikzcd}} \xrightarrow{n} \boxed{\begin{tikzcd} \overset{E}{\bullet} \arrow[r, bend left, "s"] \arrow[r, bend right, "t"'] & \overset{V}{\bullet} \end{tikzcd}}$$

where m and n respect labelling. A graph $\delta : \mathcal{G} \to$ **Set** has the desired property, *viz.* that every vertex is a source if and only if $\int(\delta)$ satisfies the lifting constraint (m, n).

Definition 3.4. Let $S \in$ **Cat** be a schema. Given a functor $m : W \to R$, we define a set $\langle m \rangle$ of lifting constraints by

$$\langle m \rangle = \left\{ W \xrightarrow{m} R \xrightarrow{n} S \mid n \in \mathrm{Hom}_{\mathbf{Cat}}(R, S) \right\}.$$

(Note that there is a bijection $\langle m \rangle \cong \mathrm{Hom}_{\mathbf{Cat}}(R, S)$, but the form of the set $\langle m \rangle$ allows us to apply Definition 3.1.) Given a set of functors

$$M = \{m_j : W_j \to R_j \mid j \in J\},$$

the union

$$\langle M \rangle := \bigcup_{j \in J} \langle m_j \rangle$$

is a constraint set, which we call the *universal constraint set generated by* M. A functor $\pi : I \to S$ satisfying the constraint set $\langle M \rangle$ is called an M-fibration. We say that elements of M are *generating constraints* for M-fibrations.

Remark 3.5. Universal constraint sets seem to be more important in traditional mathematical contexts than in 'informational' or database contexts. For example, in the world of simplicial sets, the Kan fibrations are M-fibrations for some universal constraint set $\langle M \rangle$, which is called *the set of generating acyclic cofibrations* (Hirschhorn 2003).

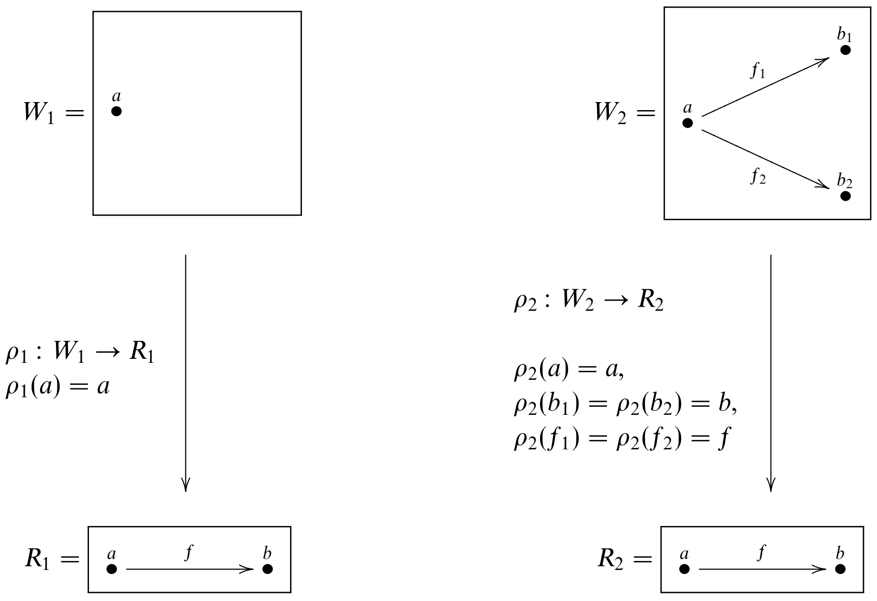


Fig. 3. The generating constraints ρ_1 and ρ_2 for discrete opfibrations

3.2. *Discrete opfibrations via lifting constraints*

In this section we will express the notion of a discrete opfibration in terms of lifting constraints. In other words, we will exhibit a finite set of functors

$$\{m_\alpha : W_\alpha \to R_\alpha\}_{\alpha \in A}$$

that serve to 'check' whether an arbitrary functor $\pi : I \to S$ is a discrete opfibration. In fact, Definition 3.6 will define π to be a discrete opfibration if and only if it is a $\{\rho_1, \rho_2\}$-fibration, where $\rho_1 : W_1 \to R_1$ and $\rho_2 : W_2 \to R_2$ are the functors shown in Figure 3.

Definition 3.6. Let I and S be categories and $\pi : I \to S$ be a functor. Then I is a *discrete opfibration* if it satisfies the lifting constraints ρ_1 and ρ_2 in Figure 3. That is, for any pair of horizontal maps $W_1 \to I$ and $R_1 \to S$ (respectively, for any pair of horizontal maps $W_2 \to I$ and $R_2 \to S$),

$$\begin{tikzcd} W_1 \arrow[r, "\forall"] \arrow[d, "\rho_1"'] & I \arrow[d, "\pi"] \\ R_1 \arrow[r, "\forall"'] \arrow[ur, dashed, "\exists"] & S \end{tikzcd} \qquad \begin{tikzcd} W_2 \arrow[r, "\forall"] \arrow[d, "\rho_2"'] & I \arrow[d, "\pi"] \\ R_2 \arrow[r, "\forall"] \arrow[ur, dashed, "\exists"] & S \end{tikzcd}$$

there exists a dotted arrow functor, as shown, such that the full diagram commutes.

Let $\pi : I \to S$ be a $\{\rho_1, \rho_2\}$-fibration. Then, for any functor $R_1 = R_2 \to S$, that is, for any arrow $f : s \to s'$ in S, we have two lifting conditions. A good way to understand the conditions of Definition 3.6 is that for any object $x \in \pi^{-1}(s)$ in the fibre over s,

(1) there exists at least one arrow in I, emanating from x, whose image under π is f; and
(2) there exists at most one arrow in I, emanating from x, whose image under π is f.

In the remainder of this section we will give some consequences of Definition 3.6.

Proposition 3.7. Let $\pi : I \to S$ be a discrete opfibration. Then for each object $s \in \mathrm{Ob}(S)$ the fibre $\pi^{-1}(s)$ is a discrete category.

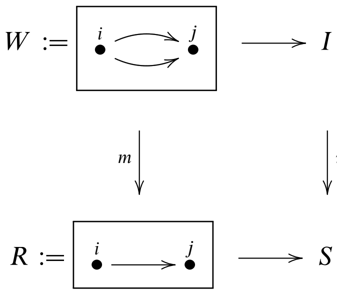
Proof. Let $s \in \mathrm{Ob}(S)$ be an object and $g : x \to y$ be a morphism in the fibre $\pi^{-1}(s) \subseteq I$. We will show that $x = y$ and that $g = \mathrm{id}_x$ is the identity morphism. Consider the map $\rho_2 : W_2 \to R_2$ in Figure 3. Let $n : R_2 \to S$ be the functor sending f to id_s. Let $p : W_2 \to I$ send f_1 to id_x and send f_2 to g. We now have a lifting diagram as in Definition 3.6, so a lift is guaranteed. This lift equates id_x and g. □

Proposition 3.8. If $\pi : I \to S$ is a discrete opfibration, then π is faithful. In other words, for any two objects $i, j \in \mathrm{Ob}(I)$, the function

$$\pi : \mathrm{Hom}_I(i, j) \to \mathrm{Hom}_S(\pi(i), \pi(j))$$

is injective.

Proof. To prove that π is faithful, we only need to find a solution for each lifting diagram of the form



We can extend this diagram on the left with either surjective map from the relational constraint functor ρ_2 (see Figure 3) to m, as indicated in the diagram

$$\begin{tikzcd}
W_2 \arrow[r] \arrow[d, "\rho_2"'] & W \arrow[r] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R_2 \arrow[urr, dashed] & R \arrow[l, Rightarrow] \arrow[r] & S
\end{tikzcd}$$

The result then follows by noting that the left-hand square is a pushout. □

Let S be a category. We define a functor

$$\partial : \mathbf{Cat}_{/S} \to S\text{–}\mathbf{Set}$$

as follows. For any $F : X \to S$, we write

$$\underline{1}^X : X \to \mathbf{Set}$$

to denote the terminal object of X–**Set** (see Notation 1.5), and note that

$$\int \left(\underline{1}^X\right) \cong X$$

in $\mathbf{Cat}_{/X}$. We define $\partial(F) : S \to \mathbf{Set}$ by

$$\partial(F) := \Sigma_F\left(\underline{1}^X\right).$$

We have the following proposition, which is well known.

Proposition 3.9.

(i) The functor ∂ is left adjoint to $\int$:

$$\begin{tikzcd}\mathbf{Cat}_{/S} \arrow[r, shift left, "\partial"] & S\text{–}\mathbf{Set}. \arrow[l, shift left, "\int"]\end{tikzcd}$$

(ii) For any $\gamma : S \to \mathbf{Set}$, the counit map is an isomorphism:

$$\partial \circ \int(\gamma) \xrightarrow{\cong} \gamma.$$

(iii) An object $X \xrightarrow{F} S$ in $\mathbf{Cat}_{/S}$ is a discrete opfibration if and only if

$$F \cong \int \partial(F)$$

in $\mathbf{Cat}_{/S}$.

Proof. Let $F : X \to S$ be an object of $\mathbf{Cat}_{/S}$ and let $\gamma : S \to \mathbf{Set}$ be an object of S–**Set**. By Proposition 2.7, we have the pullback diagram

$$\begin{tikzcd}\int(\Delta_F\gamma) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \int(\gamma) \arrow[d] \\ X \arrow[r, "F"'] & S\end{tikzcd}$$

which implies the first isomorphism in the chain

$$\begin{aligned}\mathrm{Hom}_{\mathbf{Cat}_{/S}}\left(F, \int(\gamma)\right) &\cong \mathrm{Hom}_{\mathbf{Cat}_{/X}}\left(\mathrm{id}_X, \int(\Delta_F\gamma)\right)\\ &\cong \mathrm{Hom}_{X\text{–}\mathbf{Set}}\left(\underline{1}^X, \Delta_F\gamma\right)\\ &\cong \mathrm{Hom}_{S\text{–}\mathbf{Set}}\left(\Sigma_F\left(\underline{1}^X\right), \gamma\right) = \mathrm{Hom}_{S\text{–}\mathbf{Set}}(\partial F, \gamma)\end{aligned}$$

where the second isomorphism follows from Lemma 2.6 and the third is adjointness. This completes the proof of Statement (i).

Statement (ii) follows from the same lemma.

By construction, $\pi : \int(\delta) \to S$ is a discrete opfibration for any $\delta : S \to \mathbf{Set}$, so if $X \xrightarrow{F} S$ is not a discrete opfibration, then $X \not\cong \int \partial(F)$. Thus, it remains to show that if F is a discrete opfibration, then $X \cong \int \partial(F)$. To see this, note that for each $s \in \mathrm{Ob}(S)$, the set $F^{-1}(s)$ is final in $(F \downarrow s)$, so

$$\partial(F)(s) = \Sigma_F\left(\underline{1}^X\right)(s) = \underset{(F\downarrow s)}{\mathrm{colim}}\, \underline{1}^X \cong F^{-1}(s).$$

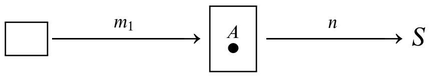
This shows that the object structure in F is the same as that in $\int \partial(F)$. Similar analyses can be carried out for arrows and path equivalences. $\square$

3.3. *Examples*

In this section we will show how to use lifting constraints (see Definition 3.1) to declare a number of different properties for tables and foreign keys in a database. Our examples will describe each of the following in turn:

— declaring a table to be non-empty;
— declaring a table to have exactly one row;
— declaring a foreign key to be injective;
— declaring a foreign key to be surjective;
— declaring a binary relation to be reflexive, symmetric and/or transitive;
— declaring a table to be a product (or more generally a limit) of other tables; and
— declaring that there are no non-trivial cycles in the data on a self-referencing table.

Example 3.10 (non-empty). Let S be a schema and $T \in \mathrm{Ob}(S)$ be a table, which we want to declare to be non-empty. We use the constraint drawn as follows



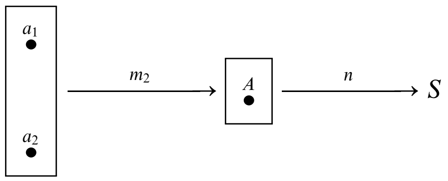
where $n(A) = T$. In other words, we set $W_1 = \emptyset$ to be the empty category, and we set $R = \{A\}$ to be the discrete category with one object A. To say that the lifting problem

$$\begin{tikzcd}
W_1 \arrow[r] \arrow[d, "m_1"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] \arrow[ur, dashed] & S
\end{tikzcd}$$

has a solution is to say that there exists an object in the instance category I whose image under π is T. In other words, there exists a row in table T. Here, the commutativity of the upper-left triangle does nothing, and the commutativity of the lower-right triangle does all the work.

Example 3.11 (cardinality=1). Let S be a schema and $T \in \mathrm{Ob}(S)$ be a table, which we want to declare to have exactly one row. We know a constraint guaranteeing the existence of a row in T from Example 3.10. In Section 3.4, we will give a general method for transforming existence constraints into uniqueness constraints, but here we will just give the result of that method.

To declare T to have at most one row, we use the constraint drawn as follows:



where

$$m_2(a_1) = m_2(a_2) = A$$

and

$$n(A) = T.$$

In other words, we set $W_2 = \{a_1, a_2\}$ to be a discrete category with two objects, and we set $R = \{A\}$ to be a discrete category with one object. The lifting problem

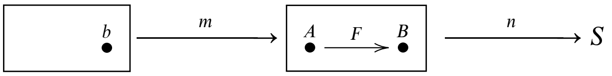
$$\begin{tikzcd}
W_2 \arrow[r] \arrow[d, "m_2"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] \arrow[ur, dashed] & S
\end{tikzcd}$$

has a solution if and only if both triangles commute. We already know that the image of a and b in I consists of two rows in table T because the square commutes. The commutativity of the upper-left triangle implies that a and b are the same, as desired. The commutativity of the lower-right triangle is implied by the surjectivity of m_2 and the commutativity of the square.

The set $\{(m_1, n), (m_2, n)\}$ is a constraint set on S that is satisfied by a discrete opfibration π if and only if the set $I(T)$ of rows in T has exactly one element.

We will treat the remaining examples more briefly. The following constraint was used in Example 3.3.

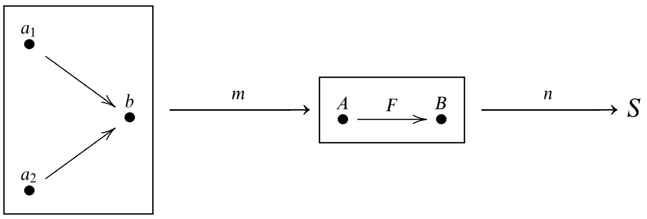
Example 3.12 (surjective foreign key). The declaration stating that a foreign key $f : T \to T'$ is surjective is achieved by the constraint



where

$$\begin{aligned}
m(b) &= B \\
n(A) &= T \\
n(B) &= T' \\
n(F) &= f.
\end{aligned}$$

Example 3.13 (injective foreign key). The declaration stating that a foreign key $f : T \to T'$ is injective is achieved by the constraint



where

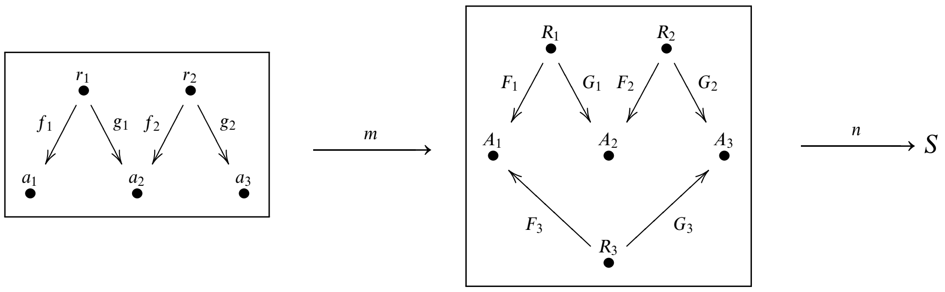
$$\begin{aligned} m(a_1) = m(a_2) &= A \\ m(b) &= B \\ n(F) &= f. \end{aligned}$$

Example 3.14 (reflexive, symmetric and/or transitive binary relation). In this example, we will only describe the constraints that ensure a binary relation $R \subseteq A \times A$ is transitive and leave the symmetric and reflexive cases as exercises.

The declaration that a relation

$$\boxed{\begin{tikzcd} R \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & A \end{tikzcd}} \subseteq S$$

is transitive is achieved by the constraint



where the functors m and n should be clear by our labelling: for example,

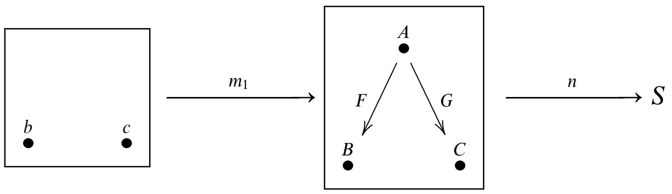
$$n(R_1) = n(R_2) = n(R_3) = R.$$

The next example describes lifting constraints for products. However, this is part of a much larger story, and in Section 3.5 we will show that any limit constraint can be modelled by lifting constraints.

Example 3.15 (product). Suppose we have a table T and two of its columns are

$$\begin{aligned} f &: T \to U \\ g &: T \to V. \end{aligned}$$

The declaration that (the set of rows in) table T is the product of (the sets of rows in) tables U and V is achieved by two constraints: an existence constraint and a uniqueness constraint. The existence constraint is



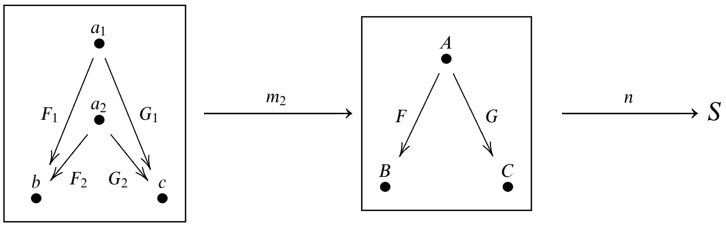
where

$$m_1(b) = B$$
$$m_1(c) = C,$$

and

$$n(F) = f$$
$$n(G) = g.$$

The uniqueness constraint is



where

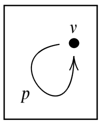
$$m_2(F_1) = m_2(F_2) = F$$
$$m_2(G_1) = m_2(G_2) = G,$$

and

$$n(F) = f$$
$$n(G) = g.$$

Thus, the constraint set for (T, f, g) to be a product is $\{(m_1, n), (m_2, n)\}$.

Example 3.16 (forests). Let S be the free category generated by the graph with one object and one arrow:

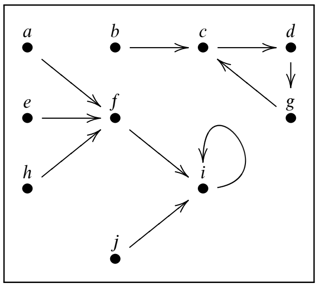
$S :=$  (12)

This is just a self-referencing table. In mathematics, an instance $\delta : S \to \mathbf{Set}$ of such a self-referencing table is called a *discrete dynamical system* or *DDS*. The set $\delta(v)$ will be called the *set of nodes* of δ, and given a node $x \in \delta(v)$, the node $\delta(p)(x)$ is called the *parent of x*. The following picture shows such an instance δ and its Grothendieck construction $I = \int(\delta)$:

$\delta :=$

v	
ID	**p**
a	f
b	c
c	d
d	g
e	f
f	i
g	c
h	f
i	i
j	i

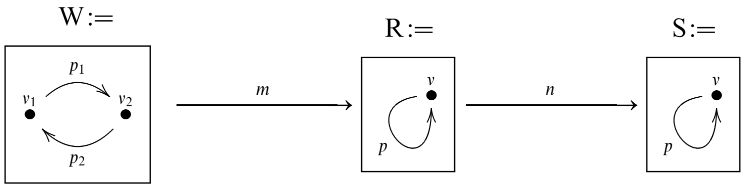
$I :=$



(13)

Note that a DDS looks like a forest (collection of trees) except that it may have cycles. These cycles can only occur at the root of a tree, and, indeed, each tree in the forest has a root cycle. In (13), the tree containing a has a root cycle of length 1, and the tree containing b has a root cycle of length 3. Forests are a useful notion in computer science: we will consider a DDS to be a forest if and only if each root cycle has length 1. This can be achieved by the following lifting constraint.

Let $R = S$ be the schema in (13), and let $n = \mathrm{id}\colon R \to S$. Let W be the free category on the graph below, and let $m\colon W \to R$ denote the functor sending p_1 and p_2 to p:



3.4. *Encoding uniqueness constraints*

Suppose we are given a constraint $W \xrightarrow{m} R \xrightarrow{n} S$. According to Definition 3.1, a functor $\pi\colon I \to S$ satisfies (m, n) if for every solid arrow diagram

$$\begin{tikzcd}
W \arrow[r] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] \arrow[ur, dashed] & S
\end{tikzcd} \tag{14}$$

there exists a dotted arrow lift making it commute. Thus, it may appear that all lifting constraints are existence declarations. However, by employing a technique found in Makkai (1997), we can always turn such an existence declaration into a uniqueness declaration using a related lifting diagram. In fact, we have already done this a couple times – see Examples 3.11 and 3.15. The uniqueness constraint corresponding to (m, n) is

$$R \amalg_W R \xrightarrow{(\mathrm{id}_R \amalg \mathrm{id}_R)} R \xrightarrow{n} S. \tag{15}$$

In other words, π satisfies constraint (15) if and only if there exists *at most one* dotted arrow lift making diagram (14) commute.

3.5. *Lifting constraints are more expressive than limit sketches*

In this section we show that lifting constraints are more expressive than limit sketches when it comes to set models. We define limit sketches in Definition 3.17, prove that lifting constraints are at least as expressive as limit sketches in Proposition 3.18, and then prove that lifting constraints are strictly more expressive than limit sketches in Proposition 3.19.

For any category $\mathcal{C}$, we let $\mathcal{C}^{\triangleleft}$ denote the category obtained by adjoining an initial object to $\mathcal{C}$.

Definition 3.17. A *limit sketch* consists of a category S and a set D of commutative diagrams in **Cat** of the form

$$\begin{tikzcd} J_d \arrow[r, "X_d"] \arrow[d, "i_d"'] & S \\ (J_d)^{\triangleleft} \arrow[ur, "L_d"'] & \end{tikzcd}$$

one for each $d \in D$. We call each X_d a *specified limit pre-cone* in S and each L_d a *specified limit cone* in S. We call S the *underlying category* of the sketch (S, D).

If (S, D) is a limit sketch, then an (S, D)-*model* is a functor $\delta : S \to \mathbf{Set}$ such that for each $d \in D$, the map i_d induces an isomorphism

$$\lim_{(J_d)^{\triangleleft}}(\delta \circ L_d) \cong \lim_{J_d}(\delta \circ X_d). \tag{16}$$

Proposition 3.18. If (S, D) is a limit sketch, we can construct a set of lifting constraints ξ such that the functor $\int : S\text{–}\mathbf{Set} \to \mathbf{Cat}_{/S}$ induces a bijection between the set of functors $\delta : S \to \mathbf{Set}$ modelling (S, D) and the set of instances $\pi : I \to S$ satisfying ξ.

Proof. It suffices to show that for each diagram $d = (J, X, L)$ as shown below

$$\begin{tikzcd} J_d \arrow[r, "X_d"] \arrow[d, "i_d"'] & S \\ (J_d)^{\triangleleft} \arrow[ur, "L_d"'] & \end{tikzcd} \tag{17}$$

there exists a set K_d of lifting constraints $\{(m_k, n_k)\}_{k \in K_d}$ with the property that $\delta : S \to \mathbf{Set}$ satisfies (16) if and only if $\int(\delta) \to S$ satisfies the constraints in K_d.

The limit $\lim_J \delta \circ X_d$ is in bijection with the set of dotted lifts

$$s_d : J_d \to \int(\delta)$$

(such that $\pi \circ s_d = X_d$) in

$$\begin{tikzcd}
& \int(\delta) \arrow[d, "\pi"] & \\
J_d \arrow[ur, dashed] \arrow[r, "X_d"] \arrow[d, "i_d"'] & S \arrow[r, "\delta"] & \mathbf{Set} \\
J_d^{\triangleleft} \arrow[ur, "L_d"'] & &
\end{tikzcd}$$

and, similarly, the limit $\lim_{(J_d)^{\triangleleft}} \delta \circ L_d$ is in bijection with the set of lifts

$$(J_d)^{\triangleleft} \to \int(\delta).$$

Thus, to say that δ models d is to say that for every commutative diagram of the form

$$\begin{tikzcd}
J_d \arrow[r, "s_d"] \arrow[d, "i_d"'] & \int(\delta) \arrow[d, "\pi"] \\
(J_d)^{\triangleleft} \arrow[r, "L_d"'] \arrow[ur, dashed] & S
\end{tikzcd}$$

there exists a unique dotted lift. We thus take K_d to be the set

$$\{(i_d, L_d), (i'_d, L_d)\},$$

where

$$i'_d : (J_d)^{\triangleleft} \amalg_{J_d} (J_d)^{\triangleleft} \to (J_d)^{\triangleleft}.$$

In other words (i_d, L_d) encodes the existence of the dotted lift and (i'_d, L_d) encodes its uniqueness, as in Section 3.4. □

Proposition 3.19. There exists a schema S and a set of lifting constraints ξ on it whose satisfaction is not modelled by any limit sketch with underlying category S.

Proof. Let $S = \underline{1}$ be the terminal category, so a functor $\delta : S \to \mathbf{Set}$ can be considered as just a set $\delta \in \mathrm{Ob}(\mathbf{Set})$ and we have $I := \int(\delta) \cong \delta$. Consider the unique lifting constraint of the form

$$\underline{2} \xrightarrow{m} \underline{1} \xrightarrow{n} S.$$

Up to isomorphism, there exist precisely two instances δ satisfying $\xi = \{(m, n)\}$, namely, $\delta \cong \underline{0}$ and $\delta \cong \underline{1}$. We will show that there is no limit sketch with underlying category S having only two models up to isomorphism.

For a limit sketch on S, each (J, X, L) either $J = \varnothing$ or $J \xrightarrow{X} S$ is an epimorphism. In the first case,

$$\lim_{(J_d)^{\triangleleft}}(\delta \circ L_d) \cong \underline{1}$$
$$\lim_{J_d}(\delta \circ X_d) \cong \delta,$$

so a model of (J, X, L) must have $\delta \cong \underline{1}$. In the second case

$$\lim_{(J_d)^{\triangleleft}}(\delta \circ L_d) \cong \delta \cong \lim_{J_d}(\delta \circ X_d),$$

so every set δ models this constraint. Thus, there is no set D such that the set of sketch models of (S, D) has precisely two elements up to isomorphism. $\square$

3.6. *Constraint implications*

Propositions 3.21 and 3.22 in this section are constraint implication results. That is, they show that instances satisfying one lifting constraint automatically satisfy another. These two constraint implications are not meant to be exhaustive, but just to give the idea.

Definition 3.20. Suppose we have a diagram of the form

$$\begin{tikzcd}
W \arrow[r, "s_1"] \arrow[d, "m"] & W' \arrow[r, "p_1"] \arrow[d, "m'"] & W \arrow[d, "m"] \\
R \arrow[r, "s_2"'] & R' \arrow[r, "p_2"'] & R
\end{tikzcd}$$

such that the top and bottom compositions are identity:

$$p_1 \circ s_1 = \mathrm{id}_W$$
$$p_2 \circ s_2 = \mathrm{id}_R.$$

In this case we say that m is a *retract* of m'.

Proposition 3.21. Suppose (m, n) is a constraint for a schema S and that m is a retract of some m', part of which is shown to the left in the diagram

$$\begin{tikzcd}
W' \arrow[r, "p_1"] \arrow[d, "m'"'] & W \arrow[d, "m"] & \\
R' \arrow[r, "p_2"'] & R \arrow[r, "n"'] & S
\end{tikzcd}$$

Then any discrete opfibration $\pi : I \to S$ satisfying $(m', n \circ p_2)$ also satisfies (m, n).

Proof. The proof is straightforward, but we include it for pedagogical reasons. Suppose we are given a lifting problem

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd} \tag{18}$$

We assume by hypothesis that the dotted arrow lift f exists making the solid arrow diagram

$$\begin{tikzcd} W \arrow[r, "s_1"] \arrow[d, "m"] & W' \arrow[r, "p_1"] \arrow[d, "m'"] & W \arrow[r, "p"] \arrow[d, "m"] & I \arrow[d, "\pi"] \\ R \arrow[r, "s_2"'] & R' \arrow[r, "p_2"'] \arrow[urr, dashed, "f"] & R \arrow[r, "n"'] & S \end{tikzcd}$$

commute. We then just need to check that

$$\ell = f \circ s_2 : R \to I$$

is a lift as in (18), which it is. □

Proposition 3.22. Suppose the square to the left in the diagram

$$\begin{tikzcd} W' \arrow[r] \arrow[d, "m'"'] & W \arrow[d, "m"] \\ R' \arrow[r, "q"'] \arrow[ur, phantom, "\ulcorner"] & R \arrow[r, "n"'] & S \end{tikzcd}$$

is a pushout (as indicated by the corner symbol $\ulcorner$). If $\pi : I \to S$ satisfies the constraint $(m', n \circ q)$, then it satisfies (m, n).

Proof. The statement is obviously true. □

4. Queries as lifting problems

In this section, we will show a correspondence between queries and lifting problems, under which the set of results for a query corresponds to the set of solutions (that is, lifts) for the associated lifting problem. The main example we will use was discussed in Example 1.1. There we were interested in learning more about a married couple, given certain information about them. After building up the necessary theory in Sections 4.1 and 4.3, we will apply it to the case of the married couple in Example 4.10.

In the Introduction, more specifically in (1), we alluded to a dictionary between certain SQL statements and lifting problems. In this section we will extend this a bit to include more specificity in the SELECT clause. Specifically, we have the correspondence

$$\begin{tikzcd} & W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ X \arrow[r, "q"'] & R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd}$$

$$\begin{array}{ll} \text{SELECT} & X \xrightarrow{q} R \\ \text{FROM} & R \xrightarrow{n} S \\ \text{WHERE} & R \xleftarrow{m} W \xrightarrow{p} I \end{array} \qquad (19)$$

The map q can be composed with any lift $\ell\colon R \to I$ to restrict our attention (that is, project) to a certain segment of the result. We will explain these ideas in Example 4.8. However, before getting to this general kind of query, we will discuss queries that do not include the WHERE-clause, that is, the collection $W \to I$ of knowns.

4.1. *WHERE-less queries*

In this section we study queries as in Diagram (19) in which the where-clause W is empty, $W = \emptyset$. Such queries are often called *views*. In this case the two maps

$$R \xleftarrow{m} W \xrightarrow{p} I$$

contain no information, so Diagram (19) reduces to

$$\begin{tikzcd} & & I \arrow[d, "\pi"] \\ X \arrow[r, "q"'] & R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd}$$

$$\begin{array}{ll} \text{SELECT} & X \xrightarrow{q} R \\ \text{FROM} & R \xrightarrow{n} S \end{array}$$

We call these *WHERE-less queries.*

Definition 4.1. Let S be a schema. A *probe on* S is a functor $n\colon R \to S$, where the category R is called *the result schema for the probe*. Given a discrete opfibration $\pi\colon I \to S$, the probe n is said to *set up the lifting problem*

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] \arrow[ur, dashed] & S \end{tikzcd}$$

In the presence of a discrete opfibration π, we may refer to the probe n as *a where-less query*. We define the *set of solutions* to the query, denoted $\Gamma(n, \pi)$ as

$$\Gamma(n, \pi) := \{\ell : R \to I \mid \pi \circ \ell = n\}.$$

Example 4.2. Consider the discrete opfibration $\pi : I \to S$ given by

$$I = \boxed{\begin{tikzcd}[row sep=small] & \texttt{Ann} \\ & \texttt{Bob} \\ x137 \arrow[ruu] \arrow[rddd] & \texttt{Deb} \\ x139 \arrow[ruu] \arrow[rdd] & \\ x144 \arrow[ruu] \arrow[rdd] & \\ & \texttt{Smith} \\ & \texttt{Jones} \end{tikzcd}}$$

$$\Big\downarrow \pi$$

$$S = \boxed{\begin{tikzcd} & \texttt{FNames} \\ \texttt{Person} \arrow[ru, "\text{First}"] \arrow[rd, "\text{Last}"'] & \\ & \texttt{LNames} \end{tikzcd}}$$

To find two people with the same last name, we find lifts of the where-less query

$$R := \boxed{\begin{tikzcd} P_1 \arrow[rd, "L_1"] & \\ & LN \\ P_2 \arrow[ru, "L_2"'] & \end{tikzcd}} \xrightarrow{\;n\;} \boxed{\begin{tikzcd} & \texttt{FNames} \\ \texttt{Person} \arrow[ru, "\text{First}"] \arrow[rd, "\text{Last}"'] & \\ & \texttt{LNames} \end{tikzcd}} = S$$

where

$$n(L_1) = n(L_2) = \left(\begin{tikzcd} \texttt{Person} \arrow[r, "\text{Last}"] & \texttt{LNames} \end{tikzcd} \right).$$

There are two people (Ann Smith, Bob Smith) with the same last name, so we may hope to get as our result set

$$\{(x137, \text{Smith}, x139)\}.$$

We compute the result set for our query as follows. We are looking for functors $\ell : R \to I$ that make the diagram

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ R \arrow[ur, "\ell"] \arrow[r, "n"'] & S \end{tikzcd} \tag{20}$$

commute. Since L_1 and L_2 in R are sent to Last in S, we need to choose two 'downward sloping' arrows in I with the same target. Doing so, we indeed find all pairs of persons in I that have the same last name. Unfortunately, this query would return five results, which we can abbreviate as

$$\begin{array}{lll} (x137, \text{Smith}, x139), & (x139, \text{Smith}, x137), & \\ (x137, \text{Smith}, x137), & (x139, \text{Smith}, x139), & (x144, \text{Jones}, x144). \end{array} \tag{21}$$

The first two are what we are looking for, but they are redundant; the last three are degenerate (for example, Deb Jones has the same last name as Deb Jones). We will deal with these issues in Example 4.5, after we have discussed morphisms of queries.

Definition 4.3. Let S be a schema. Given two probes

$$n_1 : R_1 \to S$$
$$n_2 : R_2 \to S,$$

we define a *strict morphism* from n_1 to n_2, denoted $f : n_1 \to n_2$, to be a functor $f : R_1 \to R_2$ such that

$$n_2 \circ f = n_1.$$

Let $\widetilde{\mathbf{Prb}}(S) = \mathbf{Cat}_{/S}$ denote the category whose objects are probes and whose morphisms are strict morphisms. In the presence of a discrete opfibration $\pi : I \to S$, we may refer to f as a *strict morphism of where-less queries* (as in Definition 4.1).

Given a strict morphism $f : n_1 \to n_2$, we obtain a function

$$\Gamma(f, \pi) : \Gamma(n_2, \pi) \to \Gamma(n_1, \pi),$$

because any lift ℓ_2 in the diagram

$$\begin{tikzcd} & & I \arrow[d, "\pi"] \\ R_1 \arrow[r, "f"] \arrow[rr, bend right, "n_1"'] & R_2 \arrow[ur, "\ell_2"] \arrow[r, "n_2"] & S, \end{tikzcd} \tag{22}$$

that is, with $n_2 = \pi \circ \ell_2$, induces a lift

$$\ell_1 := \ell_2 \circ f : R_1 \to I$$

with $n_1 = \pi \circ \ell_1$. We have thus produced a functor

$$\Gamma(-, \pi) : \widetilde{\mathbf{Prb}}(S)^{\mathrm{op}} \to \mathbf{Set},$$

which is just the representable functor at π,

$$\Gamma(-, \pi) = \mathrm{Hom}_{\widetilde{\mathbf{Prb}}(S)}(-, \pi).$$

Remark 4.4. We use the term *strict* morphism for the probes in Definition 4.3 because we will define a more lax version of morphism in Definition 5.2. While in the above we consider commutative triangles of categories (for example, $n_2 \circ f = n_1$ in (22)) and call the resulting category $\widetilde{\mathbf{Prb}}(S)$, the lax version will allow for natural transformations (for example, $n_2 \circ f \Rightarrow n_1$) and will be denoted by $\mathbf{Prb}(S)$. The functor

$$\Gamma(-, \pi) : \widetilde{\mathbf{Prb}}(S) \to \mathbf{Set}$$

defined in Definition 4.3 can be extended to a functor (which we give the same name),

$$\Gamma(-, \pi) : \mathbf{Prb}(S) \to \mathbf{Set}.$$

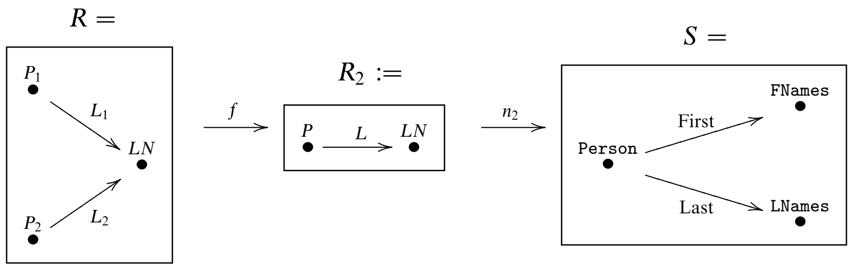
This will all be discussed fully in Section 5.1.

Example 4.5. We again consider the situation from Example 4.2, where we were using the query $n : R \to S$ to look for pairs of people who had the same last name. The solution set in (21) had two problems:

— we got degenerate answers because each person has the same last name as him/herself; and

— we got order redundancy because, given two people with the same last name, we can reverse the order and get another such pair.

In order to deal with the first issue, consider the strict morphism f of queries



where

$$f(L_1) = f(L_2) = L,$$

and note that we do indeed have

$$n = n_2 \circ f.$$

By Definition 4.3, this induces a function between the solution sets: that is, we get a function

$$\Gamma(f, \pi) : \Gamma(n_2, \pi) \to \Gamma(n, \pi).$$

In our example (21), the image of this function is precisely the set of duplicates. In other words, if we delete the elements in the image of $\Gamma(f,\pi)$, we get

$$\Gamma(n,\pi) - \Gamma(n_2,\pi) = \{(x137, \text{Smith}, x139), (x139, \text{Smith}, x137)\}.$$

In order to deal with the remaining order-redundancy issue, consider the swap map $s\colon R \to R$ given by

$$s(L_1) = L_2$$
$$s(L_2) = L_1.$$

Note that $n \circ s = n$. Thus we have a strict morphism of probes $s\colon n \to n$, which induces a function

$$\Gamma(s,\pi)\colon \Gamma(n,\pi) \to \Gamma(n,\pi).$$

By taking the orbits of this function, we effectively quotient out by order-swapping. In fact, our swap map acts not just on (R,n) but on (R_2,n_2) as well, so we can combine this method with the one above to obtain the desired answer, the one element set consisting of $(x137, \text{Smith}, x139)$, in unspecified order.

Proposition 4.6. Let $\delta : S \to \mathbf{Set}$ be an instance and $\pi_\delta : I \to S$ be the induced discrete opfibration. Given any probe $n\colon R \to S$, there is an isomorphism

$$\Gamma(n,\pi_\delta) \xrightarrow{\cong} \lim_R(\delta \circ n).$$

Proof. Consider the diagram

$$\begin{tikzcd}
 & I \arrow[r] \arrow[d, "\pi_\delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{Set}_* \arrow[d, "\pi"] \\
R \arrow[r, "n"'] & S \arrow[r, "\delta"'] & \mathbf{Set}
\end{tikzcd}$$

where the right-hand square is a pullback, as shown in Proposition 2.5. We have a bijection

$$\text{Hom}_{\mathbf{Cat}_{/S}}(n,\pi_\delta) \cong \text{Hom}_{\mathbf{Cat}_{/\mathbf{Set}}}(\delta \circ n, \pi).$$

The left-hand side is $\Gamma(n,\pi_\delta)$ and the right-hand side is a standard formula for the limit of a set-valued functor, in this case for $\lim_R(\delta \circ n)$. □

4.2. *Binding variables*

In Section 3.1 we defined lifting constraints on a schema S to be a pair of composable functors $W \xrightarrow{m} R \xrightarrow{n} S$. The idea is to think of R as a set of equations (or a *join graph*) and of W as a set of variables to be bound at run-time. Our lifting approach for queries will assume that the variables (in W) have already been bound to something in the active domain of π. As mentioned in the introduction, it is not standard to allow queries to depend on instances. In this short section we will explain how to use where-less queries to determine the active domains. In this way, we will explain how instance-independent queries can be posed using the same lifting-problems approach.

The idea is reminiscent of what is known in modern database practice as a *cursor*. Once the active domains for the variables in W are found, one can either run the cursor (that is, the query) parameterised over all values in these active domains, or prompt the user to choose bindings for these variables.

We will assume for this section that W is a discrete category, which is the most common case in practice, but all the ideas we will discuss generalise to the non-discrete case.

Suppose we are given a cursor $W \xrightarrow{m} R \xrightarrow{n} S$. To determine the active domains of each variable in W, we simply apply the where-less query given by the diagram

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ W \arrow[ru, dashed] \arrow[r, "n \circ m"'] & S \end{tikzcd}$$

The set of lifts $\Gamma(n \circ m, \pi)$ is the set of possible variable bindings. Once a lift $p \colon W \to I$ has been chosen, we have a commutative square

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[ru, dashed, "\ell"] \arrow[r, "n"'] & S \end{tikzcd}$$

and, as we will see in Section 4.3, the dotted arrow lifts ℓ will correspond to the results of the now fully defined query.

There is one more case we should discuss. Suppose we want to pose a query where it is not known in advance whether the chosen constants will or will not be available in the active domain – if they are not, the query must certainly return an empty set of results, and this is the intended behaviour. In fact, this is the type of situation that is most often called a query in the database literature. In the remainder of this section, we will explain how this is handled by lifting queries.

Let Dom denote the set of all possible domain values, $\overline{Dom}$ denote the indiscrete category on Dom and $d \colon \overline{Dom} \times S \to S$ denote the projection[†]. Recall that for any category $\mathcal{C}$, the set of functions $\mathrm{Ob}(\mathcal{C}) \to Dom$ is in natural bijection with the set of functors $I \to \overline{Dom}$.

We are given the shape of the query $W \xrightarrow{m} R \xrightarrow{n} S$, and we are also given, for each $w \in W$, a value $t(w) \in Dom$. In other words, our query is represented by the commutative square

$$\begin{tikzcd} W \arrow[r, "(t, n \circ m)"] \arrow[d, "m"'] & \overline{Dom} \times S \arrow[d, "d"] \\ R \arrow[r, "n"'] & S \end{tikzcd}$$

[†] If we want each table $s \in \mathrm{Ob}(S)$ to have its own data type, we just replace d with the appropriate category over S.

We are also given a map $v : I \to \overline{Dom}$ that sends each datum $i \in \text{Ob}(I)$ to its value in Dom:

$$\begin{tikzcd}
 & I \arrow[d, "{(v,\pi)}"] \\
W \arrow[r] \arrow[d, "m"'] & \overline{Dom} \times S \arrow[d, "d"] \\
{} \arrow[r, "n"'] & S
\end{tikzcd}$$

As above, we perform the query in two steps. First we find all lifts $p \colon W \to I$ such that

$$(v, \pi) \circ p = (t, n \circ m).$$

If this set is empty, the query will return an empty result set. However, if there do exist lifts p, then, by choosing one, we bind our W-variables to their values found in the active domain. Finally, for each of them, we find all lifts $R \to I$ making the diagram

$$\begin{tikzcd}
 & I \arrow[d, "{(v,\pi)}"] \\
W \arrow[ur, dashed, "p"] \arrow[r] \arrow[d, "m"'] & \overline{Dom} \times S \arrow[d, "d"] \\
R \arrow[uur, dashed] \arrow[r, "n"'] & S
\end{tikzcd}$$

commute. The set of all ways of doing this is the set of results for our query.

4.3. *General lifting queries*

In this section we tackle the more general lifting query. These closely resemble graph pattern queries, as used in SPARQL (Prud'hommeaux and Seaborne 2008). We will show how to perform queries like (and including) the one suggested in Example 1.1, where we hoped to find the last names of our new acquaintances Bob and Sue. We begin with the definition.

Definition 4.7. Let S be a schema and $\pi : I \to S$ be a discrete opfibration. A *query on* π is a commutative diagram of the form made up of the solid arrows in the diagram

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R \arrow[ur, dashed, "\ell"] \arrow[r, "n"'] & S
\end{tikzcd} \qquad (23)$$

The categories W and R are called the *where-category* and the *result schema*, respectively. We define the *set of solutions* to the query, denoted $\Gamma^{m,p}(n, \pi)$, to be the set of lifts ℓ making the diagram commute. Precisely,

$$\Gamma^{m,p}(n, \pi) := \{\ell : R \to I \mid \pi \circ \ell = n \text{ and } \ell \circ m = p\}.$$

Example 4.8. We have now developed enough of the theory required to make sense of the following dictionary:

$$\begin{tikzcd}
 & W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
X \arrow[r, "q"'] & R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S
\end{tikzcd}$$

SELECT $X \xrightarrow{q} R$
FROM $R \xrightarrow{n} S$
WHERE $R \xleftarrow{m} W \xrightarrow{p} I$

Each lift ℓ in the commutative square is a solution to the SELECT $*$ statement, and composing ℓ with q projects to schema X.

The following proposition says that for any query on a dataset δ, there is a canonical embedding of the query result back into δ.

Proposition 4.9. Let $\delta : S \to \mathbf{Set}$ be an instance on a schema and $\pi : I \to S$ be the associated discrete opfibration. Suppose we are given a query (lifting problem)

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] \arrow[ur, dashed] & S
\end{tikzcd}$$

with solution set $\Gamma^{m,p}(n,\pi) \in \mathbf{Set}$. Considering this set as a constant functor $\Gamma : R \to \mathbf{Set}$ (given by $\Gamma(r) = \Gamma^{m,p}(n,\pi)$ for all $r \in \mathrm{Ob}(R)$), there is an induced map of R-sets,

$$\mathrm{Res} : \Gamma \to \Delta_n \delta.$$

Proof. Let

$$\Gamma(n,\pi) = \{\ell : R \to I \mid \pi \circ \ell = n\}$$

denote the set of solutions to the where-less query $n : R \to S$. Clearly, we have an inclusion

$$\Gamma^{m,p}(n,\pi) \hookrightarrow \Gamma(n,\pi).$$

By Proposition 4.6, there is an isomorphism

$$\Gamma(n,\pi) \cong \lim_R(\delta \circ n).$$

Let $t : R \to \underline{1}$ denote the terminal functor. It follows from the definitions that for any functor $G : R \to \mathbf{Set}$, there is an isomorphism of [0]-Sets,

$$\lim_R(G) \cong \Pi_t(G),$$

so, in particular, we have an inclusion

$$\Gamma^{m,p}(n,\pi) \to \Pi_t(\delta \circ n).$$

By the (Δ_t, Π_t)-adjunction, there is an induced map

$$\Delta_t(\Gamma^{m,p}(n,\pi)) \to (\delta \circ n)$$

of R-sets. But since

$$\begin{aligned}\Delta_t(\Gamma^{m,p}(n,\pi)) &= \Gamma\\ \delta \circ n &= \Delta_n \delta,\end{aligned}$$

the result follows. □

Example 4.10 (Bob and Sue, revisited). The motivating example for this paper was presented in Section 1.1. In particular, we provided a SPARQL query to find all instances of married couples with the requisite characteristics (for example, the husband's and wife's first names being Bob and Sue, respectively). We showed that this SPARQL query could be straightforwardly transformed into a lifting problem of the form

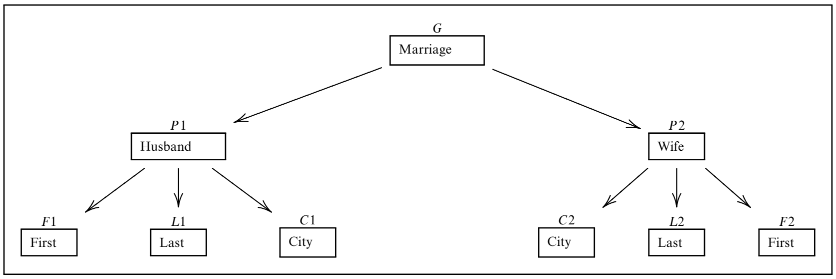
$$\begin{tikzcd} & W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ X \arrow[r, "q"'] & R \arrow[r, "n"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd}$$

as in (5), and we specified the two functors $W \xrightarrow{m} R \xrightarrow{n} S$. We did not specify the discrete opfibration $I \xrightarrow{\pi} S$ or the inclusion of the known data $p\colon W \to I$ because writing out a convincing possibility for I would have used too much space.

The lifting diagram (5) was presumed to have only one solution, because it was presumed that we knew enough about Bob and Sue that no one else would fit the description. In the language of Definition 4.7, the set $\Gamma^{m,p}(n,q)$ has one element. By Proposition 4.9, this element can be written as a database state on R. We output the result as a two-level table with one row in (6), which we repeat here:

Marriage								
ID	**Husband**				**Wife**			
	ID	**First**	**Last**	**City**	**ID**	**First**	**Last**	**City**
G3801	M881-36	Bob	Graf	Cambridge	W913-55	Sue	Graf	Cambridge

In fact, this was a state on a schema $X \xrightarrow{q} R$, where X is the schema

$X :=$  (24)

While we have not discussed two-level tables before, we hope the idea is straightforward enough not to require any further explanation.

4.4. *SPARQL queries involving predicate variables*

In Example 1.1, our SPARQL query (3) only has variables in subject and object positions (the nodes of the schema). It seems that most SPARQL queries used in practice also only have variables in the subject and object positions (see, for example, Deus *et al.* (2010)). However, general SPARQL queries can involve variables in any position, including predicate positions, which correspond to the arrows of the schema. For example, we may use

$$\text{(John ?x Mary)} \tag{25}$$

to find all known relationships between John and Mary. To deal with this type of query, we may proceed as follows.

If $S = (V, E, s, t)$ is a graph (which can be thought of as a schema with trivial path equivalences, which is in keeping with RDF schemas), then S itself can be viewed as a database instance $S : \mathcal{G} \to \mathbf{Set}$ on the schema

$$\mathcal{G} = \begin{tikzcd} \overset{\texttt{Verb}}{\bullet} \arrow[r, bend left, "\text{subj}"] \arrow[r, bend right, "\text{obj}"'] & \overset{\texttt{Noun}}{\bullet} \end{tikzcd}$$

similar to Example 3.3. We will be working with the Grothendieck construction

$$\int(S) \to \mathcal{G}.$$

The category $\int(S)$ will be generated by a bipartite graph. The set of vertices in $\int(S)$ is the union $\texttt{Noun} \amalg \texttt{Verb}$, which we can call noun vertices and verb vertices. There is a unique edge in $\int(S)$ from every verb vertex to its subject noun vertex and another to its object noun vertex.

Example 4.11. Let $X = (V, E, s, t)$ be the graph

$$X := \begin{tikzcd} \overset{\texttt{John}}{\bullet} \arrow[r, "\text{livesIn}"] \arrow[d, "\text{sonOf}"'] & \overset{\texttt{Iowa}}{\bullet} \\ \overset{\texttt{Mary}}{\bullet} \end{tikzcd}$$

If X is conceived as an instance $X : \mathcal{G} \to \mathbf{Set}$, then $\int(X)$ is the category

$$\int(X) := \begin{tikzcd} \overset{\texttt{John}}{\bullet} & \overset{\texttt{livesIn}}{\bullet} \arrow[l, "\text{subj}"'] \arrow[r, "\text{obj}"] & \overset{\texttt{Iowa}}{\bullet} \\ \overset{\texttt{sonOf}}{\bullet} \arrow[u, "\text{subj}"'] \arrow[d, "\text{obj}"] \\ \overset{\texttt{Mary}}{\bullet} \end{tikzcd}$$

An instance $\pi : I \to S$ can be considered simply as a map of graphs, that is, a map of instances on $\mathcal{G}$. Taking its Grothendieck construction yields a functor

$$\int(I) \to \int(S),$$

whereby each arrow from S (representing a foreign key column) and each arrow from I (representing a cell in a foreign key column) have become a vertex in $\int(S)$ and $\int(I)$, respectively, as in Example 4.11. We can perform the original SPARQL query (25) to this derived form of the database because our original predicate can now be accessed as a subject or an object. For example, our statement (John ?x Mary) would become the pair of statements (?x subj John) and (?x obj Mary).

5. The category of queries on a database

In this section we will discuss some formal properties of the machinery developed in earlier sections. For example, we will show that the queries on a given database can be arranged into a database of their own, which can be queried subsequently. This process is commonly known as nesting queries. To this end, we define a category of queries and prove that the process of finding solutions is functorial. We will do this in Sections 5.1 and 5.2. In Section 5.3, we extend some results from Section 3.6, giving more details of the interaction between data migration functors, on the one hand, and query containment and constraint implication on the other.

This section is technical, but it may have fruitful applications. Given any database π, the category $\mathbf{Qry}(\pi)$ organises the queries (or views) on π into a schema of their own. There is a canonical instance on $\mathbf{Qry}(\pi)$ populating each table (corresponding to a query) with its set of results. In typical applications, users of a database π are often better served by interacting with $\mathbf{Qry}(\pi)$ rather than with π. It is important to understand how schema evolution affects different parts of $\mathbf{Qry}(\pi)$, and we will discuss this briefly in Section 5.3.

5.1. *New discrete opfibrations from old*

The following theorem is not new, but its formulation in terms of databases is. Furthermore, the proof may be instructive.

Theorem 5.1. Let $\pi : I \to S$ be a discrete opfibration and B be a category. Then the induced functor $\pi^B : I^B \to S^B$ is a discrete opfibration. If

$$\delta^B = \partial(\pi^B) : S^B \to \mathbf{Set}$$

is the associated instance, then for any $F : B \to S$ in $\mathrm{Ob}(S^B)$, there is a bijection

$$\delta^B(F) \cong \Gamma(F, \pi).$$

Proof. We begin our proof of the first claim by drawing a figure for reference:

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ B \arrow[ur, "\ell_1"] \arrow[r, bend left, "F_1"] \arrow[r, bend right, "F_2"'] \arrow[r, phantom, "\Downarrow\alpha"] & S \end{tikzcd} \tag{26}$$

To see that π^B is a discrete opfibration, suppose $F_1, F_2 : B \to S$ are functors and $\alpha : F_1 \to F_2$ is a natural transformation. Given a functor $\ell_1 : B \to I$ with $\pi \circ \ell_1 = F_1$, we must show that there exists a unique functor $\ell_2 : B \to I$ and natural transformation $\beta : \ell_1 \to \ell_2$ such that

$$\pi \circ \ell_2 = F_2$$
$$\pi \circ \beta = \alpha.$$

For any object $b \in \mathrm{Ob}(B)$, the map

$$\alpha_b : F_1(b) \to F_2(b)$$

in S together with the object $\ell_1(b) \in I$, such that

$$\pi(\ell_1(b)) = F_1(b),$$

induces a unique arrow $\beta_b : \ell_1(b) \to i_b$ in I for some $i_b \in \mathrm{Ob}(I)$, because π is a discrete opfibration. We define $\ell_2(b) = i_b$. This defines $\ell_2 : B \to I$ on objects.

We now suppose that $f : b \to b'$ is any morphism in B. Applying what we have so far, we get a functor $X \to Y$, where X is the solid-arrow portion of the category to the left and Y is the commutative square category to the right,

$$X := \boxed{\begin{tikzcd} \ell_1(b) \arrow[r, "\beta_b"] \arrow[d, "\ell_1(f)"'] & \ell_2(b) \arrow[d, dotted, "?"] \\ \ell_1(b') \arrow[r, "\beta_{b'}"'] & \ell_2(b') \end{tikzcd}} \longrightarrow \boxed{\begin{tikzcd} F_1(b) \arrow[r, "\alpha_b"] \arrow[d, "F_1(f)"'] & F_2(b) \arrow[d, "F_2(f)"] \\ F_1(b') \arrow[r, "\alpha_{b'}"'] & F_2(b') \end{tikzcd}} =: Y \tag{27}$$

and we get a commutative diagram

$$\begin{tikzcd} X \arrow[r] \arrow[d] & I \arrow[d, "\pi"] \\ Y \arrow[r] & S \end{tikzcd}$$

In order to complete our definition of ℓ_2, we need to fill in the missing side (the dotted arrow labelled '?') in square X.

The map

$$F_2(f) : F_2(b) \to F_2(b')$$

in S together with the object $\ell_2(b) \in \mathrm{Ob}(I)$ with

$$\pi(\ell_2(b)) = F_2(b)$$

induces a unique arrow

$$h_{b'} : \ell_2(b) \to j_{b'}$$

for some $j_{b'} \in \mathrm{Ob}(I)$ with $\pi(j_b) = b'$. But now we have two maps in I over the composite $F_1(b) \to F_2(b')$, both with source $\ell_1(b) \in \mathrm{Ob}(I)$: namely

$$\beta_{b'} \circ \ell_1(f) : \ell_1(b) \to \ell_2(b')$$
$$h_{b'} \circ \beta_b : \ell_1(b) \to j_{b'}.$$

Since π is a discrete opfibration, their codomains must be equal, so we have a map

$$\ell_2(f) := h_{b'} : \ell_2(b) \to \ell_2(b') = j_{b'},$$

and we have completed the commutative square X in Diagram (27). We have now defined our functor $\ell_2 : B \to I$ and natural transformation $\beta : \ell_1 \to \ell_2$ over α, and they are unique: we made no choices in their constructions. Hence, we have shown that $\pi^B : I^B \to S^B$ is a discrete opfibration.

Let

$$\delta^B := \partial\,(\pi^B) : S^B \to \mathbf{Set}$$

be the instance associated with π^B and let $F \in \mathrm{Ob}(S^B)$ be an object. We can consider F as a map $\underline{1} \xrightarrow{F} S^B$, and $\delta^B(F)$ is isomorphic to the set of lifts in the diagram

$$\begin{tikzcd} & I^B \arrow[d, "\pi^B"] \\ \underline{1} \arrow[ur, dashed] \arrow[r, "F"'] & S^B \end{tikzcd}$$

which by adjointness is in bijection with the set of lifts $\Gamma(F, \pi)$ in the diagram

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ B \arrow[ur, dashed] \arrow[r, "F"'] & S \end{tikzcd}$$

Therefore, we have $\delta^B(F) \cong \Gamma(F, \pi)$, which completes the proof. □

The following definition of $\mathbf{Prb}(S)$ extends the notion of $\widetilde{\mathbf{Prb}}(S)$ given in Definition 4.3: we have

$$\widetilde{\mathbf{Prb}}(S) \subseteq \mathbf{Prb}(S)$$

is a subcategory with the same set of objects. The category $\mathbf{Prb}(S)$ is a 2-category-theoretic version of slice categories, and is not new – see, for example, Kelly (1974).

Definition 5.2. Let S be a category. We define the *category of probes on* S, denoted $\mathbf{Prb}(S)$, as follows:

$$\mathrm{Ob}(\mathbf{Prb}(S)) = \{(A, F) \mid A \in \mathrm{Ob}(\mathbf{Cat}), F : A \to S \text{ a functor}\}$$
$$\mathrm{Hom}_{\mathbf{Prb}(S)}((A, F), (A', F')) = \{G, \alpha) \mid G : A' \to A,\ \ \alpha : F \circ G \to F'\}$$

$$\begin{tikzcd} A' \arrow[r, "G"] \arrow[rr, bend right, "F'"'] & A \arrow[r, "F"] \arrow[d, phantom, "\Downarrow\alpha"] & S \\ & {} & \end{tikzcd}$$

Remark 5.3. In the presence of a discrete opfibration $\pi : I \to S$, a probe $F : A \to S$ sets up a *where-less query* on π for which the results are the lifts $\ell \in \Gamma(F, \pi)$ for the diagram

$$\begin{tikzcd} \varnothing \arrow[r] \arrow[d] & I \arrow[d, "\pi"] \\ A \arrow[r, "F"'] \arrow[ur, dashed, "\ell"] & S \end{tikzcd}$$

We call these where-less queries to emphasise the fact that the where-category (upper left of the diagram) is empty.

For any category B, there is an obvious functor $S^B \to \mathbf{Prb}(S)$. The following corollary extends Theorem 5.1 in the obvious sense. One way to understand its content is that we can query over where-less queries. In other words, this is a formalisation of nested queries. For example, we can create a join graph of where-less queries and look for a set of coherent results. Corollary 5.4 (which is not new) implies that given a morphism between two where-less queries on S and given a result for the first query, there is an induced result for the second query. We will deal with the general case of nested queries (those having non-trivial where-categories) in Proposition 5.10.

Corollary 5.4. Let $\pi : I \to S$ be a discrete opfibration. Then the induced functor

$$\overline{\pi} = \mathbf{Prb}(\pi)) : \mathbf{Prb}(I) \to \mathbf{Prb}(S)$$

is a discrete opfibration. The instance associated with π is

$$\Gamma(-, \pi) = \partial(\overline{\pi}) : \mathbf{Prb}(S) \to \mathbf{Set}.$$

Proof. Proving this corollary is really just a matter of writing down the appropriate diagram. In order to show that $\overline{\pi}$ is a discrete opfibration, we choose an object $\ell : A \to I$ in $\mathbf{Prb}(I)$ with

$$\overline{\pi}(\ell) = F : A \to S,$$

we choose a morphism

$$(G, \alpha) : (A, F) \to (A', F')$$

in **Prb**(S), and show that there exists a unique morphism

$$(G,\beta):(A,\ell)\to(A',\ell')$$

in **Prb**(I) for some $\ell':A'\to I$ such that $\pi\circ\beta=\alpha$. In diagrams, we begin with the solid-arrow portion of the diagram

$$\begin{tikzcd}
 & & I \arrow[d, "\pi"] \\
A' \arrow[r, "G"] \arrow[urr, dashed, bend left, "\ell'"] \arrow[rr, bend right, "F'"'] & A \arrow[r, "F"] \arrow[ur, "\ell"'] \arrow[u, phantom, "\Uparrow\beta"] \arrow[d, phantom, "\Downarrow\alpha"] & S \\
 & &
\end{tikzcd} \tag{28}$$

and hope to find such an $\ell':A'\to I$ and $\beta:\ell\circ G\to\ell'$.

We have

$$\pi\circ(\ell\circ G)=F\circ G.$$

Applying Theorem 5.1, there is a unique induced functor $\ell':A'\to I$ and natural transformation $\beta:\ell\to\ell'$ such that $\pi\circ\beta=\alpha$, having the required properties. This completes the proof. □

Remark 5.5. There is a way to express the set of solutions to a lifting problem using limits. Let $\pi:I\to S$ be a discrete opfibration, and consider the query

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R \arrow[r, "n"'] & S
\end{tikzcd}$$

We can consider m as a strict morphism of probes on S, so it induces a function

$$\Gamma(m,\pi):\Gamma(n,\pi)\to\Gamma(nm,\pi),$$

and we can consider $p\in\Gamma(nm,\pi)$ as an element in the codomain. There is a bijection

$$\Gamma^{m,p}(n,\pi)\cong\Gamma(n,\pi)\times_{\Gamma(nm,\pi)}\{p\} \tag{29}$$

expressing the set $\Gamma^{m,p}(n,\pi)$ of solutions to the lifting problem as the fibre of $\Gamma(m,\pi)$ over p. This idea may be useful when we have disjunctions in the WHERE-clause of a query, since it means we can replace $\{p\}$ with the set of disjuncts.

We will now present examples of two types of morphisms of where-less queries, namely, projection and indirection. These types generate all morphisms of where-less queries.

Example 5.6 (projection). Let $\delta:S\to\mathbf{Set}$ be an instance and $\pi:I\to S$ be the associated discrete opfibration. Let $n\in\mathbb{N}$ be a natural number. The *n-column table schema*, here denoted by C_n, is the category with an initial object K, precisely n other objects and precisely n non-identity arrows. It follows that C_n looks like an asterisk (or 'star schema'), for example, C_4 is drawn

$$\begin{tikzcd} & c_1 & \\ c_4 & K \arrow[l] \arrow[r] \arrow[u] \arrow[d] & c_2 \\ & c_3 & \end{tikzcd}$$

A functor $p\colon C_n \to S$ is called an *n-column table schema in S*. For each object $x \in C_n$, we call $p(x) \in \mathrm{Ob}(S)$ a *column of p* and call $p(K)$ the *primary key column of p*. In fact, p is a probe or where-less query. The result set $\Gamma(p, \pi)$ can be thought of as the set of records for instance δ in table p: indeed, $\Gamma(p, \pi)$ is isomorphic to $\delta(p)(K)$ as sets.

For any injection

$$h\colon \{1, 2, \ldots, n'\} \hookrightarrow \{1, 2, \ldots, n\},$$

there is an induced functor $C(h)\colon C_{n'} \to C_n$, which we can compose with p to get a new morphism

$$p' := p \circ C(h)\colon C_{n'} \to S$$

and a strict morphism of probes $p \to p'$. A record in table p is given by a lift ℓ as shown in

$$\begin{tikzcd} & & I \arrow[d, "\pi"] \\ C_{n'} \arrow[r, "C(h)"] \arrow[rr, bend right, "p'"'] & C_n \arrow[r, "p"] \arrow[ur, "\ell"] & S, \end{tikzcd} \tag{30}$$

and composing ℓ with $C(h)$ gives its projection as a record in table p'. Thus h induces a function $\Gamma(p, \pi) \to \Gamma(p', \pi)$, and its image is the associated projection. The diagram

$$\begin{tikzcd} \boxed{\overset{\ell}{\bullet}\qquad\qquad} \arrow[r] \arrow[d] & \mathbf{Prb}(I) \arrow[d, "\mathbf{Prb}(\pi)"] \\ \boxed{\overset{p}{\bullet} \xrightarrow{C(h)} \overset{p'}{\bullet}} \arrow[r] & \mathbf{Prb}(S) \end{tikzcd}$$

is another way of viewing diagram (30).

Remark 5.7. In Example 5.6, we did not really need to assume that the function h was injective. If h were not injective, then the morphism of queries $C(h)$ would result in some duplication of columns rather than a pure projection. In other words, the morphism of queries is simply given by substitution along the function $C(h)$.

In Example 5.6, we changed the shape of the result schema and used a strict morphism of probes (the natural transformation $p \circ C(h) \to p'$ was the identity). In the next example, we will keep the result schema fixed but allow a non-strict morphism.

Example 5.8 (indirection). Let

$$R = [1] = \boxed{\bullet^0 \xrightarrow{f} \bullet^1}$$

and let S be the schema

$$S := \boxed{\begin{tikzcd} \overset{A}{\boxed{\text{a person}}} \arrow[r, "\text{lives at}"] & \overset{B}{\boxed{\text{an address}}} \arrow[r, "\text{is in}"] & \overset{C}{\boxed{\text{a city}}} \end{tikzcd}}$$

There are three non-constant functors $R \to S$, which we denote by F_{AB}, F_{AC} and F_{BC}. There is a natural transformation $\alpha\colon F_{AB} \to F_{AC}$ and a natural transformation $\beta : F_{AC} \to F_{BC}$. Thus we get two morphisms in $\mathbf{Prb}(S)$, namely

$$(\mathrm{id}_R, \alpha)\colon (R, F_{AB}) \to (R, F_{AC})$$
$$(\mathrm{id}_R, \beta)\colon (R, F_{AC}) \to (R, F_{BC}).$$

Suppose $\pi : I \to S$ is an instance. We can draw the setup as

$$\begin{tikzcd} & & & I \arrow[d, "\pi"] \\ R \arrow[r, equal] \arrow[rrr, bend right=60, "F_{BC}"'] \arrow[r, phantom, shift right=4, "\Downarrow\beta"] & R \arrow[r, equal] \arrow[rr, bend right=50, "F_{AC}"'] \arrow[r, phantom, shift right=3, "\Downarrow\alpha"] & R \arrow[r, "F_{AB}"'] & S \end{tikzcd}$$

We can take global sections $\Gamma(-, \pi)$ for each of these three probes and obtain maps between the result sets by Theorem 5.1:

$$\Gamma(F_{AB}, \pi) \xrightarrow{\alpha} \Gamma(F_{AC}, \pi) \xrightarrow{\beta} \Gamma(F_{BC}, \pi).$$

In other words, the morphism of queries induces a morphism of result sets. Put simply, given some person and her address, we can return a person and the city she lives in; given some person and his city, we can return an address and the city it is in.

5.2. *The category of queries*

We are now ready to generalise the category $\mathbf{Prb}(S)$ of where-less queries on S to a category of all (lifting) queries on S.

Definition 5.9. Let $\pi : I \to S$ denote a discrete opfibration. We define the *category of (lifting) queries on* π, denoted $\mathbf{Qry}(\pi)$, as follows. The objects of $\mathbf{Qry}(\pi)$ are commutative diagrams of the form

$$\begin{tikzcd} W \arrow[r, "p"] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\ R \arrow[r, "n"'] & S \end{tikzcd}$$

and the morphisms

$$(F, G, \alpha, \gamma) : (R, W, n, p) \to (R', W', n', p')$$

are diagrams of the form

$$\begin{tikzcd}
W' \arrow[r, "G"'] \arrow[d, "m'"'] \arrow[rr, bend left=50, "p'"] \arrow[rr, phantom, bend left=25, "\Uparrow\gamma"] & W \arrow[r, "p"'] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R' \arrow[r, "F"] \arrow[rr, bend right=50, "n'"'] \arrow[rr, phantom, bend right=25, "\Downarrow\alpha"] & R \arrow[r, "n"] & S
\end{tikzcd}$$

where

$$m \circ G = F \circ m'$$
$$\pi \circ \gamma = \alpha \circ m'.$$

Proposition 5.10. Let $\pi : I \to S$ be a discrete opfibration, and suppose we are given the diagram

$$\begin{tikzcd}
W' \arrow[r, "G"'] \arrow[d, "m'"'] & W \arrow[r, "p"'] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R' \arrow[r, "F"] \arrow[rr, bend right=50, "n'"'] \arrow[rr, phantom, bend right=25, "\Downarrow\alpha"] & R \arrow[r, "n"] & S
\end{tikzcd}$$

where the two squares commute. Then there exists a unique morphism of queries

$$(F, G, \alpha, \gamma) : (R, W, n, , p) \to (R', W', n', p')$$

as in

$$\begin{tikzcd}
W' \arrow[r, "G"'] \arrow[d, "m'"'] \arrow[rr, bend left=50, "p'"] \arrow[rr, phantom, bend left=25, "\Uparrow\gamma"] & W \arrow[r, "p"'] \arrow[d, "m"'] & I \arrow[d, "\pi"] \\
R' \arrow[r, "F"] \arrow[rr, bend right=50, "n'"'] \arrow[rr, phantom, bend right=25, "\Downarrow\alpha"] & R \arrow[r, "n"] & S
\end{tikzcd}$$

Proof. The statement follows from a direct application of Theorem 5.1. Indeed, in place of Diagram (26), we draw

$$\begin{tikzcd}
 & I \arrow[d, "\pi"] \\
W' \arrow[ur, "pG"] \arrow[r, bend left, "nFm'"{name=U}] \arrow[r, bend right, "n'm'"'{name=D}] & S
\arrow[from=U, to=D, phantom, "\Downarrow\alpha m'"]
\end{tikzcd}$$

The unique functor and transformation labelled ℓ_2 and β given by the theorem serve as p' and γ here. □

Theorem 5.11. Let $\pi : I \to S$ be a discrete opfibration. Then

$$\Gamma^{-,-}(-, \pi) : \mathbf{Qry}(\pi) \to \mathbf{Set}$$

is functorial. That is, given a morphism of queries

$$\begin{tikzcd}
W' \arrow[r, "G"'] \arrow[d, "m'"'] \arrow[rr, bend left, "p'"{name=P}] & W \arrow[r, "p"'] \arrow[d, "m"] \arrow[to=P, phantom, "\Uparrow\gamma"] & I \arrow[d, "\pi"] \\
R' \arrow[r, "F"] \arrow[rr, bend right, "n'"'{name=N}] & R \arrow[r, "n"] \arrow[to=N, phantom, "\Downarrow\alpha"] & S
\end{tikzcd}$$

there is an induced function

$$\Gamma^{m,p}(n, \pi) \longrightarrow \Gamma^{m',p'}(n', \pi),$$

which is natural in $\mathbf{Qry}(\pi)$.

Proof (sketch). Suppose we are given a lift $\ell : R \to I$ in $\Gamma^{m,p}(n, \pi)$. By Corollary 5.4, we have a map $\ell' : R' \to I$, with $\pi \circ \ell' = n'$, and a natural transformation $\beta : \ell \circ F \to \ell'$, with $\pi \circ \beta = \alpha$. We need to show that $\ell' \circ m' = p'$ and $\beta \circ m' = \gamma$. But, using the proof technique from Proposition 5.10, this follows from Theorem 5.1 and the definition of discrete opfibration. □

Remark 5.12. Given a discrete opfibration $\pi : I \to S$, we will sometimes denote the functor $\Gamma^{-,-}(-, \pi)$ simply by

$$\Gamma(\pi) : \mathbf{Qry}(\pi) \to \mathbf{Set}.$$

5.3. *Data migration functors*

Recall from Definition 2.3 that, given a functor $F : S \to T$, three data migration functors are induced between the categories S–**Set** and T–**Set**. The most straightforward is denoted

$$\Delta_F : T\text{–}\mathbf{Set} \to S\text{–}\mathbf{Set}.$$

It has both a left adjoint, denoted

$$\Sigma_F : S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set},$$

and a right adjoint, denoted

$$\Pi_F : S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set}.$$

In standard database contexts, schemas evolve over time. We model these schema evolutions as zigzags of functors from one schema to another, along which we can migrate data using a data migration functor. It is useful to know how this will affect queries. Typically, users of a database $\pi : I \to S$ are given access to a subset of $\mathbf{Qry}(\pi)$ – they do not see the whole database, but instead some collection of queries. As the schema evolves it is important to understand how $\mathbf{Qry}(\pi)$ evolves. In this section we describe some results: for example, results are unchanged under a pullback query.

We will begin by giving a description of Π_F in terms of where-less queries (see Section 4.1). Recall that for any object $d \in \mathrm{Ob}(T)$, the 'comma' category $(d \downarrow F)$ is defined by

$$\mathrm{Ob}(d \downarrow F) = \{(c, f) \mid c \in \mathrm{Ob}(S), f : d \to F(c)\}$$
$$\mathrm{Hom}_{(d\downarrow F)}((c, f), (c', f')) = \{g : c \to c' \mid f' \circ F(g) = f\}.$$

There is a natural functor $n_d : (d \downarrow F) \to S$, and given a morphism $h : d \to d'$ in T we have a morphism $(d' \downarrow F) \to (d \downarrow F)$, or, more precisely, $n_{d'} \to n_d$, in $\mathbf{Cat}_{/S}$.

Proposition 5.13. Let $F : S \to T$ be a functor and $\gamma : S \to \mathbf{Set}$ be an instance of S with associated discrete opfibration $\pi : I \to S$. Given any object $d \in \mathrm{Ob}(T)$, there is an associated where-less query

$$\begin{tikzcd} & I \arrow[d, "\pi"] \\ (d \downarrow F) \arrow[ur, dashed] \arrow[r, "n_d"'] & S \end{tikzcd}$$

and we have

$$\Pi_F(\gamma)(d) \cong \Gamma(n_d, \pi).$$

Moreover, a morphism $d \to d'$ in T induces a strict morphism of where-less queries $n_{d'} \to n_d$. Thus, we have a functor $T \to \widetilde{\mathbf{Prb}}(\pi)^{\mathrm{op}}$. Then $\Pi_F(\gamma) : T \to \mathbf{Set}$ is the composition

$$T \xrightarrow{d \mapsto n_d} \widetilde{\mathbf{Prb}}(\pi)^{\mathrm{op}} \xrightarrow{\Gamma(-,\pi)} \mathbf{Set}.$$

Proof. Let F, γ, π, d, and $n_d : (d \downarrow F) \to S$ be as in the statement. By Proposition 4.6, we have

$$\Gamma(n_d, \pi) \cong \lim_R(\gamma \circ n_d).$$

This is exactly the formula for $\Pi_F(\gamma)(d)$ by Mac Lane (1988, Theorem X.3.1) since Π_F is a right Kan extension. The statement for morphisms follows similarly. □

While Proposition 5.13 provides an interesting relationship between right pushforwards and queries, it does not allow us to relate queries on a database to queries on its right pushforward. In the following, we will show briefly that graph pattern queries do transform nicely with respect to data migration functors Σ_F and Δ_F.

We begin by discussing the left pushforward functor. Given a functor $F : S \to T$, we have a migration functor

$$\Sigma_F : S\text{–}\mathbf{Set} \to T\text{–}\mathbf{Set}.$$

If $\delta \in S$–**Set** and $\epsilon \in T$–**Set** are instances, there is a bijection between the set of natural transformations $\Sigma_F\delta \to \epsilon$ and the set of commutative diagrams

$$\begin{tikzcd}
\int(\delta) \arrow[r] \arrow[d, "\pi_\delta"'] & \int(\epsilon) \arrow[d, "\pi_\epsilon"] \\
S \arrow[r, "F"'] & T
\end{tikzcd}$$

Given a query on π_δ, we clearly obtain an induced query on π_ϵ, and a solution to the former yields a solution to the latter:

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & \int(\delta) \arrow[r] \arrow[d, "\pi_\delta"] & \int(\epsilon) \arrow[d, "\pi_\epsilon"] \\
R \arrow[r, "n"'] \arrow[ur, dashed] & S \arrow[r, "F"'] & T
\end{tikzcd}$$

We state this formally in the following proposition.

Proposition 5.14. Let $F : S \to T$ be a functor, $\delta \in S$–**Set** and $\epsilon \in T$–**Set** be instances and $\Sigma_F\delta \to \epsilon$ be a map of T-sets. There exists an induced functor of query categories and a natural transformation diagram

$$\begin{tikzcd}
\mathbf{Qry}(\pi_\delta) \arrow[rr] \arrow[dr, "\Gamma(\pi_\delta)"'] & \Longrightarrow & \mathbf{Qry}(\pi_\epsilon) \arrow[dl, "\Gamma(\pi_\epsilon)"] \\
& \mathbf{Set} &
\end{tikzcd}$$

Proof. The statement follows from the discussion above. □

We will now consider the case where $\delta \cong \Delta_F\epsilon$.

Proposition 5.15. Let $F : S \to T$ and $\epsilon : T \to \mathbf{Set}$ be functors, and let $\delta = \Delta_F\epsilon : S \to \mathbf{Set}$ be its pullback. Let π_δ and π_ϵ be as in Diagram (31) below. Then the results of any query on π_δ are the same as the results of the induced query on π_ϵ. That is, we have a natural isomorphism diagram

$$\begin{tikzcd}
\mathbf{Qry}(\pi_\delta) \arrow[rr] \arrow[dr, "\Gamma(\pi_\delta)"'] & \arrow[d, phantom, "\overset{\cong}{\Longrightarrow}"] & \mathbf{Qry}(\pi_\epsilon) \arrow[dl, "\Gamma(\pi_\epsilon)"] \\
& \mathbf{Set} &
\end{tikzcd}$$

Proof. Consider the diagram

$$\begin{tikzcd}
\int(\delta) \arrow[r] \arrow[d, "\pi_\delta"'] \arrow[dr, phantom, "\lrcorner", very near start] & \int(\epsilon) \arrow[d, "\pi_\epsilon"] \\
S \arrow[r, "F"'] & T
\end{tikzcd} \tag{31}$$

which is a pullback by Proposition 2.7. Given a query on π_δ, we obtain a query on π_ϵ as in Proposition 5.14. The function from solutions for π_δ to solutions for π_ϵ is a bijection by the universal property of pullbacks:

$$\begin{tikzcd}
W \arrow[r, "p"] \arrow[d, "m"'] & \int(\delta) \arrow[r] \arrow[d, "\pi_\delta"] \arrow[dr, phantom, "\lrcorner", very near start] & \int(\epsilon) \arrow[d, "\pi_\epsilon"] \\
R \arrow[r, "n"'] \arrow[urr, dashed] & S \arrow[r, "F"'] & T.
\end{tikzcd}$$

Indeed, given a lift $R \to \int(\epsilon)$ of π_ϵ, the fact that (31) is a pullback means that there is a unique lift of π_δ mapping to it. □

6. Future work

This paper has set up an analogy between database queries and constraints on the one hand, and what is now a classical approach to algebraic topology – the lifting problem – on the other. Data on a schema is analogous to a covering space or fibration: the local quality of this fibration is determined by constraints, and the locating of sections that satisfy a set of properties is the posing of a query.

There are a few interesting directions for future research. The first is to make a connection to the relatively new field of *homotopy type theory* (HoTT) – see Awodey and Warren (2009) and Voevodsky (2006). The idea is that instead of two paths through a database schema being *equal*, one could declare them merely *equivalent*, and if paths are declared equivalent in more than one way, these equivalences may also be declared as equivalent (or not). In this context, two observations on data may not be definitionally equal, but provably equal, and we consider the proofs and the differences between proofs as part of the data. To make this connection, the schema of a database should be a quasi-category (Joyal 2002; Lurie 2009) $\mathcal{X}$ rather than an ordinary category. Each higher simplex encodes a proof that different paths (or paths of paths, and so on) through the schema are equivalent. We might replace the instance data by a functor (map of quasi-categories) $\mathcal{X} \to \mathbf{Type}$, where $\mathbf{Type}$ is the quasi-category of homotopy types. In this context, classical homotopical questions, for example, from the theory of model categories (Hirschhorn 2003), may be even more applicable.

Another direction for future research is to use topological tools to investigate or 'mine' data. For example, given a functor $\delta : S \to \mathbf{Set}$, we can compose with the functor $i : \mathbf{Set} \to \mathbf{Top}$ that sends each set to the corresponding discrete topological space. The homotopy colimit of $i \circ \delta$ is a topological space, of possibly any dimension and homotopy type, that encodes the connection pattern of the data. This space is homotopy equivalent to the nerve of the data bundle:

$$\mathrm{hocolim}(i \circ \delta) \simeq N(\textstyle\int \delta)$$

(Dugger 2008). Thus, we could report homotopy invariants of the data δ, such as connected components and loops. The question is whether these invariants would be meaningful and useful. For schemas of classical mathematical interest, such as the simplicial indexing category $S = \mathbf{\Delta}^{\mathrm{op}}$, the homotopy colimit of $i \circ \delta$ is exactly what we want – it is the geometric realisation of δ. It remains to be seen whether such homotopy invariants may be useful in other contexts: for example, there may be some connection to the analysis given by persistent homology (Ghrist 2008; Carlsson *et al.* 2004)).

A third and fairly straightforward project would be to adapt Garner's small object argument (Garner 2009) to our notion of constraints. Garner's argument works and provides nice universal properties in the case of what we have called 'universal constraint sets' (see Section 3.2). The question is, if we apply his techniques to local constraints, such as those in Example 3.15 used to declare that one table is the product of two others, does his procedure still result in a discrete opfibration with all the nice universal properties enjoyed in the universal case? We conjecture that it will. We should also check whether the results obtained from that procedure agree with those from the *universal chase procedure* (Deutsch *et al.* 2008). Indeed, they should provide equivalent results since both claim to be universal in the same way.

Acknowledgments

I would like to express my appreciation to Peter Gates. I am also grateful to Henrik Forssell, Rich Haney, Eric Prud'hommeaux and Emily Riehl for many useful discussions.

References

Awodey, S. and Warren, M. A. (2009) Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society* **146** (1) 45–55.

Bancilhon, F and Spyratos, N. (1981) Update semantics of relational views. *ACM TODS* **6** 557–575.

Barr, M. and Wells, C. (2005) *Toposes, triples, and theories* (corrected reprint of the 1985 original published by Springer-Verlag), Reprints in Theory and Applications of Categories **12** 1–287.

Borceux, F. (1994) *Handbook of categorical algebra 1-3*, Encyclopedia of Mathematics and its Applications **50–52**, Cambridge University Press.

Carlsson, G., Zomorodian, A., Collins, A. and Guibas, L. (2004) Persistence barcodes for shapes. In: Scopigno, R. and Zorin, D. (eds.) *Eurographics Symposium on Geometry Processing* 127–138.

Deus, H. F. *et al.* (2010) Provenance of microarray experiments for a better understanding of experiment results. Proceedings of The Second International Workshop on the role of Semantic Web in Provenance Management, Shanghai, China.

Deutsch, A., Nash, A. and Remmel, J. (2008) The Chase Revisited. *Proceedings of Symposium on Principles of Database Systems (PODS)*, ACM.

Diskin, Z. and Kadish, B. (1994) Algebraic graph-oriented=category-theory-based – manifesto of categorizing data base theory. Technical report, Frame Inform Systems.

Dugger, D. (2008) A primer on homotopy colimits. ePrint available at `http://math.uoregon.edu/~ddugger/hocolim.pdf`.

Ehresmann, C. (1968) Esquisses et types des structures algèbriques. *Buletinul Institutului Politehic din Iasi (N.S.)* **14** (18) (1-2) 1–14.

Gambino, N. and Kock, J. (2013) Polynomial functors and polynomial monads. *Mathematical Proceedings of the Cambridge Philosophical Society* **154** 153–192.

Garner, R. (2009) Understanding the small object argument. *Applied Categorical Structures* **17** (3) 247–285.

Ghrist, R. (2008) Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society (N.S.)* 45 (1) 61–75.

Hartshorne, R. (1977) *Algebraic Geometry*, Graduate Texts in Mathematics **52**, Springer-Verlag.

Hirschhorn, P. (2003) *Model categories and their localizations*, Mathematical surveys and monographs, American Mathematical Society **99**.

Johnson, M. (2001) On Category Theory as a (meta) Ontology for Information Systems Research. *Proceedings of the international conference on Formal Ontology in Information Systems.*

Johnson, M., Rosebrugh, R. and Wood, R. J. (2002) Entity-relationship-attribute designs and sketches. *Theory and Applications of Categories* **10** 94–112.

Johnstone, P. (2002) *Sketches of an elephant 1-2*, Oxford logic guides **43-44**, The Clarendon Press.

Joyal, A. (2002) Quasi-categories and Kan complexes. *Journal of Pure and Applied Algebra* **175** (1-3) 207–222.

Joyal, A. (2010) Catlab. (Available online at `http://ncatlab.org/joyalscatlab/show/Factorisation+systems`.)

Kato, A. (1983) An abstract relational model and natural join functors. *Bulletin of Informatics and Cybernetics* **20** 95–106.

Kelly, G. M. (1974) On clubs and doctrines. In: Kelly, G. M. (ed.) Category Seminar. *Springer-Verlag Lecture Notes in Mathematics* **420** 181–256.

Lurie, J. (2009) *Higher topos theory*, Annals of Mathematical Studies **170**, Princeton University Press.

Mac Lane, S. (1988) *Categories for the working mathematician* (second edition), Graduate texts in mathematics **5**, Springer Verlag.

Mac Lane, S. and Moerdijk, I. (1994) *Sheaves in Geometry and Logic: a first introduction to topos theory*, Universitext, Springer-Verlag.

Makkai, M. (1997) Generalized sketches as a framework for completeness theorems I. *Journal of Pure and Applied Algebra* **115** (1) 49–79.

May, J. P. (1999) *A concise course in Algebraic Topology*, Chicago Lectures in Mathematics, University of Chicago Press.

Morava, J. (2012) Theories of anything. (Available at `http://arxiv.org/abs/1202.0684v1`.)

Prud'hommeaux, E. and Seaborne, A. (eds.) (2008) SPARQL Query Language for RDF: W3C Recommendation 2008/01/15. (Available at `http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/`.)

Quillen, D. G. (1967) Homotopical Algebra. *Springer-Verlag Lecture Notes in Mathematics* **43**.

Spivak, D. I. (2009) Simplicial databases. (Available at `http://arxiv.org/abs/0904.2012`.)

Spivak, D. I. (2012) Functorial data migration. *Information and Computation* **217** 31–51.

Spivak, D. I. and Kent, R. E. (2012) Ologs: A Categorical Framework for Knowledge Representation. *PLoS ONE* **7** (1).

Tuijn, C. and Gyssens, M. (1992) Views and decompositions from a categorical perspective. In: Biskup, J. and Hull, R. (eds.) Database Theory – ICDT '92: Proceedings 4th International Conference. *Springer-Verlag Lecture Notes in Computer Science* **646** 99–112.

Voevodsky, V. (2006) A very short note on the homotopy λ-calculus. Unpublished note.