Representing Knowledge and Querying Data using Double-Functorial Semantics

Michael Lambert
University of Massachusetts-Boston
michael.james.lambert@gmail.com

Evan Patterson
Topos Institute
evan@epatters.org

Category theory offers a mathematical foundation for knowledge representation and database systems. Popular existing approaches model a database instance as a functor into the category of sets and functions, or as a 2-functor into the 2-category of sets, relations, and implications. The functional and relational models are unified by double functors into the double category of sets, functions, relations, and implications. In an accessible, example-driven style, we show that the abstract structure of a 'double category of relations' is a flexible and expressive language in which to represent knowledge, and we show how queries on data in the spirit of Codd's relational algebra are captured by double-functorial semantics.

1 Introduction

Knowledge representation and databases are among the most successful applications of category theory, supported not only by several decades of theoretical development [6, 11, 13, 14] but also by substantial practical and industrial deployments [10, 15]. The general scheme is that the *ontology* or *database schema* is a small category, possibly equipped with extra structure, and *instance data* is a structure-preserving functor out of the schema into a category such as Set. Under this scheme, knowledge representation and databases are two sides of the same coin: knowledge representation emphasizes rich schemas and expressive schema languages (categorical structure), whereas databases center the data instances and transformations between them. We will pass seamlessly between the terminologies of the two fields, as the distinction is artificial.

In Spivak's category-theoretic model of a database [13], a *schema* is nothing more than a small category and an *instance* is a set-valued functor. With an eye toward knowledge representation, Spivak and Kent explored a richer language [14], defining an *ontology log*, or *olog*, to be a finite-limit, finite-colimit sketch and an *instance* to be a set-valued model of the sketch. In both cases the language is "functional," as arrows in the schema are interpreted as functions between sets. Since much technology in knowledge representation, including the Web Ontology Language (OWL), favors relations over functions, the last author proposed an alternative notion of *relational olog* [8], taking Carboni and Walters' *'bicategories of relations'* as the basic structure [2]. An ontology now becomes a small 'bicategory of relations' and an instance is a structure-preserving 2-functor from it into **Rel**. In other words, morphisms in the schema are now interpreted as relations; also, there are 2-cells interpreted as implications.

We introduce double-categorical ologs, taking the first author's *'double categories of relations'* as the foundational structure [7]. In outline, a *double olog* will be a small double category, assumed locally posetal for simplicity and equipped with certain extra structure, and an *instance* will be a structure-preserving double functor into $\mathbb{R}$el, the double category of sets, functions, relations, and implications. As an obvious first benefit, this approach recognizes that functions and relations are fundamental concepts and grants them both first-class status. From the perspective of categorical logic, a double olog possesses

all four elements of a logic fibered over a type theory [5]: its objects are types and its arrows are terms; fibered over those, its proarrows are predicates and its cells are judgments or implications. By contrast, a functional olog is missing predicates and judgments, whereas a relational olog is missing terms.

Using the rich structure of a 'double category of relations', queries on the instance data of a double olog can be formulated internally to the schema, rather than through external mappings. In Spivak's minimal model of databases [13, §5.3], queries are regarded as particular cases of the adjoint triple of data migration functors induced by a schema mapping. This approach, while mathematically economical, is indirect and requires delicate analysis to reduce to SQL primitives [15]. By contrast, the basic operations of Codd's relational algebra [3, §2.1]—permutations, projections, joins—appear directly as operations in a 'double category of relations', namely as restrictions or extensions along symmetries, projections, or diagonals. Thus, as we show in Section 3, queries can be formulated as abstract relations in the schema and then evaluated by the double functor defining the instance. As a slogan, we say that *querying is double-functorial semantics*.

In this paper, we will explain the features of double ologs in an accessible, example-driven style, but for the benefit of readers familiar with double category theory, we briefly state the main technical definitions. Further background can be found in Appendix A. A **'double category of relations'** is a locally posetal cartesian equipment satisfying a discreteness or Frobenius axiom [7, Definition 3.2]. We define a **double olog**, or just an **olog** for short, to be a small 'double category of relations', which we sometimes also assume to have tabulators. An **instance** of a double olog $\mathbb{D}$ is a cartesian, strict double functor $\mathbb{D} \to \mathbb{R}\mathsf{el}$, which must also preserve tabulators whenever $\mathbb{D}$ is assumed to have them.

Acknowledgments Patterson was supported by the Air Force Office of Scientific Research (AFOSR) Young Investigator Program (YIP) through Award FA9550-23-1-0133.

2 Representing Knowledge

The two fundamental ingredients of a functional olog [14] are concepts (aka, types or objects) and attributes (aka, arrows or morphisms), schematized for example as

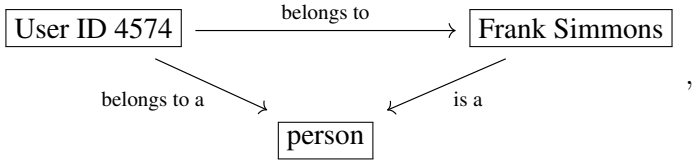
$$\boxed{\text{Sam Carter}} \xrightarrow{\text{is a}} \boxed{\text{person}} \ ,$$

where the domain stands for a concept of *being Sam Carter*, the codomain represents the concept of *personhood*, and the arrow expresses that personhood is an attribute of Sam Carter. (All of our examples are inspired by the television show Stargate SG-1, knowledge of which should enhance the reader's enjoyment, if not their understanding, of the paper.) This section will show how double categories—and in particular cartesian equipments and 'double categories of relations'—can enhance the expressive power of ologs.

2.1 Expressing Facts

A **fact** in an olog is defined by Spivak and Kent [14] to be a commutative diagram. In other words, a fact is a statement that a pair of sequences of attributes have the same composite. Suppose we wish to assert

the fact that Frank Simmons is User 4574. In a functional olog, we could write

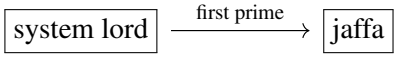


,

which says that Frank Simmons is the person to whom the ID 'User 4574' belongs.

However, something seems wrong about encoding individual people as types or objects in a categorical structure. Ordinarily, in first-order logic and type theory, individual entities are specified in a signature as constants of a certain type. In category theory, constants are interpreted as *global elements*, that is, as arrows $c\colon 1 \to A$ from a chosen terminal object to the type A. Following this approach, we would rather fix a terminal object and posit two individual constants.

We start with an olog asserting that every System Lord has a First Prime, foremost among Jaffa warriors belonging to that System Lord. This could be expressed by a function

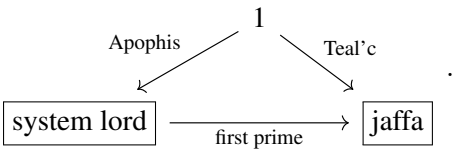


.

Let's posit two individuals, Teal'c and Apophis, whom we include in the olog as constants

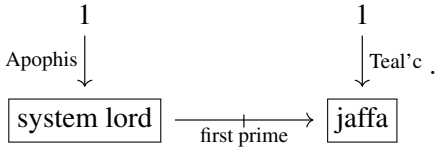
.

The fact that Teal'c is (or rather was) the First Prime of Apophis can be expressed as above by a commutative diagram



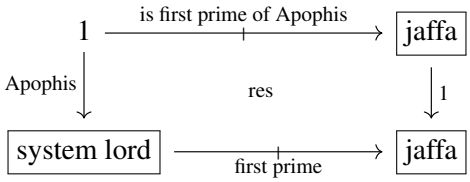
.

Now suppose we'd like to take a more nuanced view of the relationship "first prime." The phrasing above carries the assumption that to each System Lord we can assign a unique First Prime, which is arguably true at a fixed moment in time. But what if at that moment in time a System Lord is between First Primes, so that in fact we have have a partial function? Or perhaps we would like our data to include all of the First Primes a System Lord has ever had (a one-to-many relationship), or similarly we'd like to consider all of the System Lords a given Jaffa has served as First Prime. To accommodate these possibilities, we will alter the olog to specify a *relation* of being First Prime, along with two constants:

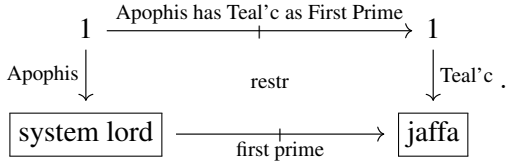


.

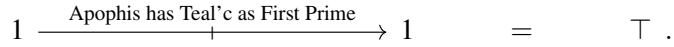
Restricting along the Apophis constant creates the cell



whose image under instance data valued in $\mathbb{R}\mathsf{el}$ would amount to a list of all of Jaffa having served Apophis as first prime. How can we express the fact that Teal'c is (or was at some point) a first prime of Apophis? This is a matter of asking that another restriction factor through (or rather be equal to) the truth value $\top$, which is the terminal object in the external hom-category on 1. That is, we first form the restriction along both constants



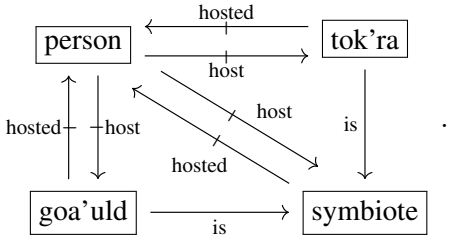
Denoting our double olog by $\mathbb{D}$, we have the truth value $\top\colon 1 \nrightarrow 1$, the terminal object of $\mathbb{D}(1,1)$. The fact that Apophis has (or had) Teal'c as a First Prime is just asking that the restricted proarrow above be equal to the local terminal:



This assertion cannot be so naturally expressed in either functional or relational ologs. In a functional olog, we must treat the relation of being First Prime as functional, which is too restrictive. On the other hand, in a relational olog, we can only introduce a constant through a relation $c : 1 \nrightarrow A$ along with side equations making the relation into a *map* in the 'bicategory of relations' [2, Definition 1.5 and Lemma 2.5]. We then rely on the semantics of the olog in **Rel** to ensure that the relation c is interpreted as (the graph of) a function. In a double olog, we work natively with both functions and relations as is convenient.

2.2 Creating Types Using Tabulators

Here is an another double olog, expressing a few relationships between persons, symbiotes, and types of symbiotes, namely, Goa'uld and Tok'ra:



Every person is a potential symbiote host. But notice that the relation host: person $\nrightarrow$ symbiote and those valued in tok'ra and goa'uld are not functional since not every person is in fact a host. An

instance of "hosted" is generally many-to-one since many symbiotes have at least one host. It should be asked in equations or as a definition that $\text{host}^\dagger = \text{hosted}$, where the dagger $(-)^\dagger$ denotes the converse or opposite relation. All Tok'ra are symbiotes; so are Goa'uld. Thus, the two arrows labeled 'is' are genuinely functional, indicating something like subtypes.

Typically a Goa'uld leaves its host only upon the the death of the host, and if the Goa'uld is forcibly removed, then the host will die due to release of toxins. So, simplifying slightly, we can make the assumption that each person is a host at most once. This is expressed by imposing the equation

$$\text{id} \quad = \quad \boxed{\text{goa'uld}} \xrightarrow{\text{hosted}} \boxed{\text{person}} \xrightarrow{\text{host}} \boxed{\text{goa'uld}}$$

asserting that the host and hosted relation compose to the identity on $\boxed{\text{goa'uld}}$. In other words, the relation host is a *partial map*. Theorem A.1 shows that if the double olog $\mathbb{D}$ were taken to be a unit-pure 'double category of relations' with tabulators, then this assumption would imply that the tabulator of the host-relation, namely $\boxed{\text{host-goa'uld pairs}}$, is the apex of a span

$$\boxed{\text{person}} \xleftarrow{d} \boxed{\text{host-goa'uld pairs}} \xrightarrow{c} \boxed{\text{goa'uld}}$$

where d is monic, that is, a partial map in the underlying category $\mathbb{D}_0$ of objects and arrows. If the tabulators in $\mathbb{D}$ are also assumed to be *strong*, then the process is reversible, in the sense that a partial map (a span with monic left leg) would induce a partial morphism in the relational sense. Thus, under suitable assumptions, one can pass back and forth between the two possible representations of a partial map in a double olog.

Even without these extra conditions, the mere presence of extensions and tabulators increases the expressive power of double ologs considerably. For example, we can create a type of persons who are or have been hosts. First, create a proposition for the notion *has hosted* as an extension cell

$$\begin{array}{ccc} \boxed{\text{person}} & \xrightarrow{\text{host}} & \boxed{\text{symbiote}} \\ {\scriptstyle =}\downarrow & \text{ext} & \downarrow{\scriptstyle !} \\ \boxed{\text{person}} & \xrightarrow[\text{has hosted}]{} & 1 \end{array} .$$

This is interpreted as the image of the host relation in the type of persons. Then extract the comprehension or subobject classified by this proposition by taking its tabulator. This gives us a type of hosts:

$$\begin{array}{ccc} & \boxed{\text{host}} & \\ {\scriptstyle \text{is}}\swarrow & \text{tab} & \searrow{\scriptstyle !} \\ \boxed{\text{person}} & \xrightarrow[\text{has hosted}]{} & 1 \end{array} .$$

Likewise, there is a type of hosted symbiotes constructed by the extension on the left followed by the tabulator on the right:

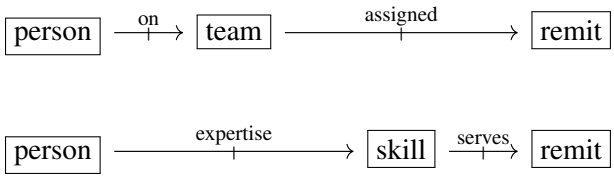
$$\begin{array}{ccc} \boxed{\text{person}} & \xrightarrow{\text{host}} & \boxed{\text{symbiote}} \\ {\scriptstyle !}\downarrow & \text{ext} & \downarrow{\scriptstyle =} \\ 1 & \xrightarrow[\text{has been hosted}]{} & \boxed{\text{symbiote}} \end{array} \quad \rightsquigarrow \quad \begin{array}{ccc} & \boxed{\text{hosted symbiote}} & \\ {\scriptstyle !}\swarrow & \text{tab} & \searrow{\scriptstyle \text{is}} \\ 1 & \xrightarrow[\text{has been hosted}]{} & \boxed{\text{symbiote}} \end{array} .$$

The arrow on the right again represents a kind of subtype since there are symbiotes such as queens who are not always hosted and symbiotes such as prim'ta who are never hosted for whatever reason. In short, tabulators enable propositions, and relations generally, to be reified as subtypes of the original types.

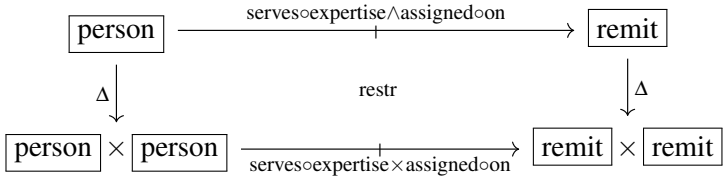
2.3 Forming Conjunctions Using Local Products

What if we want to assert as a fact that two propositions both hold or that they are equal? That can be done using *local products* (Appendix A.2).

Let's look again at an example. Consider the olog with four relations:



A fairly complex hypothetical underlies this olog, captured by the question of whether the expertise of given persons serves the remit of the teams of which they are members. We could require by fiat an equality between these objects in the appropriate hom-category. In some sense this *should be the case* if the teams have been formed correctly. But perhaps that's not necessarily the case, and we'd rather ask about *the extent to which* our question is answered in the affirmative. We can do this using a local product, which, in the example at hand, looks like



Let's postulate some concrete data to see what this local product looks like. Define a data instance $\mathbb{D} \to \mathbb{R}\mathrm{el}$ for our double olog $\mathbb{D}$ by assigning the types to the data columns

person
Carter
Connor
Jackson
Kovacek
Teal'c

team
SG1
SG3
SG9
SG11

skill
archaeology
anthropology
astrophysics
engineering
law
philology
tracking
weapons

remit
combat
diplomacy
exploration
science & engineering
search & rescue

The first two relations are then the following:

on	
person	**team**
Carter	SG1
Connor	SG9
Jackson	SG1
Kovacek	SG9
Teal'c	SG1
Teal'c	SG3

assigned	
team	**remit**
SG1	exploration
SG3	combat
SG3	search & rescue
SG9	diplomacy
SG11	science & engineering

and the remaining two are the big tables filled with all of Daniel Jackson's PhDs:

expertise	
person	**skill**
Carter	astrophysics
Carter	weapons
Connor	engineering
Jackson	anthropology
Jackson	archaeology
Jackson	philology
Kovacek	law
Teal'c	weapons
Teal'c	tracking

serves	
skill	**remit**
astrophysics	exploration
anthropology	exploration
archaeology	exploration
engineering	science & engineering
law	diplomacy
philology	exploration
weapons	combat
weapons	exploration
tracking	search & rescue

.

None of these relations are functional. Perhaps some of the associations in the **serves** relation are debatable and certainly we haven't included all skills possessed by all the individuals in the **person** table, but the point is just to show how local products work. One can trace through the construction as a restriction first in $\mathbb{D}$ and then in $\mathbb{R}\mathsf{el}$. But the local products formula in Equation (A.1) above makes the computation easy. The local product is just the set of all $\boxed{\text{person}}$-$\boxed{\text{remit}}$ pairs related under *both* composites serves $\circ$ expertise and assigned $\circ$ on. To be explicit, we can compute the two composites as

serves ∘ expertise	
person	**remit**
Carter	exploration
Carter	combat
Connor	science & engineering
Jackson	exploration
Kovacek	diplomacy
Teal'c	combat
Teal'c	search & rescue

assigned ∘ on	
person	**remit**
Carter	exploration
Connor	diplomacy
Jackson	exploration
Kovacek	diplomacy
Teal'c	exploration
Teal'c	combat
Teal'c	search & rescue

.

In each case we've used the formula for composition of relations using existential quantification. Notice that despite Laurence Connor's training in aerospace engineering, he serves on the diplomacy team SG9, while Sam Carter's weapons training would qualify her for combat, but she is instead a member of

exploration team SG1. Thus, the local product is the table excluding these two entries, namely

serves ∘ expertise ∧ assigned ∘ on	
person	**remit**
Carter	exploration
Jackson	exploration
Kovacek	diplomacy
Teal'c	combat
Teal'c	search & rescue

.

The table thus gives precisely those people whose expertise serves the purpose of the teams to which they are assigned as well as the particular remit.

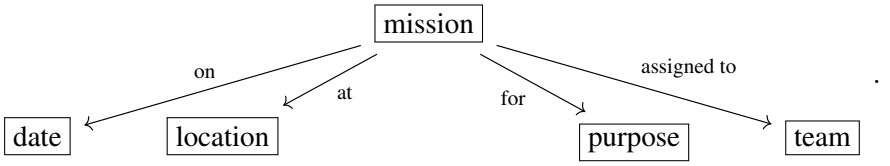
3 Querying Data

The conceit of this section is that 'double categories of relations' provide a simple and satisfying framework for querying data. In the approach based on functorial data migration [13], queries are handled externally through the device of adjoint functors between copresheaf categories and regarded as a special case of data migration. We will see here that queries can be expressed internally to the schema using the operations available in a 'double category of relations', or more generally in a cartesian equipment. As a slogan: *querying is double-functorial semantics*.

3.1 Select

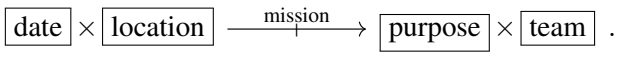
The select operation is perhaps the easiest to describe in our double-categorical formalism. Selection is performed by taking an extension along projection morphisms.

Suppose we are describing the attributes belonging to a concept of a mission. In a functional olog [14], we would schematize the concept with functional relations indicated by the arrows identifying the leaf tables, such as:



.

However, this multi-span is just an encoding of a relation, which we choose to present double-categorically as a quaternary relation:



.

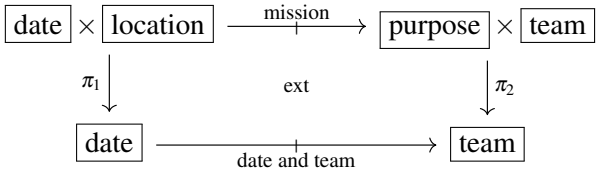
Of course, we could have chosen to partition the four types differently, but that would not meaningfully alter the result. We display the instance data for the relation in the conventional style as a multi-column

table:

mission			
date	**location**	**purpose**	**team**
2-6-98	P41-771	search & rescue	SG3
7-31-98	Cimmeria	assist Cimmerians	SG1
1-2-99	P3R-272	investigate inscriptions	SG1
10-22-99	Ne'tu	search & rescue	SG1
8-6-2004	Tegalus	negotiation	SG9

We'll assume in the background that the types have been instanced with data, but it's not necessary to say at this point what the data actually is.

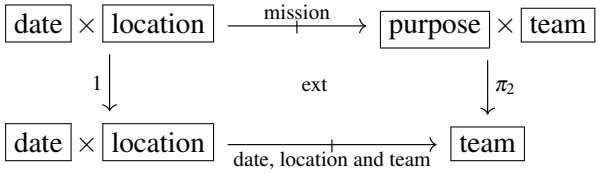
Selecting columns is now simply a matter of extending the relation along projections. For example, say we want only to remember the dates that teams went on missions but we care neither what the location nor purpose of the missions were. In this case, we take an extension along the projection morphisms as shown:



In the double category $\mathbb{R}$el, the extension is computed by taking an image. That is, a date and team are in the extension relation if, and only if, they are related (with two other elements of some 4-tuple) in the original relation. In this case, we end up with a binary relation instanced by the table

date and team	
date	**team**
2-6-98	SG3
7-31-98	SG1
1-2-99	SG1
10-22-99	SG1
8-6-2004	SG9

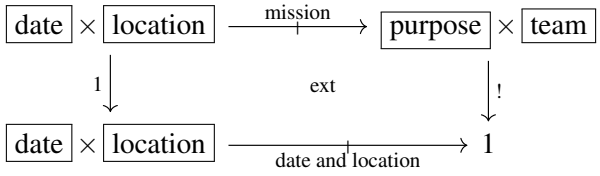
Alternatively, we might ask just for the occasions on which teams have visited locations disregarding the purpose of the mission. This would be accomplished by an extension yielding a ternary relation with accompanying table



date, location and team		
date	**location**	**team**
2-6-98	P41-771	SG3
7-31-98	Cimmeria	SG1
1-2-99	P3R-272	SG1
10-22-99	Ne'tu	SG1
8-6-2004	Tegalus	SG9

Finally, we might want to know only the date on which a location was visited. This would be an extension

using the terminal object on one side



resulting in the data

date and location	
date	**location**
2-6-98	P41-771
7-31-98	Cimmeria
1-2-99	P3R-272
10-22-99	Ne'tu
8-6-2004	Tegalus

.

We take these examples to be sufficient evidence that the select operation can be performed by extending along projection morphisms. The point is simply to use whichever projections return the desired columns.

3.2 Filter

The operation of *filtering* data asks for rows that satisfy a certain property, in the simplest case that a particular attribute has a specific value. This is one of the last topics of [13], treated in §5.3 in an example using adjoint functors arising from slices of copresheaf toposes. In a double olog, filtering can be performed using restrictions provided we allow ourselves types in the olog isolating the attribute value that we're looking for.

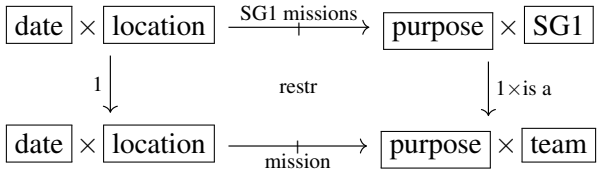
Suppose we wish to filter the previous data for just the missions assigned to the flagship team SG1. As explained in Section 2.1, we can include the individual team SG1 in our olog in two different ways. First, we could add an arrow

$$\boxed{\text{SG1}} \xrightarrow{\text{is a}} \boxed{\text{team}} \ .$$

On the other hand, we could stipulate that SG1 is rather the name of a global element of the type, namely, an arrow from the terminal

$$1 \xrightarrow{\text{SG1}} \boxed{\text{team}} \ .$$

Both approaches suffice to filter the data but return slightly different arrangements. In the former case, we take the restriction

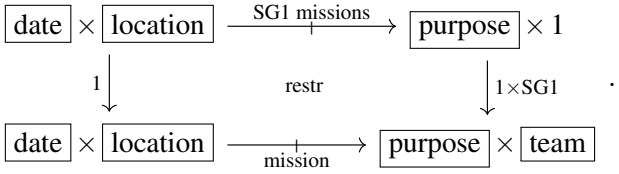


and the data returned is a subtable consisting only of certain rows of the original **mission** table, namely

SG1 missions			
date	**location**	**purpose**	**team**
7-31-98	Cimmeria	assist Cimmerians	SG1
1-2-99	P3R-272	investigate inscriptions	SG1
10-22-99	Ne'tu	search & rescue	SG1

.

Of course, the last column on the right is superfluous since we are specifically filtering for that particular value. This redundancy can be avoided using the global element approach. In this case we have to be especially careful to label the restricted relation in the olog. That is, we *define* the relation "SG1 missions" to be the restriction



.

This returns the simplified table omitting the redundant column:

SG1 missions		
date	**location**	**purpose**
7-31-98	Cimmeria	assist Cimmerians
1-2-99	P3R-272	investigate inscriptions
10-22-99	Ne'tu	search & rescue

.

If in addition we wanted to select certain columns (say we were interested only in the dates on which SG1 visited certain places), we could use the projection-extension method above to produce a table with only those columns. Such a combination is effectively a *select-from-where* query in SQL, restricted to a single table. We now turn to how such queries can be extended to multiple tables.

3.3 Inner Joins

The inner join operation is an example of local products, already used in Section 2.3 and reviewed in Appendix A.2. Let's take a simple olog, namely, a fragment of the teams olog above with the two relations

.

These relations are intended to express that a person may have expertise consisting of training or facility in a particular skill, and that each person may or may not belong to a particular SGC team. Denote this

double olog by $\mathbb{D}$. As instance data $\mathbb{D} \to \mathbb{R}\mathsf{el}$, let us first assign the following data to the types:

person
Hammond
Kovacek
Maybourne
Morrison
O'Neill
Rothman
Simmons
Warren

skill
archaeology
chicanery
combat
command
law
sociopathy

teams
SG1
SG3
SG9
SG11

.

We then assign the following tables to the two relations:

expertise	
person	**skill**
Hammond	command
Kovacek	law
Maybourne	chicanery
Morrison	combat
O'Neill	command
Rothman	archaeology
Simmons	sociopathy
Warren	combat

membership	
person	**team**
Kovacek	SG9
Morrison	SG3
O'Neill	SG1
Rothman	SG11
Warren	SG3

.

As always, we assume that this data is assigned by a cartesian strict double functor $\mathbb{D} \to \mathbb{R}\mathsf{el}$. In our example, the inner join along the shared column $\boxed{\text{person}}$ is given by the restriction

$$\begin{array}{ccc} \boxed{\text{person}} & \xrightarrow{\text{expertise}\bowtie\text{membership}} & \boxed{\text{skill}} \times \boxed{\text{team}} \\ \Big\downarrow{\scriptstyle \Delta} & \text{restr} & \Big\downarrow{\scriptstyle 1} \\ \boxed{\text{person}} \times \boxed{\text{person}} & \xrightarrow[\text{expertise}\times\text{membership}]{} & \boxed{\text{skill}} \times \boxed{\text{team}} \end{array} .$$

In relations, the restriction is computed as a pullback of the product along the arrow $\Delta \times 1$. That is, form the cartesian product of all pairs from the two relations and match the $\boxed{\text{person}}$ argument. We obtain the following table

expertise ⋈ membership		
person	**skill**	**team**
Kovacek	law	SG9
Morrison	combat	SG3
O'Neill	command	SG1
Rothman	archaeology	SG11
Warren	combat	SG3

recording both the skill and team of those persons who have both listing without any extra rows or null values. This is precisely the inner join of the two tables. Notice also that this approach avoids repeating the person column in the joined table.

References

[1] Evangelia Aleiferi (2018): *Cartesian double categories with an emphasis on characterizing spans*, PhD thesis. Dalhousie University, arXiv: 1809.06940.

[2] Aurelio Carboni & Robert F.C. Walters (1987): *Cartesian bicategories I. Journal of Pure and Applied Algebra*. 49(1-2), pages 11–32, doi: 10.1016/0022-4049(87)90121-6.

[3] Edgar F. Codd (1970): *A relational model of data for large shared data banks*. *Communications of the ACM*. 13(6), pages 377–387, doi: 10.1145/362384.362685.

[4] Marco Grandis (2019): *Higher dimensional categories: From double to multiple categories*. World Scientific. doi: 10.1142/11406.

[5] Bart Jacobs (1999): *Categorical logic and type theory*. Elsevier.

[6] Michael Johnson, Robert Rosebrugh & R.J. Wood (2002): *Entity-relationship-attribute designs and sketches*. *Theory and Applications of Categories*. 10(3), pages 94–112. Available at http://www.tac.mta.ca/tac/volumes/10/3/10-03abs.html.

[7] Michael Lambert (2022): *Double categories of relations*. *Theory and Applications of Categories*. 38(33), pages 1249–1283, arXiv: 2107.07621. Available at http://www.tac.mta.ca/tac/volumes/38/33/38-33abs.html.

[8] Evan Patterson (2017): *Knowledge representation in bicategories of relations*, arXiv: 1706.00526.

[9] Evan Patterson (2024): *Products in double categories, revisited*, arXiv: 2401.08990.

[10] Evan Patterson, Owen Lynch & James Fairbanks (2022): *Categorical data structures for technical computing*. *Compositionality*. 4(5), doi: 10.32408/compositionality-4-5, arXiv: 2106.04703.

[11] Frank Piessens & Eric Steegmans (1995): *Categorical data-specifications*. *Theory and Applications of Categories*. 1(8), pages 156–173. Available at http://www.tac.mta.ca/tac/volumes/1995/n8/1-08abs.html.

[12] Michael Shulman (2008): *Framed bicategories and monoidal fibrations*. *Theory and Applications of Categories*. 20(18), pages 650–738, arXiv: 0706.1286. Available at http://www.tac.mta.ca/tac/volumes/20/18/20-18abs.html.

[13] David I. Spivak (2012): *Functorial data migration*. *Information and Computation*. 217, pages 31–51, doi: 10.1016/j.ic.2012.05.001, arXiv: 1009.1166.

[14] David I. Spivak & Robert E. Kent (2012): *Ologs: A categorical framework for knowledge representation*. *PloS One*. 7(1), e24274, doi: 10.1371/journal.pone.0024274, arXiv: 1102.1889.

[15] David I. Spivak & Ryan Wisnesky (2015): *Relational foundations for functorial data migration*. In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 21–28, doi: 10.1145/2815072.2815075, arXiv: 1212.5303.

A Background on Double Category Theory

In this appendix, we review background material on double category theory, as well as prove a relevant new characterization of partial maps in a 'double category of relations.' A general reference on double category theory is the textbook by Grandis [4].

A.1 Double Categories of Relations

Briefly, a **'double category of relations'** [7] is a double category that is also cartesian and an equipment and that satisfies a discreteness condition distinguishing the 'bicategories of relations' of [2] from mere cartesian bicategories. For simplicity we also assume that any such 'double category of relations' is locally posetal. The usual double category of relations $\mathbb{R}\mathsf{el}$ is the canonical example. That a 'double category of relations' $\mathbb{D}$ is cartesian means that it has finite products in an appropriately coherent way [1], namely, the canonical double functors $\mathbb{D} \to \mathbb{D} \times \mathbb{D}$ and $\mathbb{D} \to 1$ have right adjoints in the 2-category of double categories, (pseudo)-double functors and transformations. That a 'double category of relations' is an equipment [12] means that the canonical source-target projection functor $\mathbb{D}_1 \to \mathbb{D}_0 \times \mathbb{D}_0$ is a bifibration.

In more practical terms, the latter bifibration condition is equivalent to the fact that niches and coniches can be completed to proper cells

$$\begin{array}{ccc} A & \dashrightarrow & B \\ {\scriptstyle f}\downarrow & \text{res} & \downarrow{\scriptstyle g} \\ C & \xrightarrow[S]{} & D \end{array} \qquad\qquad \begin{array}{ccc} A & \xrightarrow{R} & B \\ {\scriptstyle h}\downarrow & \text{ext} & \downarrow{\scriptstyle k} \\ C & \dashrightarrow & D \end{array}$$

that are (respectively) cartesian and opcartesian with respect to the projection functor $\mathbb{D}_1 \to \mathbb{D}_0 \times \mathbb{D}_0$. The cartesian cell on the left is thought of as a *restriction* since in $\mathbb{R}\mathsf{el}$ it is given by taking a pullback of the relation S along the pair $f \times g$. On the other hand, the opcartesian cell on the right is an *extension*, since in $\mathbb{R}\mathsf{el}$ it would be given by taking an image.

Such restrictions and extensions, together with the product structure, do most of the querying work in Section 3. *Local products* make sense in this context and are discussed below in Appendix A.2. A 'double category of relations' has a further property, however, namely, that of satisfying a *discreteness axiom*, which ensures that its horizontal bicategory is in fact a compact closed monoidal category. This gives a good notion of partial map which is discussed below in Appendix A.3.

A double category $\mathbb{D}$ has **tabulators** if the external identity functor $\mathbb{D}_0 \to \mathbb{D}_1$ has a right adjoint $\top$. This associates to each proarrow an object and a universal cell that we think of as a kind of *comprehension scheme*. Tabulators provide type creation in double categorical ologs by associating a type to a proarrow of those entities which satisfy the proposition represented by the given proarrow.

Our data tables will be given by strict, cartesian double functors $M\colon \mathbb{D} \to \mathbb{R}\mathsf{el}$ valued in the double category of relations. Strict means that the associators and unitors are not mere isomorphisms, but are in fact equalities. Pseudo, hence strict, double functors preserve restriction and extension cells [12, §6]. Strict cartesian double functors $M\colon \mathbb{D} \to \mathbb{R}\mathsf{el}$ also preserve local products, as discussed immediately below. These preservation properties are used through Section 3 of the paper and those calculations support our conceit that querying is functorial semantics.

A.2 Local Products

Given two proarrows, say $R\colon X \nrightarrow Y$ and $S\colon X \nrightarrow Y$ in a cartesian equipment $\mathbb{D}$, the **local product** of R and S is defined to be the proarrow domain of the restriction along the diagonals

$$\begin{array}{ccc} X & \xrightarrow{R \wedge S} & Y \\ {\scriptstyle \Delta}\downarrow & \text{restr} & \downarrow{\scriptstyle \Delta} \\ X \times X & \xrightarrow[R \times S]{} & Y \times Y \end{array} .$$

It computes a binary product in the hom-category $\mathbb{D}(X,Y)$. It's pretty easy to see that in $\mathbb{R}\mathsf{el}$, this local product is computed as a certain set of pairs. Namely, since it's a restriction, the resulting relation is the monic on the leftside of

$$\begin{array}{ccc} R \wedge S & \longrightarrow & R \times S \\ \downarrow & & \downarrow \\ X \times Y & \xrightarrow[\Delta_X \times \Delta_Y]{} & X \times X \times Y \times Y \end{array}$$

where we've forgotten about one of the associativity isomorphisms in the product on the right. In any case, $R \wedge S$ in $\mathbb{R}\mathsf{el}$ is the set of pairs

$$R \wedge S = \{(x,y) \mid xRy \text{ and } xSy\}. \tag{A.1}$$

Thus, local products can be constructed by restricting products of relations along diagonal arrows. Alternatively, local products can viewed as a primitive notion defined by their own double-categorical universal property [9, Example 6.10], and any cartesian equipment possesses finite local products in this sense [9, Corollary 8.7].

Inner joins can be constructed by adapting local products, which involves matching along a diagonal. That is, given two proarrows $R\colon A \times B \nrightarrow C$ and $S\colon B \nrightarrow D$, we define the **inner join of R and S along B** to be the restriction cell

$$\begin{array}{ccc} A \times B & \xrightarrow[]{R \bowtie S} & C \times D \\ {\scriptstyle 1\times\Delta}\downarrow & \text{restr} & \downarrow{\scriptstyle 1} \\ A \times B \times B & \xrightarrow[R \times S]{} & C \times D \end{array} .$$

Think of this as giving two tables, namely, R and S with entries from sets A, B, C and D. The tables have entries in one column in common, namely, those from the set B. We take the cartesian product of the relations $R \times S$ and then pull back along the diagonal morphism Δ on B asking that those projection values are the same while leaving the others alone.

A.3 Partial Maps

Any ordinary relation $R\colon A \nrightarrow B$ has a reverse relation $R^\dagger\colon B \to A$ which exchanges the domain and codomain. Suppose that $R\colon A \nrightarrow B$ is a relation whose associated span

$$\begin{array}{ccc} R & \xrightarrow{c} & B \\ {\scriptstyle d}\downarrow & & \\ A & & \end{array}$$

is a *partial map* in the sense that d is monic. Define a morphism $\varepsilon\colon R \times_A R \to B$ taking (r,s) in the pullback of d along itself to $cr = cs$, where equality is implied by the fact that d is monic. This map ε defines a cell

$$\begin{array}{ccc} R \times_A R & \xrightarrow{\langle c,c\rangle} & B \times B \\ {\scriptstyle \varepsilon}\vdots & \text{(I)} & \downarrow{\scriptstyle 1} \\ B & \xrightarrow[\Delta]{} & B \times B \end{array}$$

which amounts to the condition that $R^\dagger \otimes R \leq \mathrm{id}_B$ holds in $\mathbb{R}\mathsf{el}$.

The converse is true. Consider any fixed relation $R\colon A \nrightarrow B$ and its reverse relation $R^\dagger\colon B \nrightarrow A$. Suppose that $R^\dagger \otimes R \leq \mathrm{id}_B$ holds. There is then is a cell $\varepsilon\colon R^\dagger \otimes R \to B$ which is an arrow making a commutative square of the form of (I) above. The existence of ε forces d to be monic. For if $r = (x,y)$ and $s = (x,z)$ are given elements of $R \times_A R$ (which is the same as supposing that $dr = ds$), we have that

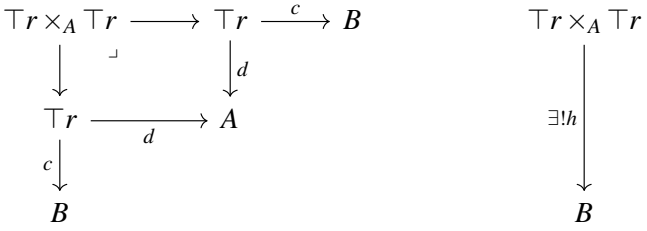
$$\Delta\varepsilon(r,s) = \langle c,c\rangle(r,s) = (y,z) \tag{A.2}$$

is a diagonal entry of $B \times B$ by (I) above meaning that $y = z$ must hold. Thus, the span consisting of d and c is a genuine partial map $A \to B$.

A general version for the necessity of the condition is valid in any unit-pure (the external identity functor is fully faithful) 'double category of relations' with tabulators $\mathbb{D}$. Recall that in this case the horizontal bicategory $\mathbf{H}(\mathbb{D})$ is then a cartesian bicategory [7, Proposition 3.1] and indeed a $\mathbf{H}(\mathbb{D})$ is a 'bicategory of relations', hence a compact closed monoidal category with involution on proarrows denoted by $(-)^\dagger$ [2, Theorem 2.4]. Recall from [2, §2] that a proarrow $r\colon A \nrightarrow B$ is a **partial map** if $r^\dagger \otimes r \leq \mathrm{id}_B$ holds.

Theorem A.1. *If $\mathbb{D}$ is unit-pure and has tabulators, then $r\colon A \nrightarrow B$ is a partial map only if $d\colon \top r \to A$ is a monic arrow, meaning, in other words, that the span formed by the legs of $\top r$ is a partial map in the sense that the domain arrow d is monic.*

Proof. Suppose that r is a partial map and suppose as given two morphisms $f,g\colon X \rightrightarrows \top r$ satisfying $df = dg$. Using the cell $r^\dagger \otimes r \leq \mathrm{id}_B$ and "unit-pure" there is a unique morphism h that is equal to each of the legs in the outside of the figure on the left in



in that $c\pi_1 = h = c\pi_2$ holds. Thus, f and g induce a pair morphism $\langle f,g\rangle\colon X \to \top r \times_A \top r$, and we can calculate that

$$cf = c\pi_1\langle f,g\rangle = h\langle f,g\rangle = c\pi_2\langle f,g\rangle = cg \tag{A.3}$$

meaning that $f = g$ must hold as a result of the uniqueness clause of the universal property of the tabulator $\top r$. Therefore, d is monic. □