Informal Data Transformation Considered Harmful

Eric Daimler, Ryan Wisnesky
Conexus AI

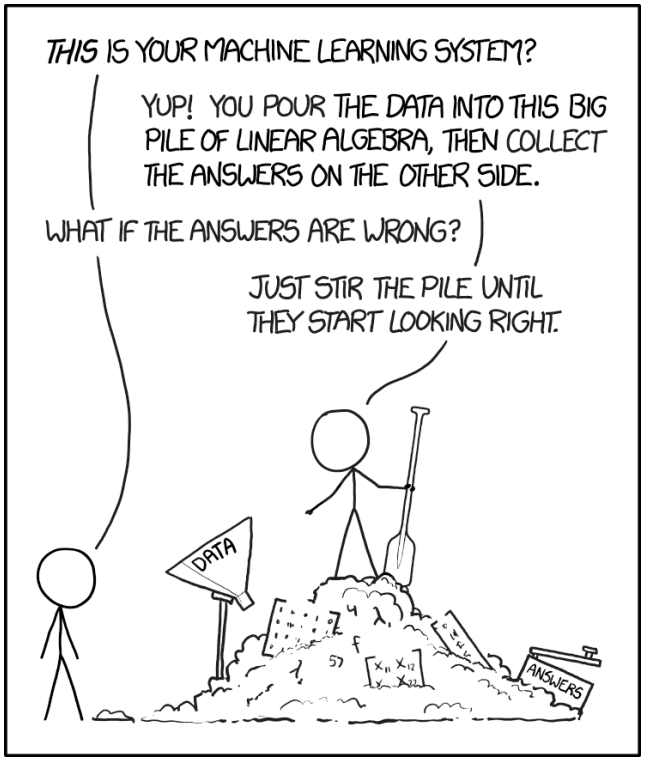


Image used under a creative commons license; original available at http://xkcd.com/1838/.

1 Introduction

In this paper we take the common position (Wickham 2014) that AI systems are limited more by the integrity of the data they are learning from than the sophistication of their algorithms, and we take the uncommon position that the solution to achieving better data integrity in the enterprise is not to clean and validate data ex-post-facto whenever needed[1], but rather to formally and automatically guarantee that data integrity is preserved as it transformed (migrated, integrated, composed, queried, viewed, etc) throughout the enterprise, so that data and programs that depend on that data need not constantly be re-validated for every particular use. Computer scientists have been developing techniques for preserving data integrity during transformation since the 1970s (Doan, Halevy, and Ives 2012); however, we agree with the authors of (Breiner, Subrahmanian, and Jones 2018) and others that these techniques are insufficient for the practice of AI and modern IT systems integration and we describe a modern mathematical approach based on *category theory* (Barr and Wells 1990; Awodey 2010), and the categorical query language CQL[2], that is sufficient for today's needs and also subsumes and unifies previous approaches.

1.1 Outline

To help motivate our approach, we next briefly summarize an application of CQL to a data science project undertaken jointly with the Chemical Engineering department of Stanford University (Brown, Spivak, and Wisnesky 2019). Then, in Section 2 we review data integrity and in Section 3 we review category theory. Finally, we describe the mathematics of our approach in Section 4, and conclude in Section 5. We present no new results, instead citing a line of work summarized in (Schultz, Spivak, and Wisnesky 2017).

1.2 Motivating Case Study

In scientific practice, computer simulation is now a third primary tool, alongside theory and experiment. Within quantum materials engineering, *density functional theory (DFT)* calculations are invaluable for determining how electrons behave in materials, but scientists typically do no share these calculations because they cannot guarantee that others will interpret them correctly, mitigating much of the value of simulation to begin with; for example, ease of replication. Although there are many standardized formats for representing chemical structures, there are no such standards for more complicated entities such as the symmetry analysis of a chemical structure, the pseudo-potentials used in DFT calculation, density of states data resulting from a DFT calculation, and the DFT calculation itself. Furthermore, many questions of interest depend not on a single calculation, but rather on ensembles of calculations, grouped in particular

[1] the so-called "data lake" approach to data management, which can lead to data scientists spending 80% of their time cleaning data (Wickham 2014)

[2] http://categoricaldata.net

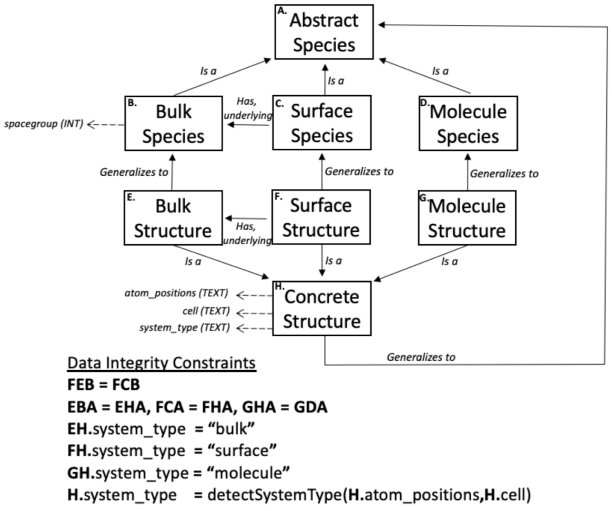


Figure 1: A Quantum Chemistry Schema

ways; for example, one is often interested in formation energies, i.e. a structure's energy relative to some reference energy, which depends on some arbitrarily-chosen mapping of its constituent elements to reference species, as well as the calculations for those reference species.

The above descriptions represent a tiny fraction of the complexity of the systems computational scientists grapple with. In practice, scientists can only communicate structured raw data in tiny fragments (e.g. specific chemical structures) of the systems they try to model, which contain concepts at higher levels of abstraction such as chemical species, reaction mechanisms, and reaction networks, and the ability to freely exchange structured data at the level of abstraction which scientists actually work supports many scientific activities such as machine learning applications which thrive on large databases.

The basic idea of the CQL solution to the above problem, which applies in many domains besides chemistry, is to design (or otherwise construct) a schema C with a rich set of data integrity constraints that capture the properties that should be preserved upon transformation, and then to verify that any schema mappings out of C, for example by other scientists, respect these constraints, a process expedited by the CQL automated theorem prover, a technology we will describe later. In this way, data cannot be exported onto schemas that do not respect the intentions of the original author. An example schema for DFT calculations (Brown, Spivak, and Wisnesky 2019) is shown in Figure 1.

2 Data Integrity

By *data integrity*, we mean the conformance of a database to a collection of *data integrity constraints* expressed in some formal language, such as first-order logic. When working with structured data, constraints are often built in to database *schemas*; for example, in SQL, a primary key constraint (stating for example that if two people have the same social security numbers, they must be the same person) can be given when defining the columns of a table. Other example constraints include range constraints (that for example an age be > 21) and join decompositions (that for example state that a table is the join of two other tables; such denormalized data sets are common in data science, for performance reasons). Another common constraint language is RDF/OWL (Doan, Halevy, and Ives 2012).

Data integrity is one of the mathematically quantifiable components of *data quality*, an informal and relative notion which roughly means that a database is "fit" for a particular purpose, such as supporting a particular forecasting model or building a particular data warehouse. For this reason, computer scientists have long urged practitioners to, usually, use data integrity constraints to formalize as much of the concept of data quality as is possible in a given scenario, to provide both conceptual clarity and possibly be used in implementation. However, unrestricted use of constraints quickly leads to undecidable problems (Doan, Halevy, and Ives 2012), as there is an innate trade-off between how expressive constraint languages are and how computationally difficult it is to reason about them. Hence, mainstream data transformation tools ("ETL" tools, such as Informatica PowerCenter and IBM DataStage) either do not support expressive constraints, or only use them internally.

The failure of tooling described above is especially tragic because there is such a significant amount of theoretical work on applying constraints to data transformation, integration, cleaning, schema mapping, and more (Doan, Halevy, and Ives 2012). For example, it is possible to formally relate constraints to the random variables of statistical models and thereby demonstrate the absence of various statistical anomalies (Wickham 2014). It is doubly tragic because many practitioners, who are often not trained in computer science and many only be exposed to computer science through data transformation, may never be exposed to the idea that data quality can be formally and in many cases automatically enforced by using data integrity constraints.

In the enterprise, data quality degradation due to loss of integrity during transformation is a systemic problem resistant to current techniques, and as such we propose to tackle it by building on top of a particular constraint language – *category theory* – that has become a de-facto constraint language for the practice of mathematics itself, in the sense that mathematicians use it to axiomatically define abstract structures such as groups using exactly the same kinds of data integrity constraints used in data management (Wells 1994)[3].

3 Category Theory

Category theory (Barr and Wells 1990; Awodey 2010) is the most recent branch of pure mathematics, originating in 1946 in algebraic topology. There are three main concepts of study: *categories*, *functors*, and *natural transformations*.

A *category* $\mathcal{C}$ consists of a set, $\mathsf{Ob}(\mathcal{C})$, the elements of which we call *objects*; a set $\mathsf{Mor}(\mathcal{C})$, the elements of which

[3] A mathematician would undoubtedly never refer to category theory as a constraint language, however. Here we more precisely are referring to categorical logic, such as that of topoi.

we call *morphisms*; functions $\mathsf{dom}_{\mathcal{C}}, \mathsf{cod}_{\mathcal{C}} : \mathsf{Mor}(\mathcal{C}) \to \mathsf{Ob}(\mathcal{C})$ called *domain* and *co-domain*; a function $\mathsf{id}_{\mathcal{C}} : \mathsf{Ob}(\mathcal{C}) \to \mathsf{Mor}(\mathcal{C})$; and a partial function $\circ_{\mathcal{C}} : \mathsf{Mor}(\mathcal{C}) \times \mathsf{Mor}(\mathcal{C}) \to \mathsf{Mor}(\mathcal{C})$ called *composition* such that (where we omit the sub-scripted $\mathcal{C}$):

$$\mathsf{dom}(\mathsf{id}(o)) \;=\; \mathsf{cod}(\mathsf{id}(o)) \;=\; o$$

$$\mathsf{cod}(g) = \mathsf{dom}(f) \;\Rightarrow\; \mathsf{dom}(f \circ g) = \mathsf{dom}(g)$$

$$\mathsf{cod}(g) = \mathsf{dom}(f) \;\Rightarrow\; \mathsf{cod}(f \circ g) = \mathsf{cod}(f)$$

$$\mathsf{id}(\mathsf{dom}(f)) \circ f \;=\; f \;=\; \mathsf{id}(\mathsf{cod}(f)) \circ f$$

$$(\; \mathsf{cod}(g) = \mathsf{dom}(f) \wedge \mathsf{cod}(h) = \mathsf{dom}(g) \Rightarrow$$

$$(f \circ g) \circ h = f \circ (g \circ h)\;)$$

Note that the objects and morphisms of every category form a directed multi-graph. An example category is Set, the category whose objects are sets and whose morphisms are functions, with composition given by function application. Another example is Group, the category of groups and group homomorphisms. Programming languages often form categories, with objects as types and morphisms as programs.

A *functor* $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ from category $\mathcal{C}$ to category $\mathcal{D}$ consists of a function, (also written) $\mathcal{F} : \mathsf{Ob}(\mathcal{C}) \to \mathsf{Ob}(\mathcal{D})$; and a function (also written) $\mathcal{F} : \mathsf{Mor}(\mathcal{C}) \to \mathsf{Mor}(\mathcal{D})$ that preserves identities and composition:

$$\mathsf{dom}_{\mathcal{D}}(\mathcal{F}(f)) = \mathcal{F}(\mathsf{dom}_{\mathcal{C}}(f)) \qquad \mathcal{F}(\mathsf{id}_{\mathcal{C}}(o)) = \mathsf{id}_{\mathcal{D}}(\mathcal{F}(o))$$

$$\mathsf{cod}_{\mathcal{D}}(\mathcal{F}(f)) = \mathcal{F}(\mathsf{cod}_{\mathcal{C}}(f)) \quad \mathcal{F}(f \circ_{\mathcal{C}} g) = \mathcal{F}(f) \circ_{\mathcal{D}} \mathcal{G}(g)$$

An example functor is the free group functor $\mathsf{free} : \mathsf{Set} \to \mathsf{Group}$ that takes each set to the free group generated by it, and each function to the associated unique group homomorphism. Another example functor is the forgetful group functor $\mathsf{forget} : \mathsf{Group} \to \mathsf{Set}$ that takes each group to its underlying carrier set. These two functors are not inverses, but are so called *adjoints*, a kind of generalization of the notion of inverse from which category theory derives much of its utility, but which we do not elaborate on here.

Finally, a *natural transformation* $\tau : \mathcal{F} \to \mathcal{G}$ between functors $\mathcal{F}, \mathcal{G} : \mathcal{C} \to \mathcal{D}$ consists of, for every object $c \in \mathsf{Ob}(\mathcal{C})$ a morphism $\tau_c \in \mathsf{Mor}(\mathcal{D})$ such that

$$\mathsf{dom}_{\mathcal{D}}(\tau_c) = \mathcal{F}(c) \qquad \mathsf{cod}_{\mathcal{D}}(\tau_c) = \mathcal{G}(c)$$

$$\tau_{\mathsf{cod}_{\mathcal{C}}(f)} \circ_{\mathcal{D}} F(f) = \mathcal{G}(f) \circ_{\mathcal{D}} \tau_{\mathsf{dom}(f)}$$

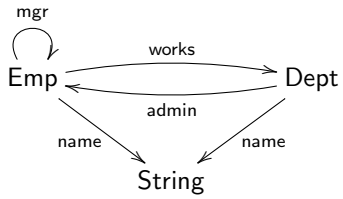
For every category $\mathcal{C}$, there is a category $\mathsf{Set}^{\mathcal{C}}$ whose objects are functors $\mathcal{C} \to \mathsf{Set}$ and whose morphisms are natural transformations. $\mathsf{Set}^{\mathcal{C}}$ is a mathematical structure called a *topos* which can interpret first-order logic and set-theory, a fact (Wells 1994) that we will make use of in the next section. Another fact we will use in the next section is that, ignoring minor issues relating to self-reference, there is a category whose objects are categories and whose morphisms are functors (and so in particular, functors compose, as do natural transformations).

4 Functorial Data Migration

In this section we present our approach to formally and automatically ensuring that data integrity is preserved during transformation, comparing to previous approaches as we go along. Our key idea (Schultz, Spivak, and Wisnesky 2017) is that database schemas are categories, and from that idea, an entire mathematical and algorithmic theory and practice of data transformation emerges, subsuming most current approaches such as SQL by virtue of the fact that category theory a kind of meta-theory for mathematics.

4.1 Algebraic Databases

To use our approach, we start by defining a directed multigraph to represent each database schema we are interested in. The nodes of this graph are names for either types or database tables and the edges of this graph are names for either attributes or foreign keys, . For example, if we are interested in a database about employees and departments, we may begin with the graph:



For data integrity constraints, we use equational logic. Continuing with our example, we might have:

$$\mathsf{works}(\mathsf{admin}(d)) = d$$

which express that every administrator works in the department he or she administers. The graph and equations together determine a category whose objects are nodes in the graph and whose morphisms with domain n_1 and co-domain n_2 are (possibly empty) paths $n_1 \to n_2$ in the graph, where we identify paths that are equivalent under the equations. In this example, the induced category has infinitely many morphisms, because there are many paths through Emp via mgr.

Having observed that database schemas are categories, we next observe that a database on schema $\mathcal{C}$ is a functor $\mathcal{C} \to \mathsf{Set}$. Such a functor may presented as a set of tables, where we omit the infinite String table that contains all strings:

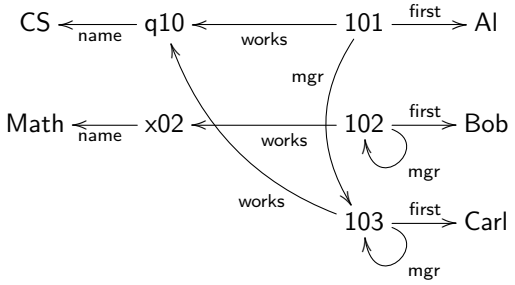
Emp	mgr	works	name
101	103	q10	Al
102	102	x02	Bob
103	103	q10	Carl

Dept	admin	name
q10	102	CS
x02	101	Math

It is easy to see that these tables satisfy the data integrity constraints. Indeed, when $\mathcal{C}$ is presented by generating morphisms and equations, every functor $\mathcal{C} \to \mathsf{Set}$ will satisfy the equations of $\mathcal{C}$, a useful property of our formalism. We will not illustrate natural transformations between two databases, but they correspond precisely to their relational counterparts (Doan, Halevy, and Ives 2012).

It is important to note that although we are using tables in the example above, our approach is not limited to relational

data; for example, we may just as easily use a representation of databases as graphs, such as found in a triple store:



4.2 Data Transformation

Given a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$, we can convert a database $\mathcal{J}$ on schema $\mathcal{D}$ to a database $\Delta_{\mathcal{F}}(\mathcal{J})$ on schema $\mathcal{C}$ by composition pre-composition with $\mathcal{F}$:

$$\mathcal{C} \xrightarrow{\mathcal{F}} \mathcal{D} \xrightarrow{\mathcal{J}} \mathsf{Set}$$
$$\Delta_{\mathcal{F}}(\mathcal{J}) : \mathcal{C} \to \mathsf{Set} := \mathcal{J} \circ \mathcal{F}$$

The operation (in fact, functor) Δ_F that takes databases on schema $\mathcal{D}$ and converts them to schema $\mathcal{C}$ does not (in general) admit an inverse, but it does (always) admit two inverse-like operations, Σ_F and Π_F, called the left and right *adjoints* of Δ_F, respectively, that convert databases on schema $\mathcal{C}$ to schema $\mathcal{D}$; i.e., they migrate data in the opposite direction as Δ_F. The operations $\Delta_F, \Sigma_F, \Pi_F$ form a core part of our approach are are used many times over in a typical CQL program.

In our approach it is natural to call a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ a "schema mapping", because functors are called maps between categories. However, in relational database theory, there are not three data migration operations corresponding to one schema mapping, there is only one, with a semantics (called "chase semantics" (Doan, Halevy, and Ives 2012)) broadly similar to our Σ_F. Our approach thus generalizes relational schema mappings (Doan, Halevy, and Ives 2012). Additionally, pairs of schema mappings with a common co-domain generalize SQL's SELECT-FROM-WHERE queries, but without issues relating to NULLs and without the need to join columns that are connected by foreign keys. Finally, our approach generalizes (Schultz, Spivak, and Wisnesky 2017) to encompass all of the classes of data integrity constraints studied in database theory, such as existential horn-clause logic (Doan, Halevy, and Ives 2012). In short, there is little to be lost in moving to a category-theoretic ("functorial") approach to data migration from the traditional, relational approach.

In fact, there is much to be gained as well: because functors preserve equations, as discussed earlier, data migrations in our approach also always preserve data integrity constraints, meaning that queries "cannot go wrong". The price to be paid for this compile-time assurance is that checking if a functor $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ is actually functorial and not just an assignment of objects to objects and arrows to arrows is an undecidable problem. The reason is that, when $\mathcal{C}$ and $\mathcal{D}$ are presented by generating morphisms and equations, we must check, for each equation $p = q$ in $\mathcal{C}$, that $\mathcal{F}(p) = \mathcal{F}(q)$ in $\mathcal{D}$, which is not decidable in general (Baader and Nipkow 1998). Hence, for our approach to be feasible in practice, automated theorem proving techniques (Baader and Nipkow 1998) must be used, and their development is one of the key contributions of the open-source CQL query language, which implements the Δ, Σ, Π data migrations, and many other constructions, in software.

5 Conclusion

In this paper we have presented a vision for a formally verified data transformation infrastructure, based on the mathematics of category theory and the categorical query language CQL, and described a motivating case study from data science. Our approach to data migration and integration extends existing approaches from relational database theory, while providing an extended semantics that has already proved indispensable in many projects, including (Wisnesky et al. 2017),(Nolan et al. 2019), and (Brown, Spivak, and Wisnesky 2019). Given that data quality is the primary obstacle in unlocking the power of AI, by transitivity we expect our approach to data transformation to be fundamental to unlocking the power of AI.

References

Awodey, S. 2010. *Category Theory*. New York, NY, USA: Oxford University Press, Inc., 2nd edition.

Baader, F., and Nipkow, T. 1998. *Term Rewriting and All That*. New York, NY, USA: Cambridge University Press.

Barr, M., and Wells, C. 1990. *Category Theory for Computing Science*. Upper Saddle River, NJ, USA: Prentice-Hall.

Breiner, S.; Subrahmanian, E.; and Jones, A. 2018. Categorical foundations for system engineering. In *Disciplinary Convergence in Systems Engineering Research*. Springer.

Brown, K. S.; Spivak, D. I.; and Wisnesky, R. 2019. Categorical data integration for computational science. *Computational Materials Science* 164:127 – 132.

Doan, A.; Halevy, A.; and Ives, Z. 2012. *Principles of Data Integration*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st edition.

Nolan, J. S.; Pollard, B. S.; Breiner, S.; Anand, D.; and Subrahmanian, E. 2019. Compositional Models for Power Systems. *Compositionality Journal* 1(1).

Schultz, P.; Spivak, D. I.; and Wisnesky, R. 2017. Algebraic model management: A survey. In *Recent Trends in Algebraic Development Techniques*. Springer.

Wells, C. 1994. Sketches: Outline with references. In *Dept. of Computer Science, Katholieke Universiteit Leuven*.

Wickham, H. 2014. Tidy data. *Journal of Statistical Software, Articles* 59(10):1–23.

Wisnesky, R.; Breiner, S.; Jones, A.; Spivak, D. I.; and Subrahmanian, E. 2017. Using Category Theory to Facilitate Multiple Manufacturing Service Database Integration. *Journal of Computing and Information Science in Engineering* 17(2).