Data Science and the Role of Theory: A Review of *Big Data Integration Theory* (2014) by Zoran Majkić and *Category Theory for the Sciences* (2014) by David Spivak

Rich Haney, 2 August 2015

As defined by Cleveland[1], while most of data science concerns work in modeling and statistical science, it also makes use of a variety of approaches to computing with data. As such the concept draws upon concepts of computing discussed by Chambers[2], and it greatly extends the concept of "greater statistics" of that author.

To be sure, work in data science admits to many perspectives; one, as author Joel Grus explains, is that:

> According to a Venn diagram that is somewhat famous in the industry, data science lies at the intersection of hacking skills, math and statistics knowledge, and substantive expertise"[3]..

But another perspective involves the role of theory. As explained with the help of some text from Cleveland[1]:

> Theory, both mathematical and non-mathematical theory, is vital to data science.... Tools of data science -- models and methods together with computational methods and computing systems -- link data and theory... New data create the need for new tools. New tools need new theory to guide their development. Mathematical theory is a means of investigation and can shed light on all areas of data science including new theories.... [A] successful implementation of the plan will bring new, exciting areas of research and development to data science (pp 8-9 of [1]).

In this review we suggest that while the *science* part of data science is mostly statistical science, it also includes the formal science of data, processes, and structural aspects of software systems overall. For this there is at least a modest role – and perhaps there is much more than a modest role – for universal algebra and category theory. In this context, the two books at hand make an even stronger case for the relevance of those two fields. One, Spivak's *Category Theory for the Sciences*, is a landmark work that -- in addition to giving some new results of relevance to data science -- makes the field of category theory itself much more accessible to those without backgrounds in advanced mathematics. The second work, *Big Data Integration Theory* by Zoran Majkić gives additional results of potential long-term significance for big data applications.

The roots of both books, beginning with ideas about structural models for data (and data constraints) defined using equations, date back to work by the mathematician Emmy Noether in the 1920s. Noether mentored the development of several different fields, to include universal algebra. That is the field a field used over time evolved to define the main algebraic perspectives on databases[4]. But universal algebra historically lacked a "constructive" perspective. Since the 1940s, that perspective has been provided by category theory. Although category theory in some sense is even abstract approach than universal algebra, it has found increasing relevance to logic, algebraic specification of software, and programming languages such as Scala and Haskell. David Spivak's remarkable volume communicates the core abstractions in a way that is the clearest to date[5].

Category Theory for the Sciences by David Spivak

Databases in the broad sense include relational databases with ACID capabilities, systems of SAS or R datasets, graph databases, document databases, and databases of other kinds. Graph databases, for example, include those defined by triple-store or Resource Description Framework (RDF) data pulled from the web. But how do you convert data from relational database tables to RDF form? As carefully explained by Spivak on page 363 of his work, the conversion from one to the other represents one of core constructions of category theory.

Work in ETL and data warehouses involves use of an extensive vocabulary that is often of heuristic nature. For example, in ETL, transformations can be "active", "passive", or "both active and passive" at the same time; in data warehouses one can speak of facts that are "fact-less" facts, and data warehouse tables can be Type I, II, II, VI and so on. While this vocabulary has heuristic value, one would at least like the option of a vocabulary for databases that is formally defined. But this is what we get in the subject at hand. In the following are some examples of abstract terms in category theory that can be used for databases as well.

Database terms	**Abstract/formal terms**
Fact-less facts	Boolean variables; when they are appropriate, Skolem variables for nulls
Joins in SQL	Fibered products
Cartesian products in SQL	Products
"where exists" clauses in SQL	Dependent sums and dependent products
"Type II" History tables	Sums (as disjoint unions)
"Type IV", "Type VI", and other history tables as those tables include fields defining the state transitions.	Finite state machines (FSMs) for which one option is to represent them using polynomial expressions
Additional history tables that include fields for the state transitions	Labeled Transition Systems (LTSs) of deterministic or probabilistic nature, f\or use in in-memory tables used for streaming analytics.
Table columns having data in comma separated list form, as is common in ELT problems	Monoids
Tables representing blended ontologies	Fibered sums
Duplicate tables	Produced by copy ("clone" or "diagonal") functors; they give rise to identity types in homotopy theory[6]
Two tables with data that are duplicates or largely the same, put in one table with duplicates seen as in common.	Fibered sums
Rules and transformations in INPUT => OUTPUT form	As written in $\text{OUTPUTS}^{\text{INPUTS}}$ form, can then be termed "exponential" objects. Give rise to function types. So, tables of rules and transformations are defined in terms of function types.
Tables with tree structures	Trees defined in terms of polynomials [specifically as polynomial functors. polynomial endofunctors. and polynomial monads. closely tied to the theory of operads.]
Tables representing different kinds of graphs	Graphs from category theory perspective
Indexing that can be hard/soft or global/local	Fibrations (in the way fibrations have been applied to logic [7]). These come with a built-in translation and substitution framework.
Data quality constraints for accuracy, completeness, consistency, conformance to type, etc.	Constraints in algebraic form (as equations) as algebraic varieties, Here, instead of having constraints that are individually programmed, or defined using an inheritance models, constraints are defined in algebraic terms. [See also discussions of tuple generating dependencies in Majkić]
Data quality "dimensions", usually as 6-8 stand-alone concepts	A system of algebraic constraints having several levels, defined all together as a lattice

*Please see below for the ETL and general case for computational pipelines as schema mappings/data mappings are involved together

Dovetailing with the formal models of data and databases are models of states and computations on that data. Three of those of general nature are as follows.

Computing problem	**Abstract/formal terms**

ETL processes	Data migration functors as discussed by Spivak. (Please observe the importance of “reverse migration” functor.)
Computational pipelines – many aspects	Monads in general
	Polynomial monads in particular. (These are closely related to operads).
Data types versus state-based systems	Data types are algebras; state-based systems are coalgebras.[8]

The above vocabulary begins to define what one needs as an overall “one stop shopping” vocabulary for data, processes, and the overall structural aspects of software systems. It permits formal representation of tables with duplicate rows, comma-separated lists of elements in fields, and other structures that occur early on in data pipelines, for example. As such, it allows formal definition of Extract, Load Transform (ELT) processes that is used in practice by many companies as an alternative to ETL. The vocabulary allows the many different kinds of objects involved – raw data, data constraints, rules, and transformations to all be defined in the same tier. For some architectures, having all of those objects in the same tier can make for a considerable simplification; that is particularly the case for newer architectures in which all data can be in memory.

In the above context, Spivak gives introductions of all of the basic constructions of category theory in a way that is the most lucid in print; he also gives his own results of value to ETL and related processes.

Formal languages for use in automated programming

Architects and developers who make use of automated programming methods will want to test out an approach that is more formal than the approaches of Entity Relationship (ER) and Unified Modeling Language (UML) will find that in the Functorial Programming and Query Language (FPQL) of Spivak and Wisnesky to be of great interest. Those such as myself who have long used algebraic methods of software specification under the hood in their systems (e.g., Z, Common Algebraic Specification Language -- CASL, and CoCASL) will find FPQL extremely promising and may even consider switching over to the use of FPQL in the near-term. Over the long run, one can only hope that FPQL will continue to evolve and include other capabilities from the world of algebraic specifications as well.

Big Data Integration Theory by Zoran Majkić

Among many aspects of this impressive work, three contributions in particular stand out:

1. Extensive use of typed operads (which Spivak introduces).
2. Incorporation of higher-order logics that in fact do seem needed for database mapping problems of complex nature.
3. Incorporation of a formal process algebra that can be used to define models of reactive systems and streaming data.

In general, data types and systems define algebras and processes (particularly state-driven, reactive processes) define coalgebras. The formal, algebraic perspective to handling the kinds of streaming data used in streaming analytics involves the latter. Majkić gives the most comprehensive discussion and integrated discussion of the two views (the structural, or algebraic view, and the process, or coalgebraic view) that has been developed.

The starting point involves a model of abstract data types (ADTs) as defined for databases in the 1980s that is coupled with a model of abstract, behavioral types, or Abstract Object Types (AOTs). But that is just the starting point. Over time, if it is possible to make the practical implementations, one would have one formalism that applies to data models and data quality constraints and that works as an overall model of states and computations. For example, automated programming tools used for big data may benefit will benefit from at least some aspects of the theory explored in this text.

The above discussion of work by Spivak made mention of several possible, formal upgrades to ER and UML diagrams, namely FPGL of Spivak and Wisnesky and various approaches to algebraic specification. But it is possible that a more extensive set of upgrades may be needed (and be beneficial).   In this context, Majkić makes use of the sketch data model of category theory that does seem to be needed to explore systems of database mappings of more complex kinds[9].

One is tempted at times to think that perspectives involving different ideas of types by themselves are enough to define formal systems, in which case category theory would not be necessary.   But the far more compelling case, as shown by the two books at hand (and other work as well[6,7,10]) is that what will be there most fruit will be a balanced approach making use of category theory and type theory together.

Conclusion

Computing with data includes many engineering (and hacking) aspects as well[11].   But as a science in its details is primarily about statistical science [12], but it also involves the theory of computation, data, and structural aspects of software systems overall.   For such work, category theory has at least a modest role – and it perhaps has much more than a modest role..   As the problems to solve become harder, approaches as discussed in these two fine works will be more and more significant for data science and big data applications.

Notes

[1] See Cleveland, "Data Science: An Action Plan for Expanding the Technical Areas of the Field of Statistics", http://www.stat.purdue.edu/~wsc/papers/datascience.pdf, 2001.

[2] See John M. Chambers, "Computing with Data: Concepts and Challenges" in *The American Statistician*, 53:1 (1999), pp. 73-84. *(Web version is extended from the journal version.)* See also Tukey, J.W. (1963), "The Inevitable Collision between Computation and Data Analysis" in Proceedings of the IBM Scientific Computing Symposium on Statistics, 141-152, White Plans, NY: IBM

[3] See Joel Grus(2015), "*Data Science From Scratch: First Principles with Python*" (2015).

[4] For the principal algebraic perspectives on databases, please see *Foundations of Databases: The Logical Level* by Serge Abiteboul, Richard Hull, Victor Vianu (1994) and B. Plotkin(1994), *Universal Algebra, Logic, and Databases*.

[5.] For an introduction even more basic than Spivak that includes very good examples for electrical circuits and also stacks, is at and RFC Walters(1991), *Categories and Computer Science*. Roughly at the same level as Spivak is *Conceptual Mathematics: A first Introduction to Categories*" by F. William Lawvere and Stephen H. Shanuel. To see some uses of operads in general developers will benefit from reading some additional examples such as https://johncarlosbaez.wordpress.com/2011/07/06/operads-and-the-tree-of-life/ After reading the above, *An Introduction to Category Theory* by Harold Simmons(2011) would make sense. That volume gives the best introduction to adjoins that is available. The best textbook in the field is now *Category Theory* by Steve Awodey(2012).

[6]. See *Homotopy Type Theory: Univalent Foundations of Mathematics*. The Univalent Foundations Program, Institute for Advanced Study (2013) at https://hottheory.files.wordpress.com/2013/03/hott-online-611-ga1a258c.pdf.

[7.] See Bart Jacobs(1999), *Categorical Logic and Type* Theory (1999), Studies in Logic and the Foundations of Mathematics, Vol 141, Elsevier, originally 1990 PhD dissertation. The reference text in the field.

[8.] Bart Jacobs(2012), *Introduction to Coalgebra. Towards Mathematics of States and Computations*, retrieved at http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf.

[9] See, for example, "Sketch data models, relational schema, and data specifications, Michael Johnson and Robert Rosebrugh, *Electronic Notes in Theoretical Computer Science*(2001) and other work on the functorial data model by Rosebrugh. Also see, for example, Brice Halimi(2012), "Diagrams as Sketches", *Synthese* (2012): 186: 387-409.

[10] See, for example, Paul-André Melliès CNRS, Université Paris Diderot Sorbonne Paris Cité (2015), "Functors are Type Refinement Systems", at http://noamz.org/papers/funts.pdf.

[11] We would see views about engineering vs. science of Henry Petrovsky, author of *The Pencil: A History of Design and Circumstance* (1990), for example) as applying here as well. As it involves computing with data, although the science has provided the main basis, the engineering has more often led the way; at least some aspects of the science may here be in a bit of "catch-up" mode, at least in terms of how the science can actually be applied.

[12]. It is true that a weak part of the argument involves the fact that (though I have found it useful in other areas) when working as a statistician I simply have not find the main resource for category theory useful for doing statistics itself.   The main resource is, Peter McCullaugh(2001), “What is a statistical model?” , *The Annals of Statistics* 2002, Vol. 30, No. 5, 1225–1310.   Perhaps someone of the level of Spivak may begin to make things clear (perhaps starting with both tree and additive models in statistics as both being both defined as polynomial functors and polynomial monads, with the latter being closely related to operads).