Module Theory and Query Processing (Extended Abstract)

Fritz Henglein
University of Copenhagen
fritz@henglein.com

Mikkel Kragh Mathiesen
University of Copenhagen
mkm@di.ku.dk

Summary. We develop the basics of module theory with an emphasis on categorical constructions. Elements of free modules have a natural interpretation as finite polysets, generalized multisets where elements can also occur a negative or even fractional number of times. Many queries traditionally expressed in SQL correspond to multilinear maps on polysets. By employing symbolic constructors for scalar multiplication and tensor products we obtain more compact representations that facilitate run-time optimizations using algebraic identities and yield asymptotic speed-ups compared to the conventional stream-of-tuples representation in database systems. In particular we show that a straightforward functional join algorithm is efficient even for cyclic joins.

From sets and multisets to polysets. Traditional query processing is based on relational algebra and SQL. In relational algebra query inputs and results are sets (elements occur 0 or 1 times) whereas SQL uses bag semantics (elements occur zero or more times). Extensions to this model have been proposed, such as probabilistic databases [2] (elements have real number multiplicities between 0 and 1) or provenance semirings [3] (elements have multiplicities from any semiring). In this manner databases are gradually progressing from set-like semantics to algebraic semantics. We propose to continue in this direction and go to using modules over rings to harvest increased expressiveness (support for subtraction/deletion), compact data structures and computational efficiency *simultaneously* facilitated by (symbolic) linear algebra [5].[1] The result is a theory of *polysets*: occurrences have multiplicities from any ring with linear maps (homomorphisms), which turn out to be sufficient for expressing a wide variety of queries. If and only if necessary, computationally expensive non-linear operations can be inserted in select places, e.g. for mapping a polyset to a multiset by eliminating negative multiplicities.

Modules. The general structure is that of an R-module, a set M equipped with operations

$$\begin{aligned} + &: M \to M \to M \\ \cdot &: R \to M \to M \end{aligned}$$

satisfying the usual axioms of identity, commutativity, associativity and distributivity where R is a ring with unity.

Compared to the usual set and multiset semantics this already gives us two new tools. Firstly, for any element x we have its negation $-x = (-1) \cdot x$. This operation takes the place of set difference in relational algebra but has the additional advantage that updates are commutative, e.g. $x + (-y)$ is the same as $(-y) + x$.

[1]Most of this development is described in the second author's thesis.

Submitted to:
MSFP 2020

Negative multiplicities of elements is not always desirable so we may wish to normalize the data at specific points. If data is normalized after every update we recover the usual set- or multiset-based semantics, whichever is desired. In this fashion query processing is not intrinsically about finite sets (relational algebra) or multisets (SQL).

The other tool is a notation for repeated elements, so we can write e.g. $4 \cdot x$ instead of $x+x+x+x$. Directly using this notation as a representation we can obtain asymptotic improvements in some cases, especially in combination with tensor products as explained later.

Universal properties. To further explore the possibilities of exploiting algebraic properties we look at module theory through the lens of category theory using universal properties. Consider the homogeneous coproduct $\coprod_X M$, i.e. X copies of M. This can be seen as a mapping with finite support from the set X into the elements of M. In particular, $\coprod_X R$ which is the free R-module generated by X. If we have attributes $A_1, \dots, A_n$ then the free $\mathbb{Z}$-module over $A_1 \times \cdots \times A_n$ serves as the domain for tables containing these attributes. By the universal property of coproducts we get natural isomorphisms such as

$$\begin{aligned} \coprod_{X+Y} M &\cong \coprod_X M \oplus \coprod_Y M \\ \coprod_{X \times Y} M &\cong \coprod_X \coprod_Y M \end{aligned}$$

In this way we precisely recover the structure of generic tries [4].

Next, we turn to tensor products which are described by their universal property relative to bilinear maps. Specifically, bilinear maps $(M_1, M_2) \to M_3$ are equivalent to linear maps $M_1 \otimes M_2 \to M_3$. A symbolic tensor product constructor gives rise to a compact representation of join results since we can represent elements of the tensor space as $x \otimes y$ for arbitrary *tables* x and y. This avoids unnecessarily multiplying it out to the standard stream-of-tuples form used in database systems, which has a worst-case quadratic blowup. Hence, we recover the advantages of factorized databases [1]. Additionally, a symbolic scalar multiplication constructor provides a potentially exponentially more compact representation of tables, storing $k \cdot x$ instead of $x + \ldots + x$. This is for example useful in representing the output of projections or SQL semi-joins.

In general, by taking an algebraic perspective we can draw on a vast repository of knowledge, exposing optimizations that are otherwise nonobvious, impractical or difficult to get right.

Joins. A major challenge of evaluating database queries is handling joins. Using the trie structures derived from coproducts and symbolic tensor products we can compute simple joins in worst-case linear time. Applying this technique to more complicated situations we get efficient joins even in more complicated situations. In particular, triangle joins, which belong to the notoriously hard class of cyclic joins, are computed in $O(n^{\frac{3}{2}})$. This is worst-case time optimal [6] and known not to be achievable by classical relational database systems [7].

References

[1] Nurzhan Bakibayev, Dan Olteanu & Jakub Závodný (2012): *FDB: A Query Engine for Factorised Relational Databases*. *Proc. VLDB Endow.* 5(11), pp. 1232–1243, doi:10.14778/2350229.2350242. Available at `https://doi.org/10.14778/2350229.2350242`.

[2] Nilesh Dalvi & Dan Suciu (2004): *Efficient Query Evaluation on Probabilistic Databases*. In: *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, VLDB Endowment, pp. 864–875.

[3] Todd J. Green, Grigoris Karvounarakis & Val Tannen (2007): *Provenance Semirings*. In: *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '07, Association for Computing Machinery, New York, NY, USA, pp. 31–40, doi:10.1145/1265530.1265535. Available at `https://doi.org/10.1145/1265530.1265535`.

[4] Ralf Hinze (2000): *Generalizing generalized tries*. *Journal of Functional Programming* 10(4), pp. 327–351, doi:10.1017/S0956796800003713.

[5] Mikkel Kragh Mathiesen (2016): *Infinite-Dimensional Linear Algebra for Query Processing*. Master's thesis, University of Copenhagen.

[6] H.Q. Ngo, E. Porat, C. Ré & A. Rudra (2012): *Worst-case optimal join algorithms*. In: *Proceedings of the 31st symposium on Principles of Database Systems*, ACM, pp. 37–48.

[7] Hung Q Ngo, Christopher Re & Atri Rudra (2013): *Skew Strikes Back: New Developments in the Theory of Join Algorithms*. *arXiv preprint arXiv:1310.3314*.