DOI 10.1007/s00165-014-0316-9
BCS © 2014
Formal Aspects of Computing (2015) 27: 283–307

Formal Aspects of Computing

A linear algebra approach to OLAP

Hugo Daniel Macedo[1] and José Nuno Oliveira[2]

[1] INRIA, Centre Paris-Rocquencourt, 23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France
[2] High Assurance Software Lab/INESC TEC and University of Minho, Braga, Portugal

Abstract. Inspired by the relational algebra of data processing, this paper addresses the foundations of data analytical processing from a linear algebra perspective. The paper investigates, in particular, how aggregation operations such as cross tabulations and data cubes essential to quantitative analysis of data can be expressed solely in terms of matrix multiplication, transposition and the Khatri–Rao variant of the Kronecker product. The approach offers a basis for deriving an algebraic theory of data consolidation, handling the quantitative as well as qualitative sides of data science in a natural, elegant and typed way. It also shows potential for parallel analytical processing, as the parallelization theory of such matrix operations is well acknowledged.

Keywords: Software engineering, Formal methods, Data science

1. Introduction

In a recent article in the Harvard Business Review, Davenport and Patil [DP12] declare *data scientist* as the *sexiest job of the 21st century*. Such high-ranking professionals should be trained to *make discoveries in the world of big data*, this showing how much companies are *wrestling with information that comes in volumes never encountered before*. The job calls for a lot of creativity mixed with solid foundations in *maths, statistics, probability, and computer science*.

Leaving aside the enormous challenges posed by big *unstructured* data, a *data scientist* is expected to live on *data science*, whatever this is. Concerning structured data, we see data science as a two-fold body of knowledge, made of *qualitative* as well as *quantitative* ingredients. The qualitative side is provided by the solid theory of databases [Mai83] which, formalized in logic and (relational) set theory, has led to standard querying languages over relational data such as SQL. As for the quantitative side, we see similar efforts in the formalization of data analytic techniques—put forward under the umbrella of the OLAP [1] acronym—but such efforts seem less successful in setting up a thorough semantic basis for understanding and optimizing analytical processing.

Correspondence and offprint requests to: H. D. Macedo, E-mail: hugo.dos_santos_macedo@inria.fr

[1] OLAP stands for *On-line Analytical Processing* [DT99, PJ01].

It is true that formal definitions for concepts such as *multi-dimension database* [GL97], data *aggregation* and data *cube* [DT99] have been given (among others), including an *algebra* of cube operators [DT99]. Little is written, however, concerning algebraic properties of such operators. And those which are given either address the qualitative side again (the *dimension algebra* [JLN00] rather than the *measure* one) or are stated without proof (e.g. the two equalities in [GCB^{+}97] concerning roll-up, group-by and cube).

These shortcomings are easy to understand: while relation algebra "à la Codd" [Cod70] and naive set theory work well for qualitative data science (focus on attribute and dimension structures), they are rather clumsy in handling the quantitative side (focus on measure structures and their operations). In this paper we propose to solve this problem by suggesting *linear algebra* (LA) as an alternative suiting both sides: the qualitative one—by regarding it as a *typed* theory—and the quantitative one—by internalizing all details of data consolidation and aggregation under the operations of matrix composition (namely multiplication) and converse (transposition).

This approach builds upon previous work on typed linear algebra and its applications in computer science, which include areas as diverse as data vectorization [MO13], probabilistic program calculation [Oli12], weighted automata [Oli13], component-oriented design [MO11b, Oli14b] etc. Details and further examples can be found in a technical report [MO11a] which also elaborates on the potential of the approach for OLAP parallelization.

Contribution. The ideas presented in this paper derive from the authors' work on typing linear algebra [MO10, Mac12, MO13] which eventually drove them into the proposed synergy between linear algebra and OLAP. Such a synergy is, to the best of their knowledge, novel in the field. Rather than relying on standard OLAP state of the art developments, a cross-field perspective is put forward that may open new ways of looking at this body of knowledge.

Overview of the paper. The remainder of this paper is structured as follows. Sections 2 and 3 explain the shift from relational to linear algebra, imposed by the shift from qualitative to quantitative processing. Section 4 gives a brief overview of *typed linear algebra*. Section 5 expresses cross tabulations solely in terms of linear algebra matrix operations. Section 6 treats cross tabulation and "rolling up" along functional dependencies, introducing dimension hierarchies into the game. Section 7 proves that the construction of cross tabulations is incremental. Section 8 goes higher-dimensional into the LA construction of OLAP cubes. Finally, Sect. 9 reviews related work and Sect. 10 draws conclusions and gives a prospect of future work. Some technical details and proofs are deferred to the two appendices.

2. From relations to matrices

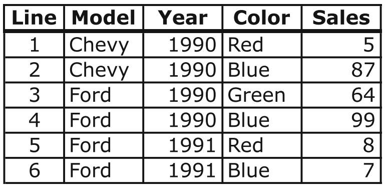
On-line analytical processing [DT99, PJ01, JPT10] aims at summarizing huge amounts of information in the form of histograms, sub-totals, cross tabulations (namely *pivot tables*), roll-up/drill-down transformations and data cubes, whereby new trends and relationships hidden in raw data can be found. The need for this technology concerns not only large companies generating huge amounts of data every day (the "big data" trend) but also the laptop spreadsheet user who wants to make sense of the data stored in a particular workbook.

Since Codd's pioneering work on the foundations of the *relational data model* [Cod70], relation algebra has been adopted as the standard basis for formalizing data processing. Given the proximity between relation and matrix algebra [Sch11, DGM14] the question arises: how much gain can one expect from translating results from one side to the other? This paper will show how a particular construction in relation algebra—that of a *binary relational projection*, defined in [Oli09, Oli11] to calculate with functional dependencies in databases—translates matrix-wise into *cross tabulations* (namely *pivot tables*) which are central to data analytical processing.

On the relational side, a binary relational projection is always of the form

$$\pi_{f,g} R = \{(f\ b, g\ a) \mid (b, a) \in R\}$$

where R is the binary relation being projected and f and g are *observation* functions, usually associated to attributes.



Line	Model	Year	Color	Sales
1	Chevy	1990	Red	5
2	Chevy	1990	Blue	87
3	Ford	1990	Green	64
4	Ford	1990	Blue	99
5	Ford	1991	Red	8
6	Ford	1991	Blue	7

Fig. 1. Collection of raw data (adapted from [GC97])

Although less common in the database literature, the alternative definition

$$\pi_{f,g} R = f \cdot R \cdot g^{\circ} \tag{1}$$

is simpler and easier to reason about, where the dot $(\cdot)$ between the symbols denotes *relational composition* and $(\_)^{\circ}$ expresses the converse operation: pair (b, a) belongs to relation R° iff pair (a, b) belongs to R.[2]

Projection pattern (1) turns up often in relation algebra [BdM97]. When expressing data dependencies, such projections take the form

$$f_A \cdot [\![T]\!] \cdot f_B^{\circ} \tag{2}$$

where T is a database file, or *table* (a set of data records, or *tuples*), A and B are attributes of the schema of T, f_A (resp. f_B) is the function which captures the semantics of attribute A (resp. B)[3] and $[\![T]\!]$ represents set T in the form of a *diagonal* relation:

$$[\![T]\!] \;=\; \{(t, t) \mid t \in T\}$$

This somewhat redundant construction proves essential to the reasoning, as shown in [Oli11, Oli14a]. Expressed in set-theoretical notation, projection (2) is set-comprehension $\{(t[A], t[B]) \mid t \in T\}$ where $t[A]$ (resp. $t[B]$) denotes the value of attribute A (resp. B) in tuple t.

Note how simple (2) is in its relying only on very basic combinators of relation algebra, namely composition and converse, which generalize to matrix multiplication and transposition, respectively. Under this generalization, we will show below that cross tabulations can be expressed by a formula similar to (2),

$$t_A \cdot [\![T]\!]_M \cdot t_B^{\circ} \tag{3}$$

where M is a *measure* attribute and attributes A and B are the *dimensions* chosen for each particular cross tabulation. Notation t_A (resp. t_B) expresses the *membership* matrix of the column addressed by dimension A (resp. B) whose construction will be explained later. Also explained later, $[\![T]\!]_M$ denotes the diagonal matrix capturing column M of T.[4]

The construction of matrices t_A, t_B and $[\![T]\!]_M$ will be first illustrated with examples. Cross tabulations will be pictured as displayed by Microsoft Excel.

3. Cross-tabulations

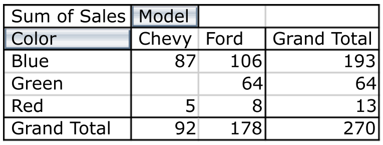
In data processing, a cross tabulation (or pivot table) provides a particular summary or view of data extracted from a raw data source. As example of raw data consider the table displayed in Fig. 1 where each row records the number of vehicles of a given model and color sold per year.

In general, the raw data out of which cross tabulations are calculated is not normalized and is collected into a central database, termed a data *warehouse* or decision support database. Different summaries answer different questions such as, for instance, *how many vehicles were sold per color and model?* For this particular question, the attributes *Color* and *Model* are selected as *dimensions* of interest, *Sales* is regarded as *measure* attribute and the corresponding cross tabulation is depicted in Fig. 2, as generated via the pivot table menu in Excel.

[2] Recall from discrete maths that, given two relations R and S, pair (c, a) will be in the composition $R \cdot S$ iff there is some b such that (c, b) is in R and (b, a) is in S. Thus, $(y, x) \in f \cdot R \cdot g^{\circ}$ in (1) means that $y = f\ b$ and $x = g\ a$ for some $(b, a) \in R$, that is, $(y, x) = (f\ b, g\ a)$. Altogether, $f \cdot R \cdot g^{\circ} = \bigcup_{(b,a) \in R} \{(f\ b, g\ a)\}$ which reduces to the given set comprehension.

[3] That is, given a tuple $t \in T$, $f_A(t)$ yields the value of attribute A in t, usually denoted by $t[A]$ (similarly for attribute B).

[4] The shift from the binary relations of (2) to the matrices in (3) will be detailed in the sequel. Although relations can be represented by Boolean matrices containing only 0s and 1s (more about this in Appendix A), matrix $[\![T]\!]_M$ will be a numeric matrix in general holding real-life quantities and measures.



Sum of Sales	Model		
Color	Chevy	Ford	Grand Total
Blue	87	106	193
Green		64	64
Red	5	8	13
Grand Total	92	178	270

Fig. 2. Pivot table as extracted by Excel from the data in Fig. 1

Large scale cross tabulation generation is an essential part of quantitative data analysis. As already mentioned, OLAP refers to the set of techniques performing such analysis over information stored in data warehouses, whose complexity is well-known [PKL02]. Quoting [DT99]: *The complexity of queries required to support OLAP applications makes it difficult to implement using standard relational database technology.* Feeling the lack of a *standard conceptual model for OLAP*, the same authors [DT99] propose one based on first order logic. Reference [VS99] provides a review of other efforts in defining logical models for OLAP.

Rather than trying to extend existing logic models towards accommodating OLAP semantics, the approach put forward in this paper changes strategy and calls for a synergy with the field of linear algebra. The key resides in expressing analytic operations in the form of matrix algebra expressions. In the particular case of reporting multi-dimensional analyses of data, one should be able to build three matrices as hinted by formula (3): two associated to the dimensions (attributes) A and B being analysed and a third one recording which *measure* or *metric* data are to be considered for consolidation.

This encoding of data into LA is quite smooth if matrix operations are *typed* in the way presented in e.g. [MO13]. For self-containedness we give a very brief overview of such *typed LA* notation below.

4. Typed linear algebra

Matrices as arrows. A matrix M with n rows and m columns is a function which tells the value $r\,M\,c$ which occupies the cell addressed by row r and column c, for $1 \leq r \leq n$, $1 \leq c \leq m$. Note that we prefer infix notation $r\,M\,c$ to e.g. M_{rc} or even $M(r, c)$ for reasons to be explained later.

Following the arrow notation of [MO13] and writing $n \xleftarrow{M} m$ to denote that matrix M is of *type* $n \leftarrow m$ (m columns, n rows), matrix *multiplication* can be expressed by arrow *composition*:

$$\begin{tikzcd} n & m \arrow[l, "M"'] & k \arrow[l, "N"'] \arrow[ll, bend left, "C=M\cdot N"] \end{tikzcd} \tag{4}$$

Point-wise, this operation is defined by:[5]

$$y\,(M \cdot N)x = \langle \Sigma\ z\ ::\ y\,M\,z \times z\,N\,x \rangle \tag{5}$$

For every n there is a matrix of type $n \longleftarrow n$ which is the unit of composition.

This is nothing but the identity matrix of size n, denoted by $n \xleftarrow{id_n} n$ or $n \xleftarrow{1} n$, indistinguishably. Therefore (diagram aside):

$$\begin{tikzcd} n \arrow[d, "M"'] & n \arrow[l, "id_n"'] \arrow[d, "M"] \arrow[dl, "M"] \\ m & m \arrow[l, "id_m"] \end{tikzcd}$$

$$id_m \cdot M \;=\; M \;=\; M \cdot id_n$$

Subscripts m and n can be omitted wherever the underlying diagrams are well-defined and can be inferred from the context.

[5] This and other pointwise definitions and rules to come are expressed in the style of the Eindhoven quantifier calculus, see e.g. [BM06]. Matrix *multiplication* is so-called because it can be regarded as an extension of numeric multiplication to matrices. Phrase *matrix composition* emphasises the underlying categorial basis [MO13] of this operation, which is less widely acknowledged. As types are central to the approach proposed in this paper, we will write *composition* instead of *multiplication* unless quoting work which explicitly uses the latter terminology.

Vectors as arrows. Vectors are special cases of matrices in which one of the dimensions is 1, for instance

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \quad \text{and} \quad w = \begin{bmatrix} w_1 \ \ldots \ w_n \end{bmatrix}$$

Column vector v is of type $m \leftarrow 1$ (m rows, one column) and row vector w is of type $1 \leftarrow n$ (one row, n columns). Our convention is that lowercase letters (e.g. v, w) denote vectors and uppercase letters (e.g. M, N) denote arbitrary matrices.

Converse of a matrix. One of the kernel operations of linear algebra is transposition, whereby a given matrix changes shape by turning its rows into columns and vice-versa. Given matrix $n \xleftarrow{M} m$, notation $m \xleftarrow{M^\circ} n$ denotes its transpose, or *converse*. The following laws hold: $(M^\circ)^\circ = M$ (idempotence) and $(M \cdot N)^\circ = N^\circ \cdot M^\circ$ (contravariance).

Block notation. Matrices can be built of other matrices using block notation. Two basic binary combinators are identified in [MO13] for building matrices out of other matrices, say M and N, regarded as blocks, either stacking these vertically, $\left[\frac{M}{N}\right]$, or horizontally, $\left[\, M | N \,\right]$. Dimensions should agree, as shown in the diagram aside, taken from [MO13], where m, n, p and t are types. Special matrices i_1, i_2, π_1 and π_2 are fragments of the identity matrix and play an important role in explaining the semantics of the two combinators. This, however, can be skipped for the purposes of the current paper [6], sufficing to know a number of laws which emerge from the underlying mathematics, namely *converse-duality*

$$\begin{tikzcd}
 & m & \\
n \arrow[ur, "M"] \arrow[r, shift right, "i_1"'] & n + p \arrow[u, "{[\, M | N \,]}"'] \arrow[l, shift right, "\pi_1"'] \arrow[r, shift left, "\pi_2"] & p \arrow[ul, "N"'] \arrow[l, shift left, "i_2"] \\
 & t \arrow[ul, "P"] \arrow[ur, "Q"'] \arrow[u, "{\left[\frac{P}{Q}\right]}"] &
\end{tikzcd}$$

$$\left[\, M | N \,\right]^\circ = \left[\frac{M^\circ}{N^\circ}\right] \tag{6}$$

divide-and-conquer

$$\left[\, M | N \,\right] \cdot \left[\frac{P}{Q}\right] = M \cdot P + N \cdot Q \tag{7}$$

which captures the essence of (parallelizable) matrix multiplication, two *fusion* laws

$$P \cdot \left[\, M | N \,\right] = \left[\, P \cdot M | P \cdot N \,\right] \tag{8}$$

$$\left[\frac{M}{N}\right] \cdot P = \left[\frac{M \cdot P}{N \cdot P}\right] \tag{9}$$

and the *abide law* [7]

$$\left[\frac{\left[\, M | N \,\right]}{\left[\, P | Q \,\right]}\right] = \left[\left[\frac{M}{P}\right] \middle| \left[\frac{N}{Q}\right]\right] = \left[\begin{array}{c|c} M & N \\ \hline P & Q \end{array}\right] \tag{10}$$

which establishes the equivalence between row-major and column-major construction of matrices by blocks. (Thus the four-block notation on the right.)

[6] The rich algebra of matrix block-operations arises essentially from the fact that vertical and horizontal block aggregation form a *biproduct*. The interested reader is referred to [MO13] for details.

[7] Neologism "abide" (= "above and beside") was introduced by Richard Bird [Bir89] as a generic name for algebraic laws in which two binary operators written in infix form change place between "above" and "beside", e.g.

$$\frac{a}{b} \times \frac{c}{d} = \frac{a \times c}{b \times d}$$

in fraction calculus.

Direct sum and Kronecker product. Given two matrices M and N, the *direct sum* of M and N is defined as follows, using block notation:

$$M \oplus N = \left[\begin{array}{c|c} M & 0 \\ \hline 0 & N \end{array}\right] \tag{11}$$

Mind the type $k + j \xleftarrow{M \oplus N} n + m$ for M and N of types $k \leftarrow n$ and $j \leftarrow m$, respectively. Direct sum is a standard linear algebra operator enjoying many useful properties [MO13]. The following equation, termed the *absorption law*, specifies how block operator $[\,|\,]$ absorbs direct sum $\oplus$, for suitably typed matrices M, N, P and Q:

$$\left[\, M | N \,\right] \cdot (P \oplus Q) = \left[\, M \cdot P | N \cdot Q \,\right] \tag{12}$$

Given the same two matrices $k \xleftarrow{M} n$ and $j \xleftarrow{N} m$, another standard construction in linear algebra is the so-called *Kronecker product* $k \times j \xleftarrow{M \otimes N} n \times m$. This operator can be defined by block-wise decomposition,

$$M \otimes N = \begin{bmatrix} x_{11}N & \cdots & x_{1n}N \\ \vdots & \ddots & \vdots \\ x_{k1}N & \cdots & x_{kn}N \end{bmatrix}$$

$$\begin{aligned} \left[\, M | N \,\right] \otimes P &= \left[\, M \otimes P | N \otimes P \,\right] \\ \left[\begin{array}{c} M \\ \hline N \end{array}\right] \otimes P &= \left[\begin{array}{c} M \otimes P \\ \hline N \otimes P \end{array}\right] \\ x \otimes N &= xN \end{aligned}$$

where x is a scalar (1-to-1 matrix) and xN denotes scalar multiplication. The picture above describes the outcome of the operation.

Khatri–Rao matrix product. Given matrices $n \xleftarrow{M} m$ and $p \xleftarrow{N} m$, the so-called Khatri–Rao [RR98] matrix product of M and N, denoted $n \times p \xleftarrow{M \triangledown N} m$ is a column-wise version of the Kronecker product operator given above,

$$\begin{aligned} u \triangledown v &= u \otimes v \\ \left[\, M_1 | M_2 \,\right] \triangledown \left[\, N_1 | N_2 \,\right] &= \left[\, M_1 \triangledown N_1 | M_2 \triangledown N_2 \,\right] \end{aligned} \tag{13}$$

where u, v are column-vectors and M_i, N_i are suitably typed matrices [8]. As an example of operation relying on this product consider row vector

$$s = \begin{bmatrix} 5 & 87 & 64 & 99 & 8 & 7 \end{bmatrix}$$

of type $1 \xleftarrow{s} 6$, capturing the transposition of the *Sales* column of Fig. 1. The Khatri–Rao product $s \triangledown id$ yields the corresponding diagonal matrix:

$$6 \xleftarrow{s \triangledown id} 6 = \begin{bmatrix} 5 & 0 & 0 & 0 & 0 & 0 \\ 0 & 87 & 0 & 0 & 0 & 0 \\ 0 & 0 & 64 & 0 & 0 & 0 \\ 0 & 0 & 0 & 99 & 0 & 0 \\ 0 & 0 & 0 & 0 & 8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix} \tag{14}$$

This conversion is essential to the LA encoding of cross tabulations, as shown in the sequel.

One can reduce over a matrix defined by rows on the right-hand side of a Khatri–Rao product whose left-hand side is a row vector:

$$v \triangledown \left[\begin{array}{c} M \\ \hline N \end{array}\right] = \left[\begin{array}{c} v \triangledown M \\ \hline v \triangledown N \end{array}\right]. \tag{15}$$

[8] As shown in [Mac12], this product generalizes to arbitrary matrices the tupling operator known as *split* in the functional setting [BdM97] or as *fork* in the relational one [Fri02, Sch11].

Should the shape of the matrix on the right hand side be a direct sum, the equation can be rewritten into:

$$\left[\, v | w \,\right] \triangledown (M \oplus N) = (v \triangledown M) \oplus (w \triangledown N) \tag{16}$$

This follows from (15) and (13).

Type generalization. Matrix types (the end points of arrows) can be generalized from traditional numeric dimensions to arbitrary denumerable types thanks to addition and multiplication of matrix elements being commutative and associative. This ensures unambiguous definition of matrix composition because the summation inside the inner product of two vectors (5) can be calculated in any order. Typewise, our convention is that lowercase letters (e.g. n, m) denote the traditional dimension types (natural numbers), letting uppercase letters (e.g. A, B) denote other types and taking disjoint union $A + B$ for $m + n$, Cartesian product $A \times B$ for mn, unit type 1 for number 1, the empty set Ø for 0 and so on. Conversely, dimension n corresponds to the initial segment $\{1, 2, \ldots, n\}$ of the natural numbers up to n.

There is another "type" associated with matrices, namely the type of the elements (cells). The default view in linear algebra is to regard them as complex or real numbers, or (more generically) as inhabitants of an algebraic *field*. The minimal structure for composition (5) to work is that of a *semiring*, e.g. the natural numbers ($\mathbf{N}_0$) under addition and multiplication. Matrices whose cells are $\mathbf{N}_0$-valued are referred to as *counting matrices* and addressed in Appendix A. They include so-called Boolean matrices, whose cells are either 0 or 1.[9]

5. Cross tabulations in LA

Recall that the core of cross tabulation generation is formula (3), which is the matrix counterpart to relational projection (2). This section explains this construct starting by showing how the move from relations to matrices is obtained by encoding functions as matrices.

Building projection functions. Let A be an attribute of raw-data table T and let n be the number of records in T (namely rows, or lines in a spreadsheet). We write $T(A)$ to denote the column of T identified by attribute A, $T(A, y)$ to denote the element occupying the y-th position (row) in such a column, and $|\, A \,|$ to denote the range of values which can be found in $T(A)$. Column $T(A)$ can be regarded as a function which tells, for each row number $1 \leq r \leq n$, which value in $|\, A \,|$ can be found in row r of such a column. Such a function can be encoded as an elementary matrix t_A of type $|\, A \,| \leftarrow n$, defined as follows:

$$a \; t_A \; r = \begin{cases} 1 & \text{if } T(A, r) = a \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

These projections can be identified with the *bitmaps* of [WOS06], regarded as matrices. In our running example (Figs. 1, 2) $n = 6$ and we want to build these matrices for attributes $Model$ and $Color$. The projection $|\, Model \,| \xleftarrow{t_{Model}} n$ associated to dimension $Model$ is matrix

$$\begin{array}{r|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline Chevy & 1 & 1 & 0 & 0 & 0 & 0 \\ Ford & 0 & 0 & 1 & 1 & 1 & 1 \end{array} \tag{18}$$

and projection $|\, Color \,| \xleftarrow{t_{Color}} n$ associated to dimension $Color$ is matrix

$$\begin{array}{r|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline Blue & 0 & 1 & 0 & 1 & 0 & 1 \\ Green & 0 & 0 & 1 & 0 & 0 & 0 \\ Red & 1 & 0 & 0 & 0 & 1 & 0 \end{array} \tag{19}$$

[9] Boolean operations can be implemented in $\{0, 1\} \subseteq \mathbf{N}_0$ by defining $a \wedge b = ab$, $a \vee b = a + b - ab$ and $\neg a = 1 - a$, which are all closed in $\{0, 1\}$. This is not, however, required in the sequel.

Note that, typewise, the composition of matrices t_{Color} and t°_{Model} makes sense, leading to matrix

$$t_{Color} \cdot t^{\circ}_{Model} = \begin{array}{r|cc} & \mathit{Chevy} & \mathit{Ford} \\ \hline \mathit{Blue} & 1 & 2 \\ \mathit{Green} & 0 & 1 \\ \mathit{Red} & 1 & 1 \end{array} \tag{20}$$

of type $|\ Color\ | \leftarrow |\ Model\ |$, which essentially counts the number of sale records per color and model. In general, given attribute values $a \in |\ A\ |$ and $b \in |\ B\ |$, the cell in $t_A \cdot t^{\circ}_B$ addressed by a and b counts the number of rows of the source dataset T in which both a and b occur in the A and B columns, respectively:

$$a\,(t_A \cdot t^{\circ}_B)\,b \;=\; \left\langle \sum n \;:\; T(A,n) = a \wedge T(B,n) = b :\; 1 \right\rangle \tag{21}$$

The derivation of (21) will be given shortly.[10]

The diagonal construction. In order to sum up the number of vehicles sold rather than just counting sale records we need to identify a *measure* attribute, that is, a numeric attribute of T to be used for consolidation. In the case of Fig. 1 only $Sales$ applies. Because such numeric data have to become available for both projection matrices of (3), to the left and to the right, the chosen column is converted into a diagonal matrix as already shown in (14).

Notation $[\![T]\!]_M$ will be used to denote the diagonal matrix representation of measure attribute M in T. Index-wise, this corresponds to the following definition:

$$j\,[\![T]\!]_M\,i = \begin{cases} T(M,j) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

Definition (72) in Appendix B gives a pointfree alternative to (22) which is better suited for calculational purposes.

LA script for cross tabulation. We are in position to run formula (3) for T as in Fig. 1, $A = Color$ and $B = Model$. The evaluation of $t_{Color} \cdot [\![T]\!]_{Sales} \cdot t^{\circ}_{Model}$ yields another matrix of type $|\ Color\ | \leftarrow |\ Model\ |$

$$\begin{array}{r|cc} & \mathit{Chevy} & \mathit{Ford} \\ \hline \mathit{Blue} & 87 & 106 \\ \mathit{Green} & 0 & 64 \\ \mathit{Red} & 5 & 8 \end{array} \tag{23}$$

which we will denote by $ctab^{Sales}_{Color \leftarrow Model}(T)$ relying on the definition

$$\begin{array}{l} ctab^{M}_{A \leftarrow B}(T) : |\ A\ | \leftarrow |\ B\ | \\ ctab^{M}_{A \leftarrow B}(T) \;=\; t_A \cdot [\![T]\!]_M \cdot t^{\circ}_B \end{array} \tag{24}$$

—recall (3)—whose pointwise meaning is

$$a\,(ctab^{M}_{A \leftarrow B}(T))\,b = \left\langle \sum n \;:\; T(A,n) = a \wedge T(B,n) = b :\; T(M,n) \right\rangle \tag{25}$$

as will be shown briefly. In words: we sum all cells $T(M, n)$ with n ranging over all rows such that $T(A, n)$ and $T(B, n)$ respectively hold the attribute values a and b being consolidated (ie. related). The derivation of (25) relies on some rules for pointwise matrix manipulation given in Appendix A.

[10] This situation (*counting*), which is what Excel outputs wherever the measure attribute chosen in pivot table calculation is not numeric, corresponds to formula (3) wherever the middle matrix is the identity.

Note the style of the equational proof where each step is labeled with references to the laws applied, written inside the curly braces that follow the equality symbol (=):

$$
\begin{array}{ll}
& a\,(ctab^{M}_{A \leftarrow B}(T))\,b \\
= & \quad \{ \text{ definition (24) } \} \\
& a\,(t_A \cdot [\![T]\!]_M \cdot t_B^{\circ})\,b \\
= & \quad \{ \text{ matrix composition (5) twice ; converse of } t_B \ \} \\
& \langle \Sigma\ n\ ::\ (a\,t_A\ n) \times \langle \Sigma\ m\ ::\ (n\,[\![T_M]\!]\,m) \times (b\,t_B\ m) \rangle \rangle \\
= & \quad \{ \ [\![T_M]\!] \text{ is diagonal (22) } \} \\
& \langle \Sigma\ n\ ::\ (a\,t_A\ n) \times \langle \Sigma\ m\ :\ m = n :\ (n\,[\![T_M]\!]\,m) \times (b\,t_B\ m) \rangle \rangle \\
= & \quad \{ \text{ one point rule (quantifying over } m = n) \ \} \\
& \langle \Sigma\ n\ ::\ (a\,t_A\ n) \times (n\,[\![T_M]\!]\,n) \times (b\,t_B\ n) \rangle \\
= & \quad \{ \text{ ``trading'' over Boolean cells } a\,t_A\ n \text{ and } b\,t_B\ n \text{ (see Appendix A) } \} \\
& \langle \Sigma\ n\ :\ a\,t_A\ n \wedge b\,t_B\ n :\ n\,[\![T_M]\!]\,n \rangle \\
= & \quad \{ \text{ pointwise meaning of projections } t_A,\, t_B \text{ (17) and diagonal } [\![T_M]\!] \text{ (22) } \} \\
& \langle \Sigma\ n\ :\ T(A, n) = a \wedge T(B, n) = b :\ T(M, n) \rangle
\end{array}
$$

□

Clearly, (21) is a corollary of (25) since, for $[\![T_M]\!] = id$, $n\,[\![T_M]\!]\,n = 1$:

$$a\,(t_A \cdot t_B^{\circ})\,b \;=\; a\,(t_A \cdot id \cdot t_B^{\circ})\,b \;=\; \langle \Sigma\ n\ :\ T(A, n) = a \wedge T(B, n) = b :\ 1 \rangle$$

Grand totals. If compared to Fig. 2, cross tabulation (23) misses the two row and column grand totals. These are easily obtained via *"bang" matrices*. Let us explain what these are and our choice of terminology. In functional programming, the popular "bang" function, which is of type $1 \leftarrow A$ (parametric on A, $\forall\, A$) and usually denoted by symbol "!", is a polymorphic constant function yielding the unique value which inhabits the singleton type 1. The encoding of this function in LA format will be the row vector $1 \xleftarrow{!_A} A$ wholly filled up with 1s. For instance, $!_{|Model|}$ will be the vector with $|\ Model\ |$-many positions all holding number 1.[11]

Clearly, the composition of row vector $1 \xleftarrow{!} A$ with any column vector of type $A \xleftarrow{v} 1$ computes a scalar: the sum of all cells in v. Thus one can define a generic *totalizer* operator,

$$tot\ X = \left[\frac{id}{!}\right] \cdot X \cdot \left[\frac{id}{!}\right]^{\circ} \tag{26}$$

which equips X with three other blocks

$$\left[\begin{array}{c|c} X & X \cdot !^{\circ} \\ \hline ! \cdot X & ! \cdot X \cdot !^{\circ} \end{array}\right] \tag{27}$$

two sum (row and column) vectors and the *grand total* scalar $! \cdot X \cdot !^{\circ}$.[12]

[11] For the purposes in this paper, type 1 can be regarded as the singleton set {ALL}. This corresponds to the *Grand Total* in Fig. 2 and is consistent with the way ALL is used in e.g. [GCB^{+}97], as explained later in Sect. 8.

[12] The transformation of (26) into (27) follows immediately from the matrix laws of Sect. 4.

By adding totals to $ctab$ (24) we define

$$
\begin{aligned}
&tctab^{M}_{A \leftarrow B}(T) : | A | +1 \leftarrow | B | +1 \\
&tctab^{M}_{A \leftarrow B}(T) = tot(ctab^{M}_{A \leftarrow B}(T)) = \begin{bmatrix} t_A \\ ! \end{bmatrix} \cdot [\![T]\!]_M \cdot \begin{bmatrix} t_B \\ ! \end{bmatrix}^{\circ}
\end{aligned}
\qquad (28)
$$

which computes the standard cross-tabulation of raw data table T with respect to dimensions A, B and measure M. Note how types (dimensions) are added with 1, the singleton type containing the distinguished element ALL labelling grand totals. In our running example, this corresponds to enriching (23) with the extra row and column corresponding to the *bang* vectors of (26), both labeled with ALL:

	Chevy	*Ford*	ALL
Blue	87	106	193
Green	0	64	64
Red	5	8	13
ALL	92	178	270

(29)

Such is the outcome of evaluating $tctab^{Sales}_{Color \leftarrow Model}(T)$, which finally achieves the effect of Fig. 2 involving LA operations only.

As illustration of how these LA-based operations can be encoded in commercial languages dealing with matrices, such as e.g. MATLAB [13], listing 1 provides MATLAB code for the generation of the *bang* vector of size r, the tot operator (26) and the calculation of cross tabulations (24,28).

Finally, among several properties of *bang* vectors we single out

$$[\, ! | ! \,] = ! \qquad (30)$$

$$! \triangledown A = A = A \triangledown ! \qquad (31)$$

where (31) identifies ! as the unit of Khatri–Rao product. Since this is associative too, one can rely on its finitary extension to a sequence of n matrices A_i (all sharing the same input type, for $1 \leq i \leq n$) by writing $\bigtriangledown_{i=1}^{n} A_i$ or even

$$\bigtriangledown_{i \leftarrow s} A_i \qquad (32)$$

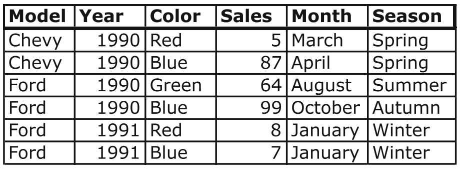
where s is a finite sequence of indices.[14] This extension will be useful in the generation of data cubes to be given in Sect. 8. Prior to this, we address below another operation central to OLAP: *roll-up*.

```
function R = bang(r)
   R = ones(1,r);
end

function R = tot(M)
   [n,m] = size(M);
   R = [ eye(n) ; bang(n) ] ∗ M ∗ [ eye(m) ; bang(m)]';
end

function R = ctab(tA,c,tB)
   [n,k] = size(c);
   [a,i] = size(tA);
   [b,j] = size(tB);
   if ˜(k==1 & i==n & j == n)
        error('Dimensions must agree');
   else id = eye(n);
          D = kr(c',id);
          R = tA∗D∗tB';
   end
end

function R = tctab(tA,c,tB)
   R = tot(ctab(tA,c,tB))
end
```

Listing 1: MATLAB encoding of *bang* (!), tot and of cross table calculation ($ctab$ and $tctab$), where the measure column is parameter c (a vector). This is converted to a diagonal as in (14) via the Khatri–Rao auxiliary operator *kr* taken from the *Tensorlab* library [SBL14].

[13] MATLAB ™ is a trademark of The MathWorks ®.

[14] Thus $\bigtriangledown_{i \leftarrow []} A_i = !$ and $\bigtriangledown_{i \leftarrow (k:s)} = A_k \triangledown (\bigtriangledown_{i \leftarrow s} A_i)$, where [] denotes the empty sequence and $(k : s)$ denotes the appending of head k to sequence s.



Model	Year	Color	Sales	Month	Season
Chevy	1990	Red	5	March	Spring
Chevy	1990	Blue	87	April	Spring
Ford	1990	Green	64	August	Summer
Ford	1990	Blue	99	October	Autumn
Ford	1991	Red	8	January	Winter
Ford	1991	Blue	7	January	Winter

Fig. 3. Augmented collection of raw data

6. "Rolling up" on functional dependencies

Rolling up means replacing a dimension by another which is more general in some sense (e.g. grouping, classification, containment). The latter is therefore "higher" in a dimension hierarchy which somehow acts as a *classification* or *taxonomy* of data records.

A simple way of seeing roll-up at work is the acknowledgement of functional dependencies (FDs) in data [Mai83]. Let us, for instance, augment the raw data of our running example with two new columns recording the month and season of each sale, as displayed in Fig. 3. Look, for instance, at the column labelled $Season$ telling in which season ($Spring$, $Summer$, $Autumn$ or $Winter$) the particular sales took place. Clearly, FD $Season \leftarrow Month$ holds, as no sales are recorded in the same month and in different seasons. This possibly happens because the $Season$ and $Month$ columns result from a join of the original table with some other table recording that $Season$ is higher than $Month$ in the temporal dimension hierarchy.[15]

Roll-up matrices. In general, a functional dependency $B \leftarrow A$ will hold in a table T iff no pair of rows can be found in T in which the values of attribute A are the same and those of attribute B differ ("B is determined by A"):

$$\langle \forall\ n, m\ :\ T(A, n) = T(A, m):\ T(B, n) = T(B, m) \rangle \qquad (33)$$

In the style of [Oli14a], we will write $B \xleftarrow{T} A$ to mean (33), abbreviated to $B \leftarrow A$ wherever T is implicit. As is shown in Appendix A (33) can be expressed solely in terms of projection matrices:

$$B \xleftarrow{T} A \quad \Leftrightarrow \quad t_A^\circ \cdot t_A \leq t_B^\circ \cdot t_B \qquad (34)$$

Whenever $B \xleftarrow{T} A$ holds, B acts as a *classifier* for A, meaning that every cross tabulation involving A can be *rolled-up* into another (less detailed) one involving B instead. In general, we define the *roll-up* matrix $|\, B \,| \xleftarrow{t_{B\leftarrow A}} |\, A \,|$ associated to FD $B \leftarrow A$ by

$$t_{B\leftarrow A} = \lfloor t_B \cdot t_A^\circ \rfloor \qquad (35)$$

where $\lfloor M \rfloor$ denotes the *support* of a given matrix M (59): the matrix of the same type whose non-zero cells are mapped to 1.

For instance, let us compute $t_{Season} \cdot t_{Month}^\circ$ (aside). This is a matrix of natural numbers *counting* the number of records in which a particular relationship holds, for instance $January$ versus $Winter$, which turns up twice. Quantities are not that important here; what matters is the univocal *relation* between

	$January$	$March$	$April$	$August$	$October$
$Spring$	0	1	1	0	0
$Summer$	0	0	0	1	0
$Autumn$	0	0	0	0	1
$Winter$	2	0	0	0	0

[15] The fact that T is not *normalized* in general reflects the preparation process of merging into the same data warehouse different tables of a (normalized) database.

Month and *Season* (*January* belongs to *Winter* only, not to two or more seasons) and this is obtained by taking the support of this matrix, yielding the roll-up matrix

$$t_{Season \leftarrow Month} = \lfloor t_{Season} \cdot t^{\circ}_{Month} \rfloor = \begin{array}{rccccc} & \mathit{January} & \mathit{March} & \mathit{April} & \mathit{August} & \mathit{October} \\ \hline \mathit{Spring} & 0 & 1 & 1 & 0 & 0 \\ \mathit{Summer} & 0 & 0 & 0 & 1 & 0 \\ \mathit{Autumn} & 0 & 0 & 0 & 0 & 1 \\ \mathit{Winter} & 1 & 0 & 0 & 0 & 0 \end{array} \qquad (36)$$

So, given a cross tabulation matrix $|\,A\,| \xleftarrow{X} |\,C\,|$, the effect of rolling it up across a given FD $B \leftarrow A$ is another cross tabulation given by matrix $t_{B \leftarrow A} \cdot X$ of type $|\,B\,| \leftarrow |\,C\,|$, to which totals can be added, e.g. $tot(t_{B \leftarrow A} \cdot X)$. Converse (transpose) caters for the same effect on the right-hand side: rolling X up across another FD $C \leftarrow D$ yields matrix $X \cdot t^{\circ}_{C \leftarrow D}$ of type $|\,A\,| \xleftarrow{X} |\,D\,|$. We illustrate this below by instantiating X with a cross tabulation from *Model* to *Month*

$$ctab^{Sales}_{Month \leftarrow Model}(T) = \begin{array}{rcc} & \mathit{Chevy} & \mathit{Ford} \\ \hline \mathit{January} & 0 & 15 \\ \mathit{March} & 5 & 0 \\ \mathit{April} & 87 & 0 \\ \mathit{August} & 0 & 64 \\ \mathit{October} & 0 & 99 \end{array} \qquad (37)$$

which, once composed with roll-up matrix (36) yields the expected rolling up effect, once equipped with totals:

$$tot(t_{Season \leftarrow Month} \cdot ctab^{Sales}_{Month \leftarrow Model}(T)) = \begin{array}{rccc} & \mathit{Chevy} & \mathit{Ford} & \textsc{all} \\ \hline \mathit{Spring} & 92 & 0 & 92 \\ \mathit{Summer} & 0 & 64 & 64 \\ \mathit{Autumn} & 0 & 99 & 99 \\ \mathit{Winter} & 0 & 15 & 15 \\ \textsc{all} & 92 & 178 & 270 \end{array} \qquad (38)$$

Note that we could have computed $tctab^{Sales}_{Season \leftarrow Model}(T)$ in one go, without the help of the roll-up matrix, obtaining the same result as (38). The general result expresses the *fusion* between roll-up matrices and cross-tabulations as follows:

$$tot(t_{B \leftarrow A} \cdot ctab^{M}_{A \leftarrow C}(T)) = tctab^{M}_{B \leftarrow C}(T) \qquad \Leftarrow \qquad B \xleftarrow{T} A \qquad (39)$$

To prove (39) if suffices, looking at the definition of *tctab* (28), to cancel *tot* on both sides and prove that $t_{B \leftarrow A} \cdot ctab^{M}_{A \leftarrow C}(T) = ctab^{M}_{B \leftarrow C}(T)$ holds modulo the same side-condition.

Before doing this, let us see a counter-example in which the side condition does not hold: we compose (37) with $t_{Color \leftarrow Month}$ (adjacent matrix, on top) obtaining the bottom-left adjacent matrix. This differs from the direct calculation of $ctab^{Sales}_{Color \leftarrow Model}(T)$ (bottom-right adjacent matrix) because *roll-up* matrix $t_{Color \leftarrow Month}$ does not capture a functional dependence: *Month* does not determine *Color*, as the *January* column shows.[16]

$$\begin{array}{rccccc} & \mathit{January} & \mathit{March} & \mathit{April} & \mathit{August} & \mathit{October} \\ \hline \mathit{Blue} & 1 & 0 & 1 & 0 & 1 \\ \mathit{Green} & 0 & 0 & 0 & 1 & 0 \\ \mathit{Red} & 1 & 1 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{rcc} & \mathit{Chevy} & \mathit{Ford} \\ \hline \mathit{Blue} & 87 & 114 \\ \mathit{Green} & 0 & 64 \\ \mathit{Red} & 5 & 15 \end{array} \qquad \begin{array}{rcc} & \mathit{Chevy} & \mathit{Ford} \\ \hline \mathit{Blue} & 87 & 106 \\ \mathit{Green} & 0 & 64 \\ \mathit{Red} & 5 & 8 \end{array}$$

[16] The support of matrix (20), given earlier, is another example of roll-up matrix which does not capture a functional dependence: *Ford* cars can be of any color, for instance.

The rest of the proof of (39) relies on properties of matrix *supports* which are deferred to Appendix A. Mind that projections are matrices which represent functions:

$$t_{B\leftarrow A} \cdot ctab^M_{A\leftarrow C}(T) = ctab^M_{B\leftarrow C}(T)$$

$$\Leftrightarrow \qquad \{ \text{ unfold definitions (35) and (24) } \}$$

$$\lfloor t_B \cdot t_A^\circ \rfloor \cdot t_A \cdot [\![T]\!]_M \cdot t_C^\circ = t_B \cdot [\![T]\!]_M \cdot t_C^\circ$$

$$\Leftarrow \qquad \{ \text{ Leibniz } \}$$

$$\lfloor t_B \cdot t_A^\circ \rfloor \cdot t_A = t_B$$

$$\Leftarrow \qquad \{ \text{ (66) in Appendix A } \}$$

$$t_A^\circ \cdot t_A \leq t_B^\circ \cdot t_B$$

$$\Leftrightarrow \qquad \{ \text{ (34) } \}$$

$$B \xleftarrow{T} A$$

□

Checking for FDs. Construction (35) enables us to check data sets for functional dependencies. In general, FD $B \leftarrow A$ will hold wherever matrix $t_B \cdot t_A^\circ$ is *functional*, or *simple*, equivalent to $t_{B\leftarrow A}$ being so. This terminology is imported from relational algebra [BdM97]: a matrix S will be said to be simple iff its *image* $S \cdot S^\circ$ is diagonal.[17] Instantiating S with $t_{Season} \cdot t_{Month}^\circ$, for instance, it can be checked that its image

	Spring	*Summer*	*Autumn*	*Winter*
Spring	2	0	0	0
Summer	0	1	0	0
Autumn	0	0	1	0
Winter	0	0	0	4

is diagonal, while that of (20)

	Blue	*Green*	*Red*
Blue	5	2	3
Green	2	1	1
Red	3	1	2

is not. Thus, FD $Color \leftarrow Model$ does not hold.

7. Incremental (parallel) construction

Cross tabulations as defined by formula (28) can be built incrementally under certain conditions. For instance, suppose one is given yesterday's cross tabulation and today's new data. Then today's cross tabulation (in matrix form) will be obtained by adding (matrix-wise) to yesterday's cross tabulation the cross tabulation of today's raw data.

Viewed from another perspective, this property allows one to *parallelize* the computation of a cross tabulation by partitioning the raw data and then summing up the cross tabulation of each partition of the raw data. Such a property, which can be regarded as generalization of the linearity property that makes linear applications parallel, can be stated by writing, given dimensions A and B, measure M and raw data sources T and T',

$$tctab^M_{A\leftarrow B}(T;\ T') = tctab^M_{A\leftarrow B}\, T + tctab^M_{A\leftarrow B}\, T' \tag{40}$$

where $T'' = T;\ T'$ denotes the append of the two data sources, i.e. T'' is a raw data table with the records of database T catenated with those of T'. T can be regarded as yesterday's raw data and T' as the new data,

[17] See Appendix A for more details on this diagonal characterization of FDs.

assuming that T has remained the same (no updates, no deletes). Alternatively, one may regard $T;\ T'$ as a partition of T'' intended for *divide-and-conquer* construction of its $tctab^{M}_{A\leftarrow B}$ cross tabulation.

We show below that (40) follows from facts

$$t''_A = \left[\, t_A | t'_A \,\right] \tag{41}$$

$$t''_B = \left[\, t_B | t'_B \,\right] \tag{42}$$

$$\llbracket T;\ T' \rrbracket_M = \llbracket T \rrbracket_M \oplus \llbracket T' \rrbracket_M \tag{43}$$

where $\oplus$ builds a diagonal matrix by direct sum (11) of two diagonal matrices. Equations (41) and (42) express that projection matrices for T'' can be built by gluing the corresponding projection matrices t_A, t'_A and t_B, t'_B built for T and for T', respectively. Note that, for (41) and (42) to be properly typed, t_A and t'_A (resp. t_B and t'_B) must have the same target type $|\ A\ |$ (resp. $|\ B\ |$) which can be easily ensured by taking sufficiently large $|\ A\ |$ and $|\ B\ |$.

To prove facts (41) to (43) we need better definitions for projections (17) and diagonals (22) saving expensive pointwise reasoning. Such definitions and proofs (given in Appendix B) can be regarded as a detour needed to smoothly move from first order database notation to linear algebra notation, linking projections (bitmaps) and diagonals to the basic linear algebra of Sect. 4.

Assuming (41) to (43), the proof of (40) follows from the definition of cross tabulation (28) by a simple equational argument resorting to the laws of matrix algebra:

$$
\begin{array}{ll}
& tctab^{M}_{A\leftarrow B}(T;\ T') \\
= & \quad \{\ (28)\ \} \\
& \left[\dfrac{t''_A}{!}\right] \cdot \llbracket T;\ T' \rrbracket_M \cdot \left[\dfrac{t''_B}{!}\right]^{\circ} \\
= & \quad \{\ (41)\ ;\ (42) \text{ and } (43)\ \} \\
& \left[\dfrac{\left[\, t_A | t'_A \,\right]}{!}\right] \cdot (\llbracket T \rrbracket_M \oplus \llbracket T' \rrbracket_M) \cdot \left[\dfrac{\left[\, t_B | t'_B \,\right]}{!}\right]^{\circ} \\
= & \quad \{\ (30) \text{ twice } ;\ \text{abide law } (10) \text{ twice}\ \} \\
& \left[\left[\dfrac{t_A}{!}\right] \middle| \left[\dfrac{t'_A}{!}\right] \right] \cdot (\llbracket T \rrbracket_M \oplus \llbracket T' \rrbracket_M) \cdot \left[\left[\dfrac{t_B}{!}\right] \middle| \left[\dfrac{t'_B}{!}\right] \right]^{\circ} \\
= & \quad \{\ \text{absorption } (12)\ ;\ \text{converse-duality } (6)\ \} \\
& \left[\left[\dfrac{t_A}{!}\right] \cdot \llbracket T \rrbracket_M \middle| \left[\dfrac{t'_A}{!}\right] \cdot \llbracket T' \rrbracket_M \right] \cdot \left[\dfrac{\left[\dfrac{t_B}{!}\right]^{\circ}}{\left[\dfrac{t'_B}{!}\right]^{\circ}} \right] \\
= & \quad \{\ \text{divide and conquer } (7)\ \} \\
& \left[\dfrac{t_A}{!}\right] \cdot \llbracket T \rrbracket_M \cdot \left[\dfrac{t_B}{!}\right]^{\circ} + \left[\dfrac{t'_A}{!}\right] \cdot \llbracket T' \rrbracket_M \cdot \left[\dfrac{t'_B}{!}\right]^{\circ} \\
= & \quad \{\ (28) \text{ twice}\ \} \\
& tctab^{M}_{A\leftarrow B}\, T + tctab^{M}_{A\leftarrow B}\, T'
\end{array}
$$

□

In retrospect, this proof establishes *tctab* (28) as a structure preserving map (homomorphism) between raw data *collection* and (cross tabulation) matrix *addition*, enabling the extraction of parallelism in a formal and direct way.

8. Higher-dimensional OLAP

This section extends cross tabulations towards higher dimensions. The aim is to formulate a basis for a general LA theory for n-dimensional OLAP, dealing with all data summary levels presented in [GCB+97], from 0 to 3-dimensional summaries, respectively: aggregate, group-by, cross-tab and cube. The approach goes further by allowing any number n of dimensions.

The proposed generalization depends on the Khatri–Rao product (13) that works as a Cartesian product on matrix types, thus a Cartesian product of the dimensions. As an illustration, remember the projections of our running example and apply the Khatri–Rao product to t_{Model} (18) and t_{Color} (19). The outcome is matrix

		1	2	3	4	5	6
Chevy	*Blue*	0	1	0	0	0	0
Chevy	*Green*	0	0	0	0	0	0
Chevy	*Red*	1	0	0	0	0	0
Ford	*Blue*	0	0	0	1	0	1
Ford	*Green*	0	0	1	0	0	0
Ford	*Red*	0	0	0	0	1	0

bearing type $|\, Model \times Color \,| \leftarrow 6$. This tells in which rows the particular dimension pairs appear, compare with Fig. 1. Put in other words, this matrix is the higher-rank projection $t_{Model\times Color}$ of the Cartesian product of the two dimensions. In general,

$$t_{A\times B} = t_A \triangledown t_B \tag{44}$$

Thus $t_{Model\times Year\times Color} = t_{Model} \triangledown t_{Year} \triangledown t_{Color}$, which is projection

			1	2	3	4	5	6
Chevy	1990	*Blue*	0	1	0	0	0	0
Chevy	1990	*Green*	0	0	0	0	0	0
Chevy	1990	*Red*	1	0	0	0	0	0
Chevy	1991	*Blue*	0	0	0	0	0	0
Chevy	1991	*Green*	0	0	0	0	0	0
Chevy	1991	*Red*	0	0	0	0	0	0
Ford	1990	*Blue*	0	0	0	1	0	0
Ford	1990	*Green*	0	0	1	0	0	0
Ford	1990	*Red*	0	0	0	0	0	0
Ford	1991	*Blue*	0	0	0	0	0	1
Ford	1991	*Green*	0	0	0	0	0	0
Ford	1991	*Red*	0	0	0	0	1	0

(45)

capturing the whole dimensional part of the raw-data table of Fig. 1.

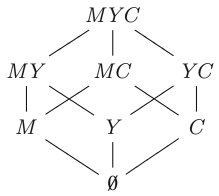
Multidimensional cross tabulations are obtained via the same formula (28) just by supplying higher-rank projections, for instance $tctab^{Sales}_{Model\times Color\leftarrow Year}(T)$ which yields:

		1990	1991	ALL
Chevy	*Blue*	87	0	87
Chevy	*Green*	0	0	0
Chevy	*Red*	5	0	5
Ford	*Blue*	99	7	106
Ford	*Green*	64	0	64
Ford	*Red*	0	8	8
ALL		255	15	270

corresponding to $A = Model \times Color$ and $B = Year$ in (28). Furthermore, by composing $[\![T]\!]_{Sales}$ with the projection of all dimensions given by (45) on the left and totalizing by $!^{\circ}$ on the right, we obtain the following column-vector representation of Fig. 1,

$$t_{Model \times Year \times Color} \cdot [\![T]\!]_{Sales} \cdot !^{\circ} = \begin{array}{lll|r} & & & \text{ALL} \\ \hline Chevy & 1990 & Blue & 87 \\ Chevy & 1990 & Green & 0 \\ Chevy & 1990 & Red & 5 \\ Chevy & 1991 & Blue & 0 \\ Chevy & 1991 & Green & 0 \\ Chevy & 1991 & Red & 0 \\ Ford & 1990 & Blue & 99 \\ Ford & 1990 & Green & 64 \\ Ford & 1990 & Red & 0 \\ Ford & 1991 & Blue & 7 \\ Ford & 1991 & Green & 0 \\ Ford & 1991 & Red & 8 \end{array} \quad (46)$$

which, as we shall soon see, is a fragment of the CUBE operator.



A generalization follows from this example. Given a finite, ordered set of *dimensions* D, one calculates the corresponding *cube* over some given *measure* attribute by iterating over the powerset 2^D of D, for instance that represented aside for $D = \{M, Y, C\}$ where M, Y and C abbreviate $Model$, $Year$ and $Color$, respectively.

Let us denote by $2^D_\star$ the sequence of all elements of 2^D ordered in some predefined way induced by the ordering on the dimensions (e.g. $M < Y < C$). Thus $2^D_\star$ is a sequence of (dimension) sequences and we can build the following projection matrix as an iteration of (44) via (32)

$$\begin{aligned} & t_{2^D_\star} : \mid 2^D_\star \mid \leftarrow n \\ & t_{2^D_\star} = \underset{s \leftarrow 2^D_\star}{\left[\frac{\quad}{\quad}\right]} \big(\underset{d \leftarrow s}{\nabla} t_d \big) \end{aligned} \quad (47)$$

where $\left[\frac{\quad}{\quad}\right]$ denotes the finitary extension of vertical blocking (recall Sect. 4) thus stacking up the intermediate projection matrices provided by the innermost iteration.

Note that (47) is not a function (functional matrix) although each contribution $\nabla_{d \leftarrow s}\, t_d$ is so.[18] This redundancy is intentional, as (47) is intended to record all possible combinations of dimension attributes—the *shape* of the cube. To fill such a shape with the cube contents we multiply by the measure diagonal and totalize with *bang* converse:

$$\begin{aligned} & cube^M_D(T) : \mid 2^D_\star \mid \leftarrow 1 \\ & cube^M_D(T) = t_{2^D_\star} \cdot [\![T]\!]_M \cdot !^{\circ} \end{aligned} \quad (48)$$

Thus the LA representation of a cube is a (column) vector. Aside we show a tabular representation of $cube^{Sales}_{\{Model, Year, Color\}}(T)$ for our running example. Note the usual convention of filling with ALL marks the "missing attributes" in each s in $2^D_\star$.

			ALL
Chevy	1990	*Blue*	87
Chevy	1990	*Red*	5
Ford	1990	*Blue*	99
Ford	1990	*Green*	64
Ford	1991	*Blue*	7
Ford	1991	*Red*	8
Chevy	1990	ALL	92
Ford	1990	ALL	163
Ford	1991	ALL	15
Chevy	ALL	*Blue*	87
Chevy	ALL	*Red*	5
Ford	ALL	*Blue*	106
Ford	ALL	*Green*	64
Ford	ALL	*Red*	8
ALL	1990	*Blue*	186
ALL	1990	*Green*	64
ALL	1990	*Red*	5
ALL	1991	*Blue*	7
ALL	1991	*Red*	8
Chevy	ALL	ALL	92
Ford	ALL	ALL	178
ALL	1990	ALL	255
ALL	1991	ALL	15
ALL	ALL	*Blue*	193
ALL	ALL	*Green*	64
ALL	ALL	*Red*	13
ALL	ALL	ALL	270

Report [MO11a] gives a MATLAB script which implements (48). A generic formula for calculating other aggregations on given sub-sequences S of $2^D_\star$ and measure M from a database table T is given by

$$\begin{aligned} & agg^M_S(T) : \mid S \mid \leftarrow 1 \\ & agg^M_S(T) = t_S \cdot [\![T]\!]_M \cdot !^{\circ} \end{aligned} \quad (49)$$

[18] Given two functions f and g, $\left[\frac{f}{g}\right]$ is never a function—it is a relation. Also note that $\mid 2^D_\star \mid = \sum_{s \leftarrow 2^D_\star} \mid s \mid$.

where t_S generalizes (47). S tells which dimensions in D are handled and in what order, thus yielding different standard operations for different S. For instance, for S containing only the empty sequence [] one has $t_{[]} = !$ in (49), thus obtaining an AGGREGATE [GCB+97]—the *grand total* block of *tot* (26,27). At the other extreme, for $S = 2_\star^D$ (49) is of course the same as (48), the whole data CUBE. Somewhere between these limit cases one finds, for $S = [s]$ some singleton subsequence of $2_\star^D$, the GROUP- BY s aggregation. Finally, for S a prefix-closed subsequence of $2_\star^D$—for instance, $[[Model, Color], [Model], []]$—(49) evaluates a ROLL-UP.

```
SELECT Color, Sum(Sales)
FROM T
GROUP BY Color
```

The authors of [GCB+97] regard GROUP- BY as "an unusual relational operator". While the operator may look "unusual" in the context of the relation algebra which supports the semantics of relational databases, it makes perfect sense in the linear algebra semantics proposed in the current paper for such constructions. Moreover, note that our LA semantics for GROUP- BY not only covers the one-attribute case—captured e.g. the SQL syntax above, which evaluates to

$$agg^{Sales}_{[[Color]]}(T) = \begin{array}{r|l} & \text{ALL} \\ \hline Blue & 193 \\ Green & 64 \\ Red & 13 \end{array}$$

—but also covers any sequence of grouping attributes—recall e.g. (46), which is the outcome of $agg^{Sales}_{[[Model, Year, Color]]}(T)$. Clearly, any GROUP- BY, AGGREGATE or ROLL- UP is always a *fragment* of the cube which represents the whole multi-dimensional analysis of the source data.

9. Related work

An overview of data warehousing and OLAP technology can be found in [CD97]. Since Gray *et al* delivered their seminal data cube paper in 1996 [GBLP96], most work in the field has been concerned with techniques for efficient OLAP, given the small time window (usually at night) when warehouses can go offline for data refreshing.

Another evolution since 1996 is the development of industry standards and specifications. Query languages such as MDX [WZP02] relying on multidimensional expressions have emerged as SQL extensions providing the features needed to perform OLAP queries. Our work can be seen as the beginning of a "SQL-free" alternative to provide the same features. We focus on defining a semantics for such features which expresses their meaning in terms of linear algebra operations, ultimately using such meaning to calculate the results.

Yang et al [YJA03] focus on the problem of data cube construction and show how a cluster middleware, called ADR (originally developed for scientific data intensive applications) can be used for carrying out scalable implementations of the construction of data cubes.

Bearing the ideal of making OLAP "truly online", Ng et al [NWY01] develop a collection of parallel algorithms directed towards online and offline creation of data cubes using low cost PC clusters to parallelize computations.

Goil and Choudhary [GC01] address scalability in multidimensional systems for OLAP and multidimensional analysis and describe the PARSIMONY system providing a parallel and scalable infrastructure for multidimensional online analytical processing, used for both OLAP and data mining. Parallel algorithms are developed for data mining on the multidimensional cube structure for attribute-oriented association rules and decision-tree-based classification.

Literature on "end-to-end" system proposals for parallel OLAP servers is scarce. SIDERA [EDD+10] is one such proposal, providing OLAP-specific functionality gathering recent results in a common framework: *"the most comprehensive OLAP platform described in the current research literature"* [EDD+10].

Closer to our approach, Sun and others [STF06, STP+08] introduce a technique based on the use of tensors in the area of pattern discovery. (Tensors generalize vectors and matrices, as happens in the mathematical domain, and can be used to represent data-cubes.) To capture temporal evolution one uses tensor streams or sequences that are time indexed structures of tensors, the advantage being a generalization of traditional streams and sequences. On the background stays *singular value decomposition* (SVD), whose matricial expression conspicuously resembles our starting point (3) and suggests a link between the two approaches which we intend to study in the future.

Our work also intersects with the area of index-based database-query (response time) optimization, namely in what respects *bitmap* indices [WOS06]. Clearly, the projection matrices built in the current paper are bitmaps regarded as matrices. Bitmaps were first implemented in IBM's Model 204 [O'N89], becoming a "de facto" device after compression techniques solved their outrageous memory space demands. They are still in use in today's commercial database systems, see [WOS06] for details.

10. Conclusions and future work

This paper addresses the foundations of quantitative data science [DP12] from a linear algebra perspective. In particular, it shows how aggregation operations such as cross tabulations and data cubes used in quantitative data analysis can be expressed solely in terms of matrix multiplication, transposition and the Khatri–Rao product. The approach offers potential for deriving a truly algebraic theory of data consolidation, handling the quantitative as well as qualitative sides of data science in an elegant and typed way. Moreover, all operations involved, namely

- the conversion of *dimension* attributes into projection matrices
- the conversion of *measure* attributes into diagonal matrices
- the calculation of *cross tabulations*, and
- the calculation of data *cubes*

become parallel ("for free") as immediate consequence of the very basic law of *divide and conquer* (7).

Our main aim is to set up a framework allowing for algebraic reasoning about data analysis operations that have hitherto been described informally or by program code only. The approach is generic and extensible, as much as the underlying mathematics is so. Take for instance the following matrix capturing the $Season \leftarrow Month$ relationship in a more refined way:

$$\begin{array}{r|cccccccccccc}
 & \rotatebox{90}{\textit{January}} & \rotatebox{90}{\textit{February}} & \rotatebox{90}{\textit{March}} & \rotatebox{90}{\textit{April}} & \rotatebox{90}{\textit{May}} & \rotatebox{90}{\textit{June}} & \rotatebox{90}{\textit{July}} & \rotatebox{90}{\textit{August}} & \rotatebox{90}{\textit{September}} & \rotatebox{90}{\textit{October}} & \rotatebox{90}{\textit{November}} & \rotatebox{90}{\textit{December}} \\
\hline
Spring & 0 & 0 & 0.3 & 1 & 1 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 \\
Summer & 0 & 0 & 0 & 0 & 0 & 0.3 & 1 & 1 & 0.7 & 0 & 0 & 0 \\
Autumn & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 & 1 & 1 & 0.7 \\
Winter & 1 & 1 & 0.7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3
\end{array} \tag{50}$$

In this case, FD $Season \leftarrow Month$ does not strictly hold, for equinoctial and solsticial months are doubly classified in the seasons they border, in different proportions (70% for the season which ends, 30% for the one which starts).

One may say that a "fuzzy" data dependency holds in (50). In spite of the possible complexity that this extension to the standard situation might raise from a traditional OLAP perspective, in our setting it doesn't change anything, as such a "fuzzy" months-into-seasons roll-up process would work precisely in the same way: using this matrix [19] in (38), for instance, one would obtain

	Chevy	*Ford*	ALL
Spring	88.5	0	88.5
Summer	0	64	64
Autumn	0	99	99
Winter	3.5	15	18.5
ALL	92	178	270

indicating that some (between 3 and 4) of the 92 *Chevy*s sold are likely to have been Winter sales rather than Spring sales. Note that (50) can be regarded as a *probabilistic function*, meaning that the linear algebra semantics of such functions as studied in e.g. [Oli12] can also be useful in this *data* (rather than *algorithmic*) context.

[19] Pre-composed with the obvious 5 → 12 *type coercion* matrix embedding five into twelve months, of course.

Future work. Further research in the direction of thoroughly justifying our approach is under way [MO14]. In the current paper, the data cube construction is derived from that of cross tabulation. [MO14] exploits the alternative view of regarding the data cube as the primitive construction wherefrom the other 2D, 1D and 0D aggregators are derived. This makes it easier to prove a number of results, for instance the commutation between cube construction and generic *vectorization* [MO13].

Moreover, we have to better cross-check our matrix encoding of OLAP (and FDs) with already existing OLAP formal models [DT99, PKL02]. Mimicking OLAP algebra (whatever this means) in terms of linear algebra may provide better and simpler proofs for existing results and generate new ones, as our experience in pointfree calculation already shows in the relational algebra field [Oli14a]. This research agenda should also include, of course, a closer look at [STF06].

Extending the LA encoding to other forms of data consolidation such as e.g. *averaging* is within reach. Averaging rather than summing up measure vectors is obtained once again via *bang* matrices and scalar division, $avg\ v = (! \cdot v)/(! \cdot !^\circ)$, for $n \xleftarrow{v} 1$ and $1 \xleftarrow{!} n$, where $! \cdot v$ reduces vector v to the scalar which records the sum of its elements. Averaging holds since $(! \cdot !^\circ)$ is $1 \xleftarrow{n} 1$, also a scalar. It is easy to see that obtaining cross tabulations consolidated by averaging is a question of augmenting equation (3) with the (index-wise) division of the cross tabulation matrix by the corresponding counting matrix (taking care of divisions by 0, of course):

$$\frac{t_A \cdot [\![T]\!]_M \cdot t_B^\circ}{t_A \cdot t_B^\circ}$$

Extremes (min and max) are achievable by tuning multiplication and sum of matrix elements to suitable semirings. But calculating more exotic data consolidation forms as e.g. population's standard deviation is challenging due to the complexity of the formulas. This is achievable with intensive use of Khatri–Rao products and other non-trivial matrix operations, but further research is needed to evaluate the practicality of such usage.

Another direction for future work is to benchmark a realistic implementation of our approach (derivable from the MATLAB scripts) against existing OLAP systems (e.g. those mentioned in Sect. 9) thus testing whether the parallelism inherent in the LA scripts materializes in real-life applications. Recall that our approach is column-driven. Given that column-store databases for OLAP are being used as an alternative to ROLAP (relational row-driven OLAP) or MOLAP (multidimensional OLAP), it would be interesting to analyze if our LA semantics for OLAP could also improve its processing [Sor12].

Clearly, one needs to be able to process *sparse matrices* (which our projection bitmaps and diagonals are) as efficiently as possible. Bell and Garland [BG09] explore the design of efficient sparse matrix-vector kernels for throughput oriented processors and implement these kernels in a parallel computing architecture developed by NVIDIA. The OSKI Library [WOV$^+$09] is a collection of low-level C primitives that provide automatically tuned computational kernels on sparse matrices, for use in solver libraries and applications. OSKI has a BLAS-style interface, providing basic kernels like sparse matrix-vector multiply and sparse triangular solve, among others.

Last but not least, Yang et al [YPS11] propose architecture-aware optimizations for sparse matrix multiplication on GPUs and study the impact of their efforts on graph mining. This work is another piece of evidence suggesting that future OLAP and data mining should rely on linear algebra.

Acknowledgements

The research reported in this paper was funded by project *LeanBigData: Ultra-Scalable and Ultra-Efficient Integrated and Visual Big Data Analytics* (FP7-619606), by ERDF—European Regional Development Fund through the COMPETE Programme (operational programme for competitiveness) and by National Funds through the FCT—Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project FCOMP-01-0124-FEDER-010047. While doing this work Hugo Macedo held FCT grant number SFRH/BD/33235/2007. The authors would like to thank the insightful comments, suggestions and references provided by the reviewers. Special thanks go to the reviewer who suggested definition (72) and drafted a version of the proof of (43). Comments by Gabriel David and Orlando Belo on an earlier version of this paper are also gratefully acknowledged.

Appendix A: Appendix on counting matrices and function injectivity

This appendix is concerned with functional dependencies and their relationship with *counting matrices*, that is, matrices whose cells are natural numbers. Although some results below hold for arbitrary matrices, we shall restrict to counting matrices for economy of presentation. As special case we have the *Boolean matrices*, so called because they hold either 0 or 1 in their cells, which can be interpreted as the Boolean truth values. Clearly, a counting matrix B is Boolean iff $B \leq \top$, where $\top$ denotes the everywhere-1 matrix of its type obtainable by composing "bang" (30) with its converse: $\top = !^\circ \cdot !$.

Boolean matrices represent binary relations in matricial form. Given Boolean matrices B and B', $B \leq B'$ expresses inclusion of the binary relations represented by such matrices, that is $\left\langle \forall \ y, x \ :: \ y \, B \, x \Rightarrow y \, B' \, x \right\rangle$ recalling our use of infix notation $y \, M \, x$ to express the cell of matrix M addressed by row y and column x.[20]

Any function f is a special case of the Boolean matrix such that $y \, f \, x = 1$ if $y = f \ x$ and $y \, f \, x = 0$ otherwise. Note the use of symbol f to denote two mathematical objects, the function itself (as in $y = f \ x$) and its matrix representation (as in $y \, f \, x = 1$). This abuse of notation (common in relation algebra) enables the following rules interfacing index-free and index-wise matrix notation, where f and g functional matrices:

$$y \, (g^\circ \cdot M \cdot f) x = (g \ y) \, M \, (f \ x) \tag{51}$$

$$y \, (f \cdot M) x = \left\langle \Sigma \ z \ : \ y = f \ z : \ z \, M \, x \right\rangle \tag{52}$$

$$y \, (M \cdot f^\circ) x = \left\langle \Sigma \ z \ : \ x = f \ z : \ y \, M \, z \right\rangle \tag{53}$$

These rules are expressed in the style of the Eindhoven quantifier calculus (see e.g. [BM06]) and are convenient shorthands for the corresponding instances of matrix composition (5). Rule (51) extends to typed matrix algebra a similar rule known from relation algebra [BB04]. Note how (52) is obtained from (5) by "trading" Boolean cell $y \, f \, z$ with the corresponding Boolean formula $y = f \ z$, as explained in [Oli13]. This can be done with any other Boolean term $y \, B \, x$.

Rule (51) is enough to derive the equalities

$$! \cdot f = ! \tag{54}$$

$$g^\circ \cdot (M \ \theta \ N) \cdot f = (g^\circ \cdot M \cdot f) \, \theta \, (g^\circ \cdot N \cdot f) \tag{55}$$

for suitably typed functions f and g and matrix-cell binary operation θ promoted to a matrix operator (with the usual notation overloading), that is, $y \, (M \ \theta \ N) x = (y \, M \, x) \, \theta \, (y \, N \, x)$. From (54) and $\top = !^\circ \cdot !$ one immediately draws:

$$\top \cdot f = \top \tag{56}$$

Supports. Let $n \in \mathbf{N}_0$ be a natural number and define its *support* $\lfloor n \rfloor = \mathit{if} \ n \geq 1 \ \mathit{then} \ 1 \ \mathit{else} \ 0$, that is, $\lfloor n \rfloor = n \downarrow 1$ where $m \downarrow n$ denotes the least of m or n. Clearly,

$$x \leq \lfloor n \rfloor \Leftrightarrow x \leq n \wedge x \leq 1 \tag{57}$$

that is, $\lfloor n \rfloor$ is the largest "Boolean number" (0 or 1) at most n. Thus $\lfloor 0 \rfloor = 0$, $\lfloor 1 \rfloor = 1$ and, in general $\lfloor n \rfloor = n \Leftrightarrow n \leq 1$: the support of a "Boolean number" (0 or 1) is itself.

Let us now extend $\lfloor n \rfloor$ from naturals to matrices of naturals (counting matrices): 0 becomes $\perp$, the everywhere-0 matrix of its type; 1 becomes $\top$, the everywhere-1 matrix of its type and (57) becomes

$$X \leq \lfloor N \rfloor \Leftrightarrow X \leq N \wedge X \leq \top \tag{58}$$

equivalent to the following, closed definition

$$\lfloor M \rfloor = M \downarrow \top \tag{59}$$

where $y \, (M \downarrow N) x = (y \, M \, x) \downarrow (y \, N \, x)$, overloading $m \downarrow n$.

Cancellation in (58) yields $\lfloor N \rfloor \leq N$ and $\lfloor N \rfloor \leq \top$, the latter saying that $\lfloor N \rfloor$ is a *Boolean* matrix. All equalities above extend to counting matrices, e.g. $\lfloor N \rfloor = N \Leftrightarrow N \leq \top$: the support of a Boolean matrix is itself. Moreover, $\lfloor \_ \rfloor$ is a monotonic function from counting to Boolean matrices. From (58) one also obtains (via converses):

$$\lfloor M^\circ \rfloor = \lfloor M \rfloor^\circ \tag{60}$$

[20] As advocated in [Oli13], this notation finds its inspiration in terms such as e.g. $y \leq x$ which one is familiar with since school maths.

In general $\lfloor M \cdot N \rfloor \neq \lfloor M \rfloor \cdot \lfloor N \rfloor$, since composition is not closed over Boolean matrices ($\top \cdot \top > \top$, for instance). Nevertheless, the special case

$$\lfloor M \cdot f \rfloor = \lfloor M \rfloor \cdot \lfloor f \rfloor = \lfloor M \rfloor \cdot f \tag{61}$$

holds:

$$
\begin{array}{ll}
& \lfloor M \cdot f \rfloor \\
= & \{ \text{ (59); (56) } \} \\
& (M \cdot f) \downarrow (\top \cdot f) \\
= & \{ \text{ (55) } \} \\
& (M \downarrow \top) \cdot f \\
= & \{ \text{ (59) } \} \\
& \lfloor M \rfloor \cdot f
\end{array}
$$

□

From (61) the more general rule

$$\lfloor g^\circ \cdot M \cdot f \rfloor = g^\circ \cdot \lfloor M \rfloor \cdot f \tag{62}$$

can be derived by taking converses:

$$
\begin{array}{ll}
& \lfloor g^\circ \cdot M \cdot f \rfloor \\
= & \{ \text{ contravariance ; idempotence } \} \\
& \lfloor (M^\circ \cdot g)^\circ \cdot f \rfloor \\
= & \{ \text{ (61) ; (60) } \} \\
& \lfloor M^\circ \cdot g \rfloor^\circ \cdot f \\
= & \{ \text{ (61) ; (60) } \} \\
& (\lfloor M \rfloor^\circ \cdot g)^\circ \cdot f \\
= & \{ \text{ contravariance } \} \\
& g^\circ \cdot \lfloor M \rfloor \cdot f
\end{array}
$$

□

A counting matrix M is *diagonal* iff $\lfloor M \rfloor \leq id$, that is, $\langle \forall\ y, x \ : \ y \neq x : \ y\, M\, x = 0 \rangle$. An example of diagonal matrix is the *image* $g \cdot g^\circ$ of a function g since, by (53), $b' (g \cdot g^\circ)\, b = \left\langle \Sigma\ a \ : \ b = g\ a : \ b'\ g\ a \right\rangle$ which is the same as $\left\langle \Sigma\ a \ : \ b' = g\ a \wedge b = g\ a : \ 1 \right\rangle$ trading term $b' = g\ a$, since g is a function. Thus

$$b' (g \cdot g^\circ)\, b = \left\langle \Sigma\ a \ : \ b' = g\ a \wedge b = g\ a \wedge b' = b : \ 1 \right\rangle \tag{63}$$

and therefore $b' (g \cdot g^\circ)\, b = 0$ for $b' \neq b$.

Functional injectivity. For $M := id$ one draws from (62) that $g^\circ \cdot f$ is Boolean, $\lfloor g^\circ \cdot f \rfloor = g^\circ \cdot f$. Thus the *kernel* of a function f [Oli14a]

$$f^\circ \cdot f = \lfloor f^\circ \cdot f \rfloor \tag{64}$$

is Boolean. By (51), Leibniz rule $x' = x \Rightarrow f\ x' = f\ x$ encodes into $id \leq f^\circ \cdot f$, whereby one obtains (by monotonicity) $g \leq g \cdot f^\circ \cdot f$ and

$$g \leq \lfloor g \cdot f^\circ \rfloor \cdot f \tag{65}$$

by taking supports and (61).

Functions can be compared by comparing their kernels: by unfolding $f^\circ \cdot f \leq g^\circ \cdot g$ once again by (51), we get:

$$
\begin{array}{ll}
 & x' (f^\circ \cdot f) x \leq x' (g^\circ \cdot g) x \\
\Leftrightarrow & \quad \{ \text{ (51) twice } \} \\
 & (f\ x') id (f\ x) \leq (g\ x') id (g\ x) \\
\Leftrightarrow & \quad \{ \ b\,(id)\,a \text{ encodes } b = a \text{ and } \leq \text{ over } \{0, 1\} \text{ encodes implication } \} \\
 & f\ x' = f\ x \Rightarrow g\ x' = g\ x
\end{array}
$$

Colloquially: "g does not distinguish what f regards as equal". Formally: g is *less injective* than f.[21] The following result

$$\lfloor g \cdot f^\circ \rfloor \cdot f = g \quad \Leftarrow \quad g \text{ is less injective than } f \tag{66}$$

is required in Sect. 6 of the current paper and relies on the injectivity ordering on functions:

$$
\begin{array}{ll}
 & \lfloor g \cdot f^\circ \rfloor \cdot f = g \\
\Leftrightarrow & \quad \{ \text{ (65) } \} \\
 & \lfloor g \cdot f^\circ \rfloor \cdot f \leq g \\
\Leftarrow & \quad \{ \ \lfloor g \cdot g^\circ \rfloor \cdot g \leq g \text{ by monotonicity of composition since } g \cdot g^\circ \text{ is diagonal (63) } \} \\
 & \lfloor g \cdot f^\circ \rfloor \cdot f \leq \lfloor g \cdot g^\circ \rfloor \cdot g \\
\Leftrightarrow & \quad \{ \text{ (61) twice } \} \\
 & \lfloor g \cdot f^\circ \cdot f \rfloor \leq \lfloor g \cdot g^\circ \cdot g \rfloor \\
\Leftarrow & \quad \{ \text{ monotonicity of } \lfloor \_ \rfloor \text{ and of composition } \} \\
 & f^\circ \cdot f \leq g^\circ \cdot g
\end{array}
$$

□

Appendix B: Appendix on bitmaps, projections and diagonals

Bitmaps. Suppose an array $a = \left[d_1\ d_2\ \ldots\ d_n \right]$ holds n elements of data type D. This uniquely determines the function $f_a : n \rightarrow D$ such that $f_a(i) = d_i$ for $1 \leq i \leq n$, that is, f_a tells which datum lives in which position of array a. Once such a function is represented as a Boolean matrix of type $D \leftarrow n$ one obtains a *bitmap* matrix representation of a. Note that a itself can be regarded as a *generalized*[22] D-valued row vector of type $1 \leftarrow n$. Let this change of representation can be captured by function

$$\mathit{bm} : (1 \ _D\!\leftarrow n) \rightarrow (D \leftarrow n)$$

where notation $1 \ _D\!\leftarrow n$ is intended to warn the reader that cells in bm's input are of type D, possibly not a semiring essential for matrix composition to work. (More about this below.) We define bm inductively as follows: for $n = 1$, $\mathit{bm}\ d_1 = D \xleftarrow{d_1} 1$, the Boolean (column) vector representing constant function $\underline{d_1}$; for $n > 1$—bm—is defined by:

$$\mathit{bm}\ \left[a_1 | a_2 \right] = \left[\mathit{bm}\ a_1 | \mathit{bm}\ a_2 \right] \tag{67}$$

Let a given raw data table T have n rows (records) and as many columns as the set of its attributes $S = \{A, B, \ldots\}$. Then T may also be regarded as a *generalized* matrix of type $n \leftarrow S$ whereby the raw-data append

[21] Cf. e.g. [Oli14a], where the same inequality is handled relationally.

[22] Generalized in the sense that it will hold any kind of heterogeneously typed data, not just numerical data.

operation $T;\ T'$ (catenation of T with T') is faithfully captured in matrix block notation by

$$T;\ T' = \left[\begin{array}{c} T \\ \hline T' \end{array}\right] \tag{68}$$

since both T and T' share the same input type S. For $A \in S$, constant function $S \xleftarrow{\underline{A}} 1$ is a Boolean vector with 0s everywhere but a 1 in the row addressed by attribute $A \in S$. Then $T(A, n)$, the value of attribute A in the n-th row of T can be re-written as follows:

$$\begin{array}{ll}
& T(A, n) \\
= & \quad \{ \text{ using infix notation once } T \text{ is regarded as a } n \leftarrow S \text{ matrix } \} \\
& n\ T\ A \\
= & \quad \{ \text{ since } S \xleftarrow{\underline{A}} 1 \text{ is a constant function } \} \\
& n\ T\,(\underline{A}\ 1) \\
= & \quad \{ \text{ (51) } \} \\
& n\,(T \cdot \underline{A})\,1
\end{array}$$

Thus

$$n \xleftarrow{T \cdot \underline{A}} 1 \tag{69}$$

is the (column) vector which represents the A-column of T. This can be turned into a bitmap via bm,

$$t_A = bm\,(T \cdot \underline{A})^\circ \tag{70}$$

providing a pointfree alternative to (17), as well as

$$t'_A = bm\,(T' \cdot \underline{A})^\circ \tag{71}$$

for another raw-data set T' sharing A-values in the same range type $|\ A\ |$.

Fact (41) can then be calculated as follows:

$$\begin{array}{ll}
& \left[\, t_A | t'_A \,\right] \\
= & \quad \{ \text{ (70) twice } \} \\
& \left[\, bm\,(T \cdot \underline{A})^\circ | bm\,(T' \cdot \underline{A})^\circ \,\right] \\
= & \quad \{ \text{ (67) } \} \\
& bm\ \left[\, (T \cdot \underline{A})^\circ | (T' \cdot \underline{A})^\circ \,\right] \\
= & \quad \{ \text{ converse-duality (6) } \} \\
& bm\ \left[\begin{array}{c} T \cdot \underline{A} \\ \hline T' \cdot \underline{A} \end{array}\right]^\circ \\
= & \quad \{ \text{ fusion (9) } \} \\
& bm\,(\left[\begin{array}{c} T \\ \hline T' \end{array}\right] \cdot \underline{A}^\circ) \\
= & \quad \{ \text{ define } T'' = \left[\begin{array}{c} T \\ \hline T' \end{array}\right] = T;\ T' \text{ (68) } \} \\
& bm\,(T'' \cdot \underline{A}^\circ) \\
= & \quad \{ \text{ (70) } \} \\
& t''_A
\end{array}$$

□

The proof of (42) is the same, for attribute B instead of A.

Diagonals. Back to (22), let $M \in S$ be a measure attribute and let us rely on (69) to capture its diagonalization, via (14):

$$\llbracket T \rrbracket_M = (T \cdot \underline{M})^\circ \triangledown id \tag{72}$$

This definition is convenient for proving fact (43), as follows:

$$\begin{array}{cl}
& \llbracket T;\ T' \rrbracket_M \\
= & \quad \{\ (68)\ \} \\
& \llbracket \left[\begin{array}{c} T \\ \hline T' \end{array}\right] \rrbracket_M \\
= & \quad \{\ \text{definition (72)}\ \} \\
& (\left[\begin{array}{c} T \\ \hline T' \end{array}\right] \cdot \underline{M})^\circ \triangledown id \\
= & \quad \{\ \text{fusion (9)}\ ;\ id \oplus id = id\ \} \\
& \left[\begin{array}{c} T \cdot \underline{M} \\ \hline T' \cdot \underline{M} \end{array}\right]^\circ \triangledown (id \oplus id) \\
= & \quad \{\ \text{converse-duality (6)}\ \} \\
& \left[\,(T \cdot \underline{M})^\circ \middle| (T' \cdot \underline{M})^\circ\,\right] \triangledown (id \oplus id) \\
= & \quad \{\ \text{(16) since } T \cdot \underline{M} \text{ and } T' \cdot \underline{M} \text{ are row vectors}\ \} \\
& ((T \cdot \underline{M})^\circ \triangledown id) \oplus ((T' \cdot \underline{M})^\circ \triangledown id) \\
= & \quad \{\ \text{definition (72) twice}\ \} \\
& \llbracket T \rrbracket_M \oplus \llbracket T' \rrbracket_M
\end{array}$$

□

Recall from Sect. 7 that (43) is central to showing that cross tabulation evaluation is parallelizable (40).

References

[BB04] Backhouse K, Backhouse RC (2004) Safety of abstract interpretations for free, via logical relations and Galois connections. Sci Comput Programm 15(1–2):153–196

[BdM97] Bird R, de Moor O (1997) Algebra of programming. In: Hoare CAR (ed) Series in computer science. Prentice-Hall International, New Jersey

[BG09] Bell N, Garland M (2009) Implementing sparse matrix-vector multiplication on throughput-oriented processors. In: Proceedings of the conference on high performance computing networking, storage and analysis, SC'09. ACM, New York, pp 18:1–18:11

[Bir89] Bird RS (1989) Lecture notes on constructive functional programming, 1989. In: Broy M (ed) CMCS Int. Summer School directed by F.L. Bauer [et al.], vol 55. Springer, NATO Adv. Science Institute (Series F: Comp. and System Sciences), Berlin

[BM06] Backhouse RC, Michaelis D (2006) Exercises in quantifier manipulation. In: Uustalu T (ed) MPC'06. LNCS, vol 4014. Springer, Berlin, pp 70–81

[CD97] Chaudhuri S, Dayal U (1997) An overview of data warehousing and OLAP technology. SIGMOD Rec. 26:65–74

[Cod70] Codd EF (1970) A relational model of data for large shared data banks. CACM 13(6):377–387

[DGM14] Desharnais J, Grinenko A, Möller B (2014) Relational style laws and constructs of linear algebra. J Logic Algebr Meth Program 83(2):154–168

[DP12] Davenport TH, Patil DJ (2012) Data scientist: the sexiest job of the 21st century. Oct Harv Bus Rev

[DT99] Datta A, Thomas H (1999) The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. Dec Supp Syst 27(3):289–301

[EDD+10] Eavis T, Dimitrov G, Dimitrov I, Cueva D, Lopez A, Taleb A (2010) Parallel OLAP with the Sidera server. Future Gener Comput Syst 26(2):259–266
[Fri02] Frias MF (2002) Fork algebras in algebra, logic and computer science. In: Logic and computer science. World Scientific Publishing Co., Singapore
[GBLP96] Gray J, Bosworth A, Layman A, Pirahesh H (1996) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-total. In: Su SYW (ed) Proceedings of the 12th int. conf. on data engineering, Feb. 26–Mar. 1, 1996, New Orleans, Louisiana. IEEE Computer Society, New York, pp 152–159
[GC97] Goil S, Choudhary A (1997) High performance OLAP and data mining on parallel computers. Data Min Knowl Discov 1:391–417
[GC01] Goil S, Choudhary A (2001) Parsimony: an infrastructure for parallel multidimensional analysis and data mining. J Parallel Distrib Comput 61(3):285–321
[GCB+97] Gray J, Chaudhuri S, Bosworth A, Layman A, Reichart D, Venkatrao M, Pellow F, Pirahesh H (1997) Data cube: a relational aggregation operator generalizing group-by, cross-tab, and sub-totals. Data Min Knowl Disc 1(1):29–53
[GL97] Gyssens M, Lakshmanan LVS (1997) A foundation for multi-dimensional databases. VLDB J 106–115
[JLN00] Johnson T, Lakshmanan LV, Ng RT (2000) The 3w model and algebra for unified data mining. VLDB 21–32
[JPT10] Jensen CS, Pedersen TB, Thomsen C (2010) Multidimensional databases and data warehousing. In: Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael
[Mac12] Macedo H. (2012) Matrices as arrows—why categories of matrices matter. PhD thesis, University of Minho, October, MAPi PhD programme
[Mai83] Maier D (1983) The theory of relational databases. Computer Science Press, Rockville
[MO10] Macedo HD, Oliveira JN (2010) Matrices as arrows! A biproduct approach to typed linear algebra. In: MPC, LNCS, vol 6120. Springer, Berlin, pp 271–287
[MO11a] Macedo HD, Oliveira JN (2011) Do the two middle letters of "OLAP" stand for linear algebra ("LA")? Technical report TR-HASLab:4:2011, HASLab, U.Minho & INESC TEC, July. http://wiki.di.uminho.pt/twiki/bin/view/DI/FMHAS/TechnicalReports
[MO11b] Macedo HD, Oliveira JN (2011) Towards linear algebras of components. In: FACS 2010 of LNCS, vol 6921. Springer, Berlin, pp 300–303
[MO13] Macedo HD, Oliveira JN (2013) Typing linear algebra: a biproduct-oriented approach. Sci Comput Program 78(11):2160–2191
[MO14] Macedo HD, Oliveira JN (2014) Typed linear algebra for the data scientist (In preparation)
[NWY01] Ng RT, Wagner A, Yin Y (2001) Iceberg-cube computation with PC clusters. SIGMOD Rec. 30:25–36
[Oli09] Oliveira JN (2009) Extended static checking by calculation using the pointfree transform. LNCS, vol 5520. Springer, Berlin, pp 195–251
[Oli11] Oliveira JN (2011) Pointfree foundations for (generic) lossless decomposition. Technical report TR-HASLab:3:2011, HASLab, U.Minho & INESC TEC. http://wiki.di.uminho.pt/twiki/bin/view/DI/FMHAS/TechnicalReports.
[Oli12] Oliveira JN (2012) Towards a linear algebra of programming. Formal Aspects Comput 24(4–6):433–458
[Oli13] Oliveira JN (2013) Weighted automata as coalgebras in categories of matrices. Int J Found Comp Sci 24(06):709–728
[Oli14a] Oliveira JN (2014) A relation-algebraic approach to the “Hoare logic” of functional dependencies. JLAP 83(2):249–262
[Oli14b] Oliveira JN (2014) Relational algebra for “just good enough" hardware. In: RAMiCS. LNCS, vol 8428. Springer, Berlin, pp 119–138
[O’N89] O’Neil P (1989) Model 204 architecture and performance. In: Gawlick D, Haynie M, Reuter A (ed) High performance transaction systems. Lecture notes in computer science, vol 359. Springer, Berlin, pp 39–59
[PJ01] Pedersen TB, Jensen CS (2001) Multidimensional database technology. Computer 34:40–46
[PKL02] Park C-S, Kim MH, Lee Y-J (2002) Finding an efficient rewriting of OLAP queries using materialized views in data warehouses. Dec Supp Syst 32(4):379–399
[RR98] Rao C.R., Rao M.B. (1998) Matrix algebra and its applications to statistics and econometrics. World Scientific Pub Co Inc
[SBL14] Sorber L, Barel M, Lathauwer L (2014) Tensorlab v2.0: a MATLAB toolbox for tensor computations, January. http://www.tensorlab.net
[Sch11] Schmidt G (2011) Relational mathematics. Encyclopedia of mathematics and its applications, vol 132, Cambridge U.P.
[Sor12] Sorjonen S (2012) OLAP query performance in column-oriented databases. Columnar databases seminar, DCS. University of Helsinki. https://www.cs.helsinki.fi/en/courses/58312305/2012/s/s/1.
[STF06] Sun J, Tao D, Faloutsos C (2006) Beyond streams and graphs: dynamic tensor analysis. In: KDD’06: proc. of the 12th ACM SIGKDD int. conf. on knowledge discovery and data mining. ACM, New York, pp 374–383
[STP+08] Sun J, Tao D, Papadimitriou S, Yu PS, Faloutsos C (2008) Incremental tensor analysis: theory and applications. ACM Trans Knowl Discov Data 2:11:1–11:37
[VS99] Vassiliadis P, Sellis T (1999) A survey of logical models for OLAP databases. SIGMOD Rec 28(4):64–69
[WOS06] Wu K, Otoo EJ, Shoshani A (2006) Optimizing bitmap indices with efficient compression. ACM Trans Database Syst 31:1–38
[WOV+09] Williams S, Oliker L, Vuduc R, Shalf J, Yelick K, Demmel J (2009) Optimization of sparse matrix-vector multiplication on emerging multicore platforms. Parallel Comput 35:178–194
[WZP02] Whitehorn M, Zare R, Pasumansky M (2002) Fast track to MDX. Springer, Berlin
[YJA03] Yang G, Jin R, Agrawal G (2003) Implementing data cube construction using a cluster middleware: algorithms, implementation experience, and performance evaluation. Future Gener Comput Syst 19(4):533–550
[YPS11] Yang X, Parthasarathy S, Sadayappan P (2011) Fast sparse matrix-vector multiplication on GPUs: implications for graph mining. Proc VLDB Endowment 4:231–242

Received 25 April 2013
Revised 14 August 2014
Accepted 18 August 2014 by Eerke Boiten
Published online 30 September 2014