The Data Cube as a Typed Linear Algebra Operator

J.N. Oliveira
HASLAB - High Assurance Software Laboratory
INESC TEC & University of Minho
Braga, Portugal
jno@di.uminho.pt

H.D. Macedo
Department of Engineering
Aarhus University
Aarhus, Denmark
hdm@eng.au.dk

ABSTRACT

There is a need for a typed notation for linear algebra applicable to the fields of econometrics and data mining. In this paper we show that such a notation exists and can be useful in formalizing and reasoning about data aggregation operations.

One such operation — the construction of a *data cube* — is shown to be easily expressible as a linear algebra operator. The construction is shown to be type-generic and some of its properties are derived from its typed definition and proved using matrix algebra. Other forms of data aggregation such as eg. *rollup* and *cross tabulation* are shown to be algebraically derivable from data cubes.

CCS CONCEPTS

• **Theory of computation → Data modeling**; Logic and databases;

ACM Reference Format:
J.N. Oliveira and H.D. Macedo. 2017. The Data Cube as a Typed Linear Algebra Operator. In *Proceedings of DBPL 2017, Munich, Germany, September 1, 2017,* 11 pages.
https://doi.org/10.1145/3122831.3122834

1 INTRODUCTION

In [2] Abadir and Magnus stress on the need for a standardized notation for linear algebra in the field of econometrics and statistics. More recently, the authors have shown how data consolidation can be expressed in *typed* linear algebra [11], a categorial approach [10] to linear algebra which has shown useful elsewhere in the quantitative side of the software sciences, both at behaviour [14, 18] and data [17] level.

The acronym LAoP (for *Linear Algebra of Programming*) [15] captures this research trend, exposing linear algebra as an evolution of the (relational) algebra of programming [4] applicable to handling quantitative aspects of software modeling and design.

The current paper resumes the work reported in [11] by showing how to express all data aggregation operators in LAoP, focussing on the *data cube* as central construction. Our main aim is to formalize previous work in the field — see e.g. [6] and [19] — in an unified way, so as to develop a linear algebra basis for data aggregation useful in query optimization and data mining in general.

DBPL 2017, September 1, 2017, Munich, Germany
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-5354-0/17/09…$15.00
https://doi.org/10.1145/3122831.3122834

Chevy	1990	Red	5
Chevy	1990	Blue	87
Ford	1990	Green	64
Ford	1990	Blue	99
Ford	1991	Red	8
Ford	1991	Blue	7

Figure 1: Sample raw data table T.

Contribution. In [11] the data cube construction is derived from that of cross tabulation. This paper exploits the alternative view of regarding the data cube as the *primitive construction* of data analysis, 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 data-cube construction and generic *vectorization* [10]. This calls for a generalization, given in the current paper, of the so-called *Roth's relationship* [22] which Abadir and Magnus [2] regard as the fundamental result of the whole theory of vectorization.

2 BUILDING DATA CUBES

Given some raw data as in Fig. 1, let *Model*, *Year*, *Color* and *Sale* be the names of the three attributes involved, each corresponding to a "column" in the table displayed (Fig.1). There are $n = 6$ rows in the data set, each corresponding to a data record. Non-numeric attributes *Model*, *Year* and *Color* are regarded as *dimensions* (the first, second and third columns in Fig.1, from left to right, pictured in grey) while numeric attributes are regarded as *measures* (the fourth column in the same figure, pictured white).

Dimensions. Data analysis involves many analytical procedures, which in particular include *consolidation* of measure quantities across the several ways *dimensions* can be organized. Reference [11] shows how to express such operations using linear algebra constructs. In short, dimension columns (attributes) are represented by *projection* functions of type $n \to A$, where n is the number of records and A is the attribute labelling the column being represented. Note the simplified notation: n abbreviating set $\{1, ..., n\}$ and A abbreviating the set of corresponding attribute values. In the example,

$$Year = \{1990, 1991\}$$
$$Model = \{Chevy, Ford\}$$
$$Color = \{Blue, Green, Red\}$$

are dimension attributes. Following the notation conventions of [11], given data set T (note the uppercase T) we denote by $t_A : n \to A$

(note the lowercase t) the projection function corresponding to the A column of T. For instance, $t_{Color}\ 3 = Green$.

Let $f : A \to B$ and $g : A \to C$ be functions with the same source type. We denote by $f \triangledown g : A \to B \times C$ the *pairing* of f and g, that is, the function defined by

$$(f \triangledown g)\ a = (f\ a, g\ a) \tag{1}$$

For instance,

$$(t_{Color} \triangledown t_{Model})\ 2 = (Blue, Chevy),$$
$$(t_{Year} \triangledown (t_{Color} \triangledown t_{Model}))\ 3 = (1990, (Green, Ford))$$

and so on.

The LAoP approach of [10, 11] consists in representing such projection functions by (Boolean) matrices, as follows: let $f : A \to B$ be a function, where A and B are finite. Function f can be represented by a matrix $\llbracket f \rrbracket$ with A-many columns and B-many rows such that, for any $b \in B$ and $a \in A$, matrix cell [1] $b\ \llbracket f \rrbracket\ a = 1$ if $b = f\ a$, otherwise $b\ \llbracket f \rrbracket\ a = 0$. For instance, matrix $\llbracket t_{Model} \rrbracket$ is as follows:

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

Likewise, it can be easily shown that

$$(b, c)\ \llbracket f \triangledown g \rrbracket\ a = (b\ \llbracket f \rrbracket\ a) \times (c\ \llbracket g \rrbracket\ a) \tag{2}$$

holds, since multiplication within $\{0, 1\}$ implements logic conjunction.

As in [10, 11] we shall abuse of notation and (very conveniently, as we shall see) drop the parentheses from $\llbracket f \rrbracket$. This is consistent with writing $f \triangledown g$ to denote the operation above, which in fact corresponds to a well-known matrix operator, the so-called *Khatri-Rao product* [21] $M \triangledown N$ of two arbitrary matrices M and N, defined index-wise by

$$(b, c)\ (M \triangledown N)\ a \quad = \quad (b\ M\ a) \times (c\ N\ a) \tag{3}$$

Thus the Khatri-Rao product is a "column-wise" version of the well-known *Kronecker product* $M \otimes N$ defined by:

$$(y, x)\ (M \otimes N)\ (b, a) \quad = \quad (y\ M\ b) \times (x\ N\ a) \tag{4}$$

Both products are intimately related by the *absorption* law

$$(M \otimes N) \cdot (P \triangledown Q) = (M \cdot P) \triangledown (N \cdot Q) \tag{5}$$

valid for any (suitably typed) matrices M, N, P, Q [8].

Given two functions $g : A \to B$ and $f : B \to C$, their *composition* $f \cdot g$ is defined by

$$(f \cdot g)\ a = f\ (g\ a)$$

Matrix-wise, one can define $\llbracket g \rrbracket \cdot \llbracket f \rrbracket$ too, where the dot means matrix *multiplication* too. In general, given two matrices $N : A \to B$ and $M : B \to C$, their *composition* (or multiplication) is the matrix $M \cdot N$ defined by:

$$c\ (M \cdot N)\ a \quad = \quad \langle \Sigma\ b\ ::\ c\ M\ b \times b\ N\ a \rangle \tag{6}$$

Note how we also extend the arrow notation used to type functions to also type arbitrary matrices, $M : A \to B$ meaning the type of a matrix with A-many columns and B-many rows. Wherever a matrix has one sole row it is said to be a *row vector* and we write e.g. $v : A \to 1$ to say this. Type $1 = \{\textsc{all}\}$ is the singleton type whose unique element ALL will play a significant role in data consolidation, as we shall soon see.

[1] Following the infix notation usually adopted for relations (which are Boolean matrices), for instance $y \leqslant x$, we write $y\ M\ x$ to denote the contents of the cell in matrix M addressed by row y and column x. This and other notational conventions of the linear algebra of programming LAoP are explained in detail in [16].

Year x (Color x Model)			ALL
1990	Blue	Chevy	87
		Ford	99
	Green	Chevy	0
		Ford	64
	Red	Chevy	5
		Ford	0
1991	Blue	Chevy	0
		Ford	7
	Green	Chevy	0
		Ford	0
	Red	Chevy	0
		Ford	8

Figure 2: Typed matrix representation of raw data T of Fig.1.

Given a type A, there is a unique row vector wholly filled with 1s. This is termed "bang" [11] and denoted by $! : A \to 1$. There is also a unique square matrix of type $A \to A$ whose diagonal is wholly filled with 1s and otherwise is filled with 0s — it is termed the *identity* and is denoted by $id : A \to A$. This is the unit of matrix composition: $M \cdot id = M = id \cdot M$.

Given a matrix $M : A \to B$ we can always define its *transposition*, or *converse* $M^\circ : B \to A$ defined by $a\ M^\circ\ b = b\ M\ a$. The following laws hold: $(M^\circ)^\circ = M$ (idempotence) and $(M \cdot N)^\circ = N^\circ \cdot M^\circ$ (contravariance). Clearly, the converse of a *row* vector $v : A \to 1$ is a *column* vector $v^\circ : 1 \to A$.

Measures. Measure columns in source data sets (e.g. the rightmost column of T in Fig.1) will be represented by row vectors of type $n \to 1$ whose cells contain the corresponding numeric data. We use superscripted notation $t^M : n \to 1$ to distinguish *projection* matrices (e.g. t_A) from *measure* vectors (e.g. t^M). Thus $t^{Sale} = \begin{bmatrix} 5 & 87 & 64 & 99 & 8 & 7 \end{bmatrix}$ in our running example.

The (typed) matrix representation of T (Fig.1) is given by the column vector

$$v = (t_{Year} \triangledown (t_{Color} \triangledown t_{Model})) \cdot (t^{Sale})^\circ \tag{7}$$

of type $v : 1 \to (Year \times (Color \times Model))$, which is depicted in Fig. 2. Compared to Fig.1 it has 12 rows whose 0s correspond to combinations of dimensions not present in Fig.1.

Totalisers. Given any matrix $M : A \to B$, the expression $! \cdot M$ (where $! : B \to 1$) is the row vector (of type $A \to 1$) that contains the sums (totals) of all columns of M,

$$\textsc{all}\ (! \cdot M)\ a \quad = \quad \langle \Sigma\ b\ ::\ b\ M\ a \rangle$$

since cell ALL $(!)$ b is 1 for all b in (6). Functions are the only Boolean matrices satisfying property

$$! \cdot f = ! \tag{8}$$

Given a type A, we define its *totalizer* matrix $\tau_A : A \to A + 1$ by

$$\tau_A \quad : \quad A \to (A + 1)$$
$$\tau_A \quad = \quad \left[\begin{array}{c} id \\ \hline ! \end{array}\right] \tag{9}$$

This calls for some extra explanations: given types A and B, we denote by $A + B$ their *disjoint union* (thus $A + 1$ "adds" ALL to type A); given two matrices $M : C \to A$ and $N : C \to B$ with the same input type C, combinator $\left[\frac{M}{N}\right]$ describes the matrix with type $C \to A+B$ which glues M and N vertically [10]. The same operation, gluing horizontally rather than vertically is given by:

$$\left[\; M \mid N \;\right]^\circ = \left[\frac{M^\circ}{N^\circ}\right] \tag{10}$$

As is well known,

$$\left[\; M \mid N \;\right] \cdot \left[\frac{P}{Q}\right] = M \cdot P + N \cdot Q \tag{11}$$

captures the *divide & conquer* property of matrix composition. Knowing the following *fusion* property of this combinator,

$$\left[\frac{M}{N}\right] \cdot P = \left[\frac{M \cdot P}{N \cdot P}\right] \tag{12}$$

it is clear that, given $M : A \to B$, $\tau_B \cdot M = \left[\frac{id}{!}\right] \cdot M = \left[\frac{M}{! \cdot M}\right]$ since $id \cdot M = M$. That is, $(\tau_B \cdot M) : A \to B + 1$ yields a copy of M on top of the row vector of its column sums, for instance:

$$\tau \cdot \begin{bmatrix} 50 & 40 & 85 & 115 \\ 50 & 10 & 85 & 75 \end{bmatrix} = \begin{bmatrix} 50 & 40 & 85 & 115 \\ 50 & 10 & 85 & 75 \\ 100 & 50 & 170 & 190 \end{bmatrix}$$

For the special of M being a function $f : A \to B$ we have

$$\tau_B \cdot f = \left[\frac{f}{!}\right] \tag{13}$$

cf. (8).

Recall the standard *biproduct projections* [10] $\pi_1 : A + B \to A$ and $\pi_1 : A+B \to B$ such that $\left[\frac{\pi_1}{\pi_2}\right] = id$. From property $\pi_1 \cdot \left[\frac{M}{N}\right] = M$ one immediately obtains a way of cancelling a totalizer:

$$\pi_1 \cdot \tau_A = id \tag{14}$$

The other cancellation yields:

$$\pi_2 \cdot \tau_A = \,! \tag{15}$$

Cubes. Data cubes are obtained rather simply from products of totalizers. Recall the Kronecker (tensor) product $M \otimes N$ of two matrices $M : A \to B$ and $N : C \to D$, which is of type $(M \otimes N) : A \times C \to B \times D$ (5). Clearly, the product

$$\tau_A \otimes \tau_B : A \times B \to (A + 1) \times (B + 1)$$

provides for totalization on two dimensions. Indeed, type $(A + 1) \times (B + 1)$ is isomorphic to $A \times B + A + B + 1$, whose four parcels represent the four elements of the "dimension powerset of $\{A, B\}$".

Take vector $v : 1 \to Year \times (Color \times Model)$ above (7) which encodes raw data T as a column vector and build vector $c : 1 \to (Year + 1) \times ((Color + 1) \times (Model + 1))$ as follows:

$$c = (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot v \tag{16}$$

This yields a vector representing the corresponding *data cube*, depicted in Fig. 3. Putting (16) together with (7) via property (5) we obtain the final formula which extracts c from the corresponding measure column,

$$c = (t'_{Year} \triangledown (t'_{Color} \triangledown t'_{Model})) \cdot (t^{Sale})^\circ \tag{17}$$

(Year+1) x ((Color+1) x (Model+1))			ALL
1990	Blue	Chevy	87
		Ford	99
		ALL	186
	Green	Chevy	0
		Ford	64
		ALL	64
	Red	Chevy	5
		Ford	0
		ALL	5
	ALL	Chevy	92
		Ford	163
		ALL	255
1991	Blue	Chevy	0
		Ford	7
		ALL	7
	Green	Chevy	0
		Ford	0
		ALL	0
	Red	Chevy	0
		Ford	8
		ALL	8
	ALL	Chevy	0
		Ford	15
		ALL	15
ALL	Blue	Chevy	87
		Ford	106
		ALL	193
	Green	Chevy	0
		Ford	64
		ALL	64
	Red	Chevy	5
		Ford	8
		ALL	13
	ALL	Chevy	92
		Ford	178
		ALL	270

Figure 3: Cube for measure *Sale* calculated from the raw data of Fig.1 using typed linear algebra.

where t'_A abbreviates the expression $\tau_A \cdot t_A$, for each dimension A. By (12) and (8), $t'_A = \left[\frac{t_A}{!}\right]$, since t_A is a function. Summing up, a data *cube* is a multi-dimensional column vector — a special case of a *tensor* in the standard terminology [23].

Pointwise versus pointfree notation. Before generalizing the above definition of a data cube, it is worthwhile reflecting briefly on the expressive power of definition (16) and showing how much notation one saves in adopting it.

By taking the expansion (17) of (16) and moving to pointwise notation as given by (6) and (3) we get that the entry in the cube corresponding to year y (or ALL mark), color z (or ALL mark) and model x (or ALL mark) — that is, cell $(y,(z,x))\ c\ \textsc{all}$ in Fig. 3 — is given by

$$\langle \sum n\ ::\ (y,(z,x))\ (t'_{Year} \triangledown (t'_{Color} \triangledown t'_{Model}))\ n \times (n\ t^{Sale}\ \textsc{all})\rangle$$

Now, from $t'_{Year} = \left[\frac{t_{Year}}{!}\right]$ we get

$$y\ t'_{Year}\ n = \textbf{if}\ (y = \textsc{all} \vee y = t_{Year}\ n)\ \textbf{then}\ 1\ \textbf{else}\ 0$$

and similarly for the other attributes. Putting everything together, we get

$$(y,(z,x))\ c\ \textsc{all} = \langle \sum n\ :\ \left\{\begin{array}{l} y = \textsc{all} \vee y = t_{Year}\ n \\ z = \textsc{all} \vee z = t_{Color}\ n \\ x = \textsc{all} \vee x = t_{Model}\ n \end{array}\right.\ :\ n\ t^{Sale}\ \textsc{all}\rangle$$

where the brace means logic conjunction. Using the convention of [17] of abbreviating cell $x\ v\ \textsc{all}$ of a vector $v : A \leftarrow 1$ by $v\ [x]$, we obtain

$$c\ [y,z,x] = \langle \sum n\ :\ \left\{\begin{array}{l} y = \textsc{all} \vee y = Year\ [n] \\ z = \textsc{all} \vee z = Color\ [n] \\ x = \textsc{all} \vee x = Model\ [n] \end{array}\right.\ :\ Sale\ [n]\rangle$$

as pointwise expansion of (16). This gives and indication of how painful it would be to adopt such a pointwise notation throughout the reasoning and calculations to be found in the rest of this paper.

3 GENERALIZING DATA CUBES

Although Fig. 3, corresponding to construction (16), is the usual way to present data cubes (measure vectors addressed by all possible combinations of dimension attribute values), in our approach a cube is not necessarily a (column) vector. As will be shown shortly, the cube construction works also over multi-dimensional data aggregations.

The key to such a generic construction of data cubes is (generalized) *vectorization* [10], a kind of "matrix currying": given a matrix $M : A \times B \to C$, with $A \times B$-many columns and C-many rows, it makes sense to think of reshaping M into its *vectorized* version $\mathbf{vec}_A\ M : B \to A \times C$ with B-many columns and $A \times C$-many rows. Such matrices, M and $\mathbf{vec}_A\ M$, are *isomorphic* in the sense that they contain the same information in different formats[2], that is,

$$c\ M\ (a,b) \quad = \quad (a,c)\ (\mathbf{vec}_A\ M)\ b \tag{18}$$

holds for every a, b, c.

In fact, M can be retrieved back from $\mathbf{vec}_A\ M$ by devectorizing it. Thus we have the equivalence

$$V = \mathbf{vec}_A\ M \quad \equiv \quad \mathbf{unvec}_A\ V = M \tag{19}$$

This equivalence is studied in detail in [10].

As examples, let us devectorize vector $v : 1 \to Year \times (Color \times Model)$ (7) across $Year$, obtaining a matrix (say M) of type $Year \to Color \times Model$:

$M : Year \to Color \times Model$

$$M \quad = \quad \mathbf{unvec}_{Year}\ v \quad = \quad \begin{array}{ll|cc} & & 1990 & 1991 \\ \hline Blue & Chevy & 87 & 0 \\ & Ford & 99 & 7 \\ \hline Green & Chevy & 0 & 0 \\ & Ford & 64 & 0 \\ \hline Red & Chevy & 5 & 0 \\ & Ford & 0 & 8 \end{array} \tag{20}$$

We can further devectorize M, this time across $Color$:

$N : Color \times Year \to Model$

$N = \mathbf{unvec}_{Color}\ M =$

	Blue		*Green*		*Red*	
	1990	1991	1990	1991	1990	1991
Chevy	87	0	0	0	5	0
Ford	99	7	64	0	0	8

(21)

It turns out that calculating the cube of such two-dimensional versions M and N of the same data amounts to using totalizers both on inputs (conversed) and outputs. For instance, the cube of matrix $N : Color \times Year \to Model$ above is given by

$$\text{cube}\ N : (Color + 1) \times (Year + 1) \to (Model + 1)$$

$$\text{cube}\ N = \tau_{Model} \cdot N \cdot (\tau_{Color} \otimes \tau_{Year})^{\circ}$$

which can be depicted as follows:

	Blue			*Green*			*Red*			ALL		
	1990	1991	ALL	1990	1991	ALL	1990	1991	ALL	1990	1991	ALL
Chevy	87	0	87	0	0	0	5	0	5	92	0	92
Ford	99	7	106	64	0	64	0	8	8	163	15	178
ALL	186	7	193	64	0	64	5	8	13	255	15	270

(22)

See how the 36 entries of the cube of Fig. 3 have been rearranged in a 3*12 rectangular layout, as dictated by the cardinality of the (dimension) attributes. Thus we are lead to the following linear algebra definition of the *generic* data cube operator.

Definition 3.1 (Cube). Let M be a matrix of type

$$\Pi_{j=1}^{n} B_j \xleftarrow{M} \Pi_{i=1}^{m} A_i \tag{23}$$

We define matrix cube M, the *cube of* M, as follows:

$$\text{cube}\ M \quad = \quad (\bigotimes_{j=1}^{n} \tau_{B_j}) \cdot M \cdot (\bigotimes_{i=1}^{m} \tau_{A_i})^{\circ} \tag{24}$$

So cube M has type $\Pi_{j=1}^{n}(B_j + 1) \longleftarrow \Pi_{i=1}^{m}(A_i + 1)$.
$\square$

The case of figures 1 and 3 corresponds to $m = 0$ in (23,24).

Note that, for $M : A \to B$, cube $M : (A + 1) \to (B + 1)$ has the type of a *cross-tabulation* [11]. Indeed, cross-tabulations are special cases of generalized cubes, as we shall see later.

Definition 3.1 is far simpler and amenable to calculations than the one given in [11]. Indeed, the perspective given in the sequel is in a sense dual of that in [11]: instead of using *cross-tabulation* as building-block, we define the *cube* as the basis for obtaining cross-tabulations and all other forms of data aggregation.

4 PROPERTIES OF DATA CUBING

We start from a very simple but relevant result. Suppose cube M is yesterday's data cube and that matrix N records today's data.

[2]Matrices of higher-rank such as M above are normally known as *tensors* [23].

By the end of the day we can calculate the whole cube $(M+N)$ by simply adding yesterday's cube with today's cube, thanks to the following theorem.

Theorem 4.1 (Linearity). *The data cube operator is linear:*

$$\mathsf{cube}\ (M+N) \quad = \quad \mathsf{cube}\ M + \mathsf{cube}\ N \tag{25}$$

Proof: *Immediate by bilinearity of matrix composition:*

$$M \cdot (N+P) \quad = \quad M\cdot N + M \cdot P \tag{26}$$

$$(N+P)\cdot M \quad = \quad N\cdot M + P\cdot M \tag{27}$$

□

Thus data cubing is additive and this can be taken advantage of not only in incremental data cube construction but also in parallelizing data cube generation.

The next result expresses an interesting property relating cubes with vectorization. If we compare the cube of Fig. 3 with that displayed in (22), we can observe that the former has the same type as that of a double vectorization of the latter, first over type *Color* and then over type *Year*.

This observation instantiates a general result: any vectorization of a cube is still a cube. This fact is captured by the theorem which follows.

Theorem 4.2 (Cube commutes with vectorization). *Let matrix* $X \xleftarrow{M} Y\times C$ *be given and* $Y\times X \xleftarrow{\mathsf{vec}\,M} C$ *be its* Y*-vectorization. Then*

$$\mathsf{vec}\,(\mathsf{cube}\ M) = \mathsf{cube}\ (\mathsf{vec}\,M) \tag{28}$$

holds.

□

To prove this theorem we need to recall a number of properties of matrix vectorization [10]: the so-called *fusion*-law

$$(\mathsf{vec}\,M)\cdot N \quad = \quad \mathsf{vec}\,(M\cdot(id\otimes N)) \tag{29}$$

and the *absorption*-law:

$$\mathsf{vec}\,(M\cdot N) \quad = \quad (id\otimes M)\cdot \mathsf{vec}\,N \tag{30}$$

We also need a property which generalizes a result by Magnus and Neudecker [12] concerning what they call the *commutation matrix*. Let $\mathsf{xng} : A\times(B\times C) \to B\times(A\times C)$ be the matrix that encodes the (homonym) isomorphism

$$\mathsf{xng}\ (a,(b,c)) = (b,(a,c)) \tag{31}$$

Then

$$\mathsf{vec}\,(M^\circ \otimes id) = \mathsf{xng}\cdot \mathsf{vec}\,(M\otimes id) \tag{32}$$

holds. For $M = id$ we obtain the corollary

$$\eta = \mathsf{xng}\cdot\eta \tag{33}$$

where η abbreviates $\mathsf{vec}\ id$. From the *absorption*-law (30) we also get corollary:

$$\mathsf{vec}\,B \quad = \quad (id\otimes B)\cdot\eta \tag{34}$$

The proof of (32) is given in the appendix as illustration of typed linear algebra *pointwise* calculation. To prove Theorem 4.2 we need yet another property of matrix vectorization:[3]

$$\mathsf{vec}\,(M\cdot(N\otimes id)) \quad = \quad (N^\circ\otimes id)\cdot\mathsf{vec}\,M \tag{35}$$

[3]This property is a generalization of $\mathsf{vec}\,(M\cdot N) = (N^\circ \otimes id)\cdot \mathsf{vec}\,M$ from which the Roth's relationship [22] is derived in [10].

The proof of (35) is given in the appendix as illustration of standard typed linear algebra *pointfree* calculation.

As preparation for the proof of Theorem 4.2, let us start by drawing a diagram exhibiting the types involved in equality (28):

$$\begin{tikzcd}
Y\times X \arrow[d, "\tau_Y\otimes\tau_M"'] & C \arrow[l, "\mathbf{vec}_Y\ M"'] & \cong & X \arrow[d, "\tau_X"'] & Y\times C \arrow[l, "M"'] \\
(Y+1)\times(X+1) & C+1 \arrow[u, "\tau_C^\circ"'] \arrow[l, shift right, "\mathbf{cube}\,(\mathbf{vec}_Y\ M)"'] \arrow[l, shift left, "\mathbf{vec}_{Y+1}\,(\mathsf{cube}\ M)"] & \cong & X+1 & (Y+1)\times(C+1) \arrow[l, "\mathbf{cube}\ M"'] \arrow[u, "(\tau_Y\otimes\tau_C)^\circ"']
\end{tikzcd}$$

Then we calculate the equality itself, with subscripts helping to track the types at each stage:

$$\begin{array}{ll}
& \mathbf{vec}_{Y+1}\,(\mathsf{cube}\ M) \\
= & \quad \{\ \text{definition of cube (24)}\ \} \\
& \mathbf{vec}_{Y+1}\,(\tau_X\cdot M\cdot(\tau_Y\otimes\tau_C)^\circ) \\
= & \quad \{\ \text{absorption (30) ; converses}\ \} \\
& (id_{Y+1}\otimes\tau_X)\cdot\mathbf{vec}_{Y+1}\,(M\cdot(\tau_Y^\circ\otimes\tau_C^\circ)) \\
= & \quad \{\ \text{Kronecker is a bifunctor [10] ; identity}\ \} \\
& (id_{Y+1}\otimes\tau_X)\cdot\mathbf{vec}_{Y+1}\,(M\cdot(\tau_Y^\circ\otimes id_C)\cdot(id_{Y+1}\otimes\tau_C^\circ)) \\
= & \quad \{\ \text{fusion (29) from right to left}\ \} \\
& (id_{Y+1}\otimes\tau_X)\cdot\mathbf{vec}_{Y+1}\,(M\cdot(\tau_Y^\circ\otimes id_C))\cdot\tau_C^\circ \\
= & \quad \{\ \text{(35), for } N=\tau_Y\ \} \\
& (id_{Y+1}\otimes\tau_X)\cdot(\tau_Y\otimes id_X)\cdot\mathbf{vec}_Y\,M\cdot\tau_C^\circ \\
= & \quad \{\ \text{Kronecker bifunctor again}\ \} \\
& (\tau_Y\otimes\tau_X)\cdot\mathbf{vec}_Y\,M\cdot\tau_C^\circ \\
= & \quad \{\ \text{definition of cube (24)}\ \} \\
& \mathsf{cube}\ (\mathbf{vec}_Y\,M)
\end{array}$$

□

The following theorem has to do with changing the dimensions of a data cube.

Theorem 4.3 (Free theorem). *Let matrix* $B \xleftarrow{M} A$ *be cubed into* $B+1 \xleftarrow{\mathsf{cube}\ M} A+1$ *, and let* $r : C \to A$ *and* $s : D \to B$ *be arbitrary functions. Then pre and post composing* cube M *by* $r\oplus id$ *and* $s^\circ \oplus id$*, respectively, yields a cube. Moreover,*

$$\mathsf{cube}\ (s^\circ\cdot M\cdot r) \quad = \quad (s^\circ\oplus id)\cdot(\mathsf{cube}\ M)\cdot(r\oplus id) \tag{36}$$

holds.

Proof: *Matrices are in 1-to-1 correspondence with binary functions on some suitable, fixed semiring. First we convert* M *into one such function and* **cube** *to the corresponding higher-order polymorphic function. Then we calculate its free theorem [24], instantiate it to functions and map the result back to matrices. (The technical details can be found in the appendix.)*

□

In general, free theorems of this kind essentially express how linear operations commute with changes to the base vectors.

5 MANIPULATING CUBES

This section shows how to select data from a data cube so as to obtain from it the other forms of data inspection/aggregation. The simplest of all these is the *slicing* operation.

Slicing. Slicing is a specialized filter for a particular value in a dimension. Starting with a simple example, suppose that from the cube of Fig. 3 one is only interested in data concerning year 1991. We recall the general (categorial) notion of a *point*: let $p \in A$ be given, for A non-empty. The constant function $\underline{p} : 1 \rightarrow A$ is said to be a *point* of A. So $\underline{p}^\circ : A \rightarrow 1$ acts as a selector, reducing *dimension* A to the single value determined by p. For instance,

$$\underline{1991} : 1 \rightarrow Year + 1$$

$$\underline{1991} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

that is, $\underline{1991}^\circ = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}$. Now suppose we wish to obtain the 1991-slice of cube $c : 1 \rightarrow (Year + 1) \times ((Color + 1) \times (Model + 1))$ given by (17) and depicted in Fig. 3. Since the year dimension will be cancelled, the type of the desired slice s will be $1 \rightarrow ((Color + 1) \times (Model + 1))$. Then post-composing c with matrix

$$\begin{tikzcd} (Year + 1) \times ((Color + 1) \times (Model + 1)) \arrow[d, "\underline{1991}^\circ \otimes id"'] \\ 1 \times ((Color + 1) \times (Model + 1)) \end{tikzcd}$$

will perform the intended slicing, cf:

$$s = (\underline{1991}^\circ \otimes id) \cdot c = \begin{bmatrix} 0 \\ 7 \\ 7 \\ 0 \\ 0 \\ 0 \\ 0 \\ 8 \\ 8 \\ 0 \\ 15 \\ 15 \end{bmatrix}$$

since the types X and $1 \times X$ contain the same information (in categories of finite matrices, they are the same). Clearly, this retrieves the 1991-block of the original cube depicted in Fig. 3.

Consolidation. Recall fact (8) about the "bang" function (!). Clearly, $! \cdot M \cdot !^\circ$ is a scalar (i.e. a matrix of type $1 \rightarrow 1$) that yields the sum of all entries in matrix M. Let us define

$$tot\ M = ! \cdot M \cdot !^\circ$$

We say a matrix $\Phi\ M$ is a *consolidation* of M, or that transformation Φ is a data *consolidation operator* wherever *tot* is an *invariant* for Φ, that is,

$$tot\ (\Phi\ M) = tot\ M$$

It is easy to see that

$$\Phi_{f,g}\ M = f \cdot M \cdot g^\circ \qquad (37)$$

is, for any functions f and g, a data consolidation — this follows immediately from (8). In the (degenerate) case $\Phi_{id,id} = M$ no consolidation actually takes place. At the other extreme, $\Phi_{!,!} = tot\ M$, which loses all information apart from that given by the invariant (total).

In presence of multidimensional data, the standard projections

$$\mathsf{fst}\ (a, b) = a \quad , \quad \mathsf{snd}\ (a, b) = b \qquad (38)$$

(or combinations thereof, in case of nested pairs) are immediate candidates for data consolidation. Think of some matrix $M : A \times B \leftarrow X$ capturing the values of some measure of a 3-dimensional data set.[4] Now suppose we want to summarize the same information across dimensions A and X only, therefore hiding all details concerning dimension B. Looking at types only, consolidated matrix $\Phi_{\mathsf{fst},id}\ M = \mathsf{fst} \cdot M$ of type $A \leftarrow X$ is a candidate for this operation and indeed its meaning is precisely what we want,

$$a\ (\mathsf{fst} \cdot M)\ x \quad = \quad \langle \textstyle\sum\ b\ ::\ (a, b)\ M\ x \rangle$$

cf.:

$$\begin{array}{ll} & a\ (\mathsf{fst} \cdot M)\ x \\ = & \qquad \{ \text{ matrix composition (6) } \} \\ & \langle \sum\ a', b\ ::\ (a\ \mathsf{fst}\ (a', b)) \times ((a', b)\ M\ x) \rangle \\ = & \qquad \{ \ a\ \mathsf{fst}\ (a', b) \ \Leftrightarrow\ a = \mathsf{fst}\ (a', b)\ \Leftrightarrow\ a = a'\ (38) \ \} \\ & \langle \sum\ a', b\ :\ a' = a\ :\ (a', b)\ M\ x \rangle \\ = & \qquad \{ \text{ "one-point" rule [11] } \} \\ & \langle \sum\ b\ ::\ (a, b)\ M\ x \rangle \end{array}$$

□

Similarly:

$$b\ (\mathsf{snd} \cdot M)\ x \quad = \quad \langle \textstyle\sum\ a\ ::\ (a, b)\ M\ x \rangle$$

This extends to higher dimensions. For instance, cancelling out dimension B from $A \times (B \times C)$ will be achieved by projection function $id \otimes \mathsf{snd} : A \times (B \times C) \rightarrow A \times C$, and so on. So, looking at the types suffices.

Cross tabulations. A cross tabulation is a consolidation enriched will totals,

$$\mathsf{ctab}_{f,g}\ M = \tau \cdot (\Phi_{f,g}\ M) \cdot \tau^\circ \qquad (39)$$

recall (9). Typewise:

$$\begin{tikzcd} Y + 1 & Y \arrow[l, "\tau_Y"'] & B \arrow[l, "f"'] & A \arrow[l, "M"'] \arrow[r, "g"] & X \arrow[r, "\tau_X"] & X + 1 \arrow[lllll, bend left, "\mathsf{ctab}_{f,g}\ M"] \end{tikzcd}$$

From (37) and (13) we obtain

$$\mathsf{ctab}_{f,g}\ M = \left[\frac{f}{!}\right] \cdot M \cdot \left[\frac{g}{!}\right]^\circ$$

as in [10].

Cross-tabulations are usually two-dimensional, cf. X and Y above. Generalizing the definition above to the case where Y and X are Cartesian products, a cross-tabulation becomes

$$\mathsf{ctab}_{f,g}\ M = \mathsf{cube}\ (\Phi_{f,g}\ M) \qquad (40)$$

In words: *a (generic) cross-tabulation is the cube of a data consolidation.*

Now the question is: can any cross-tabulation on two particular dimensions of a data cube — like that pictured in Fig. 3 – be retrieved from the whole cube itself?

[4] With no loss of generality, we gain in economy of presentation by supposing that there are only 3 dimensions involved in matrix M.

As earlier on, we look at the types first to check how to proceed. Let matrix $M : A \times B \leftarrow C$ be given in the first place, from which its cube was derived, cube $M = (\tau_A \otimes \tau_B) \cdot M \cdot \tau_C^\circ$, which has type $(A + 1) \times (B + 1) \leftarrow (C + 1)$.

Suppose we want to extract cross tabulation

$$\mathsf{ctab}_{\mathsf{fst},\,id}\ M : A + 1 \leftarrow C + 1$$

from cube M, that is, we aim at defining ψ such that

$$\mathsf{ctab}_{\mathsf{fst},\,id}\ M = \psi\ (\mathsf{cube}\ M).$$

The types in

$$A + 1 \xleftarrow{\mathsf{fst}} (A + 1) \times (B + 1) \xleftarrow{\mathsf{cube}\ M} C + 1$$

hint that $\mathsf{fst} \cdot (\mathsf{cube}\ M)$ might yield the cross-tabulation between types A and C, of type $A + 1 \rightarrow C + 1$. We reason:

$$\begin{array}{cl}
& \mathsf{fst} \cdot (\mathsf{cube}\ M) \\
= & \quad \{\ (24)\ \} \\
& \mathsf{fst} \cdot (\tau_A \otimes \tau_B) \cdot M \cdot \tau_C^\circ \\
= & \quad \{\ \text{temporarily assume: } \mathsf{fst} \cdot (\tau_A \otimes \tau_B) = \tau_A \cdot \mathsf{fst}\ \} \\
& \tau_A \cdot \mathsf{fst} \cdot M \cdot \tau_C^\circ \\
= & \quad \{\ (24)\ \} \\
& \mathsf{cube}\ (\mathsf{fst} \cdot M) \\
= & \quad \{\ (40)\ \} \\
& \mathsf{ctab}_{\mathsf{fst},\,id}\ M
\end{array}$$

□

However, things are not so immediate since the rules that cancel totalizers via the projections,

$$\mathsf{fst} \cdot (\tau_A \otimes \tau_B) = 2\ (\tau_A \cdot \mathsf{fst}) \tag{41}$$

$$\mathsf{snd} \cdot (\tau_A \otimes \tau_B) = 2\ (\tau_B \cdot \mathsf{snd}) \tag{42}$$

(proofs in the appendix) are not quite what we assumed above. So the overall outcome has to be halved. To see why this doubling side-effect takes place, suppose we want to discard dimension *Color* in the cube of Fig. 3. Selecting columns *Year* + 1 and *Model* + 1 alone does not work because we are left will spurious lines — those corresponding to the ALL entries of type *Color* + 1.

A better, alternative procedure is the following, illustrated over the cube of Fig. 3. The main idea is to cancel the totals of the dimensions we wish to abstract from *before* projections actually take place, by using cancellation property (14). Example, cancelling dimension *Color*:

$$\begin{tikzcd}
(Year + 1) \times ((Color + 1) \times (Model + 1)) \arrow[d, "id \otimes (\pi_1 \otimes id)"'] & 1 \arrow[l, "\mathsf{cube}\ v"'] \\
(Year + 1) \times (Color \times (Model + 1)) \arrow[d, "id \otimes \mathsf{snd}"'] & \\
(Year + 1) \times (Model + 1) &
\end{tikzcd}$$

Checking:

$$\begin{array}{cl}
& (id \otimes \mathsf{snd}) \cdot (id \otimes (\pi_1 \otimes id)) \cdot \mathsf{cube}\ v \\
= & \quad \{\ (24)\ ;\ \otimes \text{ is a bifunctor [10]}\ \} \\
& (id \otimes \mathsf{snd} \cdot (\pi_1 \otimes id)) \cdot (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot v \\
= & \quad \{\ \text{bifunctor } \otimes\ ;\ \text{identity}\ ;\ \text{cancellation (14)}\ \} \\
& (\tau_{Year} \otimes \mathsf{snd} \cdot (id \otimes \tau_{Model})) \cdot v \\
= & \quad \{\ \mathsf{snd} \cdot (id \otimes M) = M \cdot \mathsf{snd}\ \} \\
& (\tau_{Year} \otimes \tau_{Model} \cdot \mathsf{snd}) \cdot v \\
= & \quad \{\ \text{functor} \cdot \otimes \cdot\ ;\ (24)\ \} \\
& \mathsf{cube}\ ((id \otimes \mathsf{snd}) \cdot v) \\
= & \quad \{\ (39)\ \} \\
& \mathsf{ctab}_{id \otimes \mathsf{snd},\,id}\ v
\end{array}$$

□

Roll-up. Another data summarization operator is *roll-up*, an operator that has multiple, different semantic interpretations in the literature. The interpretation we formalize below is that of [7].

Suppose we have 3-dimensional data with attributes A, B and C. A roll-up operation over these dimensions is the following form of summarization

$$\begin{array}{l} A \times (B \times C) \\ A \times B \\ A \\ 1 \end{array}$$

where we see that each step loses one attribute until there is none. For only two dimensions this reduces to

$$\begin{array}{l} A \times B \\ A \\ 1 \end{array} \tag{43}$$

and so on. Below we illustrate roll-up for this two dimensional case, thus saving some space concerning some matrices we wish to depict.

Above we said that the "output" type of a cube, say $(A+1)\times(B+1)$, is isomorphic to $A \times B + A + B + 1$, therefore representing (typewise) the four elements of the dimension powerset of $\{A, B\}$. This can be made formal by defining the injections of each such element in $A \times B + A + B + 1$ into $(A + 1) \times (B + 1)$,

$$\begin{tikzcd}
 & & (A + 1) \times (B + 1) & & \\
A \times B \arrow[urr, "\theta"] & & A \arrow[u, "\alpha"] & B \arrow[ul, "\beta"] & 1 \arrow[ull, "\omega"']
\end{tikzcd}$$

where

$$\begin{array}{l}
\theta = i_1 \otimes i_1 \\
\alpha = i_1 \triangledown i_2 \cdot ! \\
\beta = i_1 \cdot ! \triangledown i_2 \\
\omega = i_2 \triangledown i_2
\end{array}$$

Moreover, the following properties hold (proofs in the appendix):

$$\theta^\circ \cdot (\tau_A \otimes \tau_B) = id \tag{44}$$

$$\alpha^\circ \cdot (\tau_A \otimes \tau_B) = \mathsf{fst} \tag{45}$$

$$\beta^\circ \cdot (\tau_A \otimes \tau_B) = \mathsf{snd} \tag{46}$$

$$\omega^\circ \cdot (\tau_A \otimes \tau_B) = \,! \tag{47}$$

From these we may build compound injections, for instance

$$\delta : (A+1) \times (B+1) \leftarrow A \times B + 1$$

$$\delta = \left[\begin{array}{c|c} \theta & \omega \end{array}\right] \tag{48}$$

Interestingly, δ acts as mediator between $\tau_{A\times B}$ — the totalizer of "compound" attribute $A \times B$ — and $\tau_A \otimes \tau_B$ — the compound of the two totalizations of A and B:

$$\delta^\circ \cdot (\tau_A \otimes \tau_B) = \tau_{A\times B} \qquad \begin{tikzcd} (A+1)\times(B+1) \arrow[d, "\delta^\circ"'] & A \times B \arrow[l, "\tau_A \otimes \tau_B"'] \arrow[dl, "\tau_{A\times B}"] \\ A \times B + 1 & \end{tikzcd}$$

This follows immediately from the properties of the injections involved:

$$\delta^\circ \cdot (\tau_A \otimes \tau_B) = \tau_{A\times B}$$

$\equiv$ { definition of δ (48) ; (10) ; (9) }

$$\left[\begin{array}{c} \theta^\circ \\ \hline \omega^\circ \end{array}\right] \cdot (\tau_A \otimes \tau_B) = \left[\begin{array}{c} id \\ \hline ! \end{array}\right]$$

$\equiv$ { fusion (12) }

$$\left[\begin{array}{c} \theta^\circ \cdot (\tau_A \otimes \tau_B) \\ \hline \omega^\circ \cdot (\tau_A \otimes \tau_B) \end{array}\right] = \left[\begin{array}{c} id \\ \hline ! \end{array}\right]$$

$\equiv$ { (44) and (47) ; structural equality }

true

□

The combination of injections which implements the roll-up of (43) is given by

$$\rho : (A+1) \times (B+1) \leftarrow A \times B + (A+1)$$

$$\left[\begin{array}{c|c} \theta & \left[\begin{array}{c|c} \alpha & \omega \end{array}\right] \end{array}\right] \tag{49}$$

Then, for $M : C \to A \times B$:

$$\rho^\circ \cdot (\mathrm{cube}\ M)$$

$=$ { definition of ρ above; (10) twice }

$$\left[\begin{array}{c} \theta^\circ \\ \hline \left[\begin{array}{c} \alpha^\circ \\ \hline \omega^\circ \end{array}\right] \end{array}\right] \cdot (\tau_A \otimes \tau_B) \cdot M \cdot \tau_C^\circ$$

$=$ { fusion (12) twice }

$$\left[\begin{array}{c} \theta^\circ \cdot (\tau_A \otimes \tau_B) \\ \hline \left[\begin{array}{c} \alpha^\circ \cdot (\tau_A \otimes \tau_B) \\ \hline \omega^\circ \cdot (\tau_A \otimes \tau_B) \end{array}\right] \end{array}\right] \cdot M \cdot \tau_C^\circ$$

$=$ { (44); (45); (47); fusion again }

$$\left[\begin{array}{c} M \\ \hline \left[\begin{array}{c} \mathsf{fst} \cdot M \\ \hline ! \cdot M \end{array}\right] \end{array}\right] \cdot \tau_C^\circ$$

extracts from cube M the corresponding *roll-up*. Taking as starting point the following (generalized) cube of matrix (20),

		1990	1991	ALL
	Chevy	87	0	87
Blue	*Ford*	99	7	106
	ALL	186	7	193
	Chevy	0	0	0
Green	*Ford*	64	0	64
	ALL	64	0	64
	Chevy	5	0	5
Red	*Ford*	0	8	8
	ALL	5	8	13
	Chevy	92	0	92
ALL	*Ford*	163	15	178
	ALL	255	15	270

matrix ρ° of type $(Color+1)\times(Model+1) \to Color\times Model+Color+1$ will be

		Blue			*Green*			*Red*			ALL		
		Chevy	*Ford*	ALL	*Chevy*	*Ford*	ALL	*Chevy*	*Ford*	ALL	*Chevy*	*Ford*	ALL
Blue	*Chevy*	1	0	0	0	0	0	0	0	0	0	0	0
	Ford	0	1	0	0	0	0	0	0	0	0	0	0
Green	*Chevy*	0	0	0	1	0	0	0	0	0	0	0	0
	Ford	0	0	0	0	1	0	0	0	0	0	0	0
Red	*Chevy*	0	0	0	0	0	0	1	0	0	0	0	0
	Ford	0	0	0	0	0	0	0	1	0	0	0	0
	Blue	0	0	1	0	0	0	0	0	0	0	0	0
	Green	0	0	0	0	0	1	0	0	0	0	0	0
	Red	0	0	0	0	0	0	0	0	1	0	0	0
	ALL	0	0	0	0	0	0	0	0	0	0	0	1

that, post-composed with the cube, will yield:

		1990	1991	ALL
Blue	*Chevy*	87	0	87
	Ford	99	7	106
Green	*Chevy*	0	0	0
	Ford	64	0	64
Red	*Chevy*	5	0	5
	Ford	0	8	8
	Blue	186	7	193
	Green	64	0	64
	Red	5	8	13
	ALL	255	15	270

As the matrices illustrate, a roll-up is a particular "subset" of a cube. Boolean matrix ρ° performs the (quantitative) selection of such a subset.

6 CONCLUSIONS AND FUTURE WORK

Typed linear algebra offers a polymorphic, strongly typed approach to OLAP semantics and data analysis. Previous work by the same authors [11] derived the *data cube* construction from that of *cross tabulation*. This paper exploits the alternative view of regarding data cubing as a *primitive construction* in data analysis, wherefrom the other 2D, 1D and 0D aggregators are derived.

The central difference between [11] and the current approach is that the cube operator proposed here is type-generic when compared to the same operator in [11]. Such a generalization makes it easier to derive some results, for instance the commutation between data-cubing and vectorization (Theorem 4.2). Interestingly, the proof also calls for a generalization of the so-called *Roth's relationship* [22], a standard result that is regarded as the basis of the whole theory of vectorization [2].[5]

[5]Practically speaking, we claim that such a generalized cube operator is *universal* for classical data aggregation operations in data warehouse systems. Proving this *formally* is a topic for future research.

On the practical side, rooting OLAP querying on linear algebra will lead to linear-algebra based optimization of data analysis processes in a natural way. Theorem 4.1 already is one such optimization, saving the re-building of a whole data cube when new data arrives, by just cubing up what is new and adding this to the old cube. This extends to other "CRUDE" operations, e.g. record updating, deletion and so on.

This work can be pushed further in several other directions. One is that of exploiting the parallelism inherent in linear algebra (LA) processing to implement data cubing in a more efficient way. Preliminary results [17, 20] are showing LA scripts encoding data analysis operations performing better on HPC architectures than the standard competitors.[6]

Another direction has to do with the formal semantics of data-analytical operations started in [11], which we want to develop further envisaging another example of advantageous usage of linear algebra as a computing model for software components [9]. The typed LA approach offers a very simple semantics to OLAP when compared to e.g. [5, 6] but its strongly typed basis does not let properties such as e.g. "Gray's axioms" [7],

$$\begin{array}{l} CUBE\ (ROLLUP) = CUBE \\ ROLLUP\ (GROUP\ BY) = ROLLUP \end{array}$$

to be expressed as such. (Although not formally proved in [7], these intuitively make sense in an untyped, set-theoretical model.) References [17, 20] show how to encode SQL into LA scripts in a way similar to the approach of columnar database systems [1]. We would like to extend this to data analysis languages such as e.g. Microsoft's MDX.[7]

Finally, through linear algebra one might possibly gain typed *linear logic* as formal language for data analysis. Linear logic is a *resource-aware* kind of logic that captures quantification in a subtle way. The formal connection between logic and linear algebra presented in e.g. [13] could be taken as starting point for such a prospective work.

Acknowledgements. The authors would like to thank the insightful comments and suggestions provided by the reviewers. The research reported in this paper has received funding from the European Union's Horizon 2020 - The EU Framework Programme for Research and Innovation 2014-2020, under grant agreement No. 732051 (CloudDBAppliance project).

REFERENCES

[1] D. Abadi, P. Boncz, S. Harizopoulos, S. Idreos, and S. Madden. 2012. The Design and Implementation of Modern Column-Oriented Database Systems. *Foundations and Trends in Databases* 5, 3 (2012), 197–280.
[2] K.M. Abadir and J.R. Magnus. 2005. *Matrix algebra. Econometric exercises 1.* C.U.P.
[3] R.C. Backhouse and D. Michaelis. 2006. Exercises in Quantifier Manipulation. In *MPC'06*, T. Uustalu (Ed.). LNCS, Vol. 4014. Springer-Verlag, 70–81.
[4] R. Bird and O. de Moor. 1997. *Algebra of Programming.* Prentice-Hall.
[5] G. Bültzingsloewen. 1987. Translating and Optimizing SQL Queries Having Aggregates. In *VLDB'87.* 235–243.
[6] A. Datta and H. Thomas. 1999. The cube data model: a conceptual model and algebra for on-line analytical processing in data warehouses. *Decis. Support Syst.* 27, 3 (1999), 289–301.
[7] Jim Gray, Surajit Chaudhuri, Adam Bosworth, Andrew Layman, Don Reichart, Murali Venkatrao, Frank Pellow, and Hamid Pirahesh. 1997. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. *J. Data Mining and Knowledge Discovery* 1, 1 (1997), 29–53. citeseer.nj.nec.com/article/gray95data.html
[8] H. Macedo. 2012. *Matrices as Arrows — Why Categories of Matrices Matter.* Ph.D. Dissertation. University of Minho. MAPi PhD programme.
[9] H.D. Macedo and J.N. Oliveira. 2012. Towards Linear Algebras of Components. In *FACS'10 (LNCS)*, Vol. 6921. Springer-Verlag, 300–303.
[10] H.D. Macedo and J.N. Oliveira. 2013. Typing linear algebra: A biproduct-oriented approach. *SCP* 78, 11 (2013), 2160–2191.
[11] H.D. Macedo and J.N. Oliveira. 2015. A linear algebra approach to OLAP. *FAoC* 27, 2 (2015), 283–307.
[12] Jan Magnus and H. Neudecker. 1979. The Commutation Matrix: Some Properties and Applications. *The Annals of Statistics* 7, 2 (1979), 381–394.
[13] D. Murfet. 2014. Logic and linear algebra: an introduction. *CoRR* abs/1407.2650 (2014), 1–42. http://arxiv.org/abs/1407.2650
[14] D. Murta and J.N. Oliveira. 2015. A study of risk-aware program transformation. *SCP* 110 (2015), 51–77.
[15] J.N. Oliveira. 2012. Towards a linear algebra of programming. *FAoC* 24, 4-6 (2012), 433–458.
[16] J.N. Oliveira. 2013. Weighted Automata as Coalgebras in Categories of Matrices. *Int. JFCS* 24, 06 (2013), 709–728.
[17] J.N. Oliveira. 2016. Towards a linear algebra semantics for query languages. (June 2016). Presented at IFIP WG 2.1 #74 Meeting, U. Strathclyde, Glasgow, 13-17 June (slides available from the WG's website.).
[18] J.N. Oliveira and V.C. Miraldo. 2016. "Keep definition, change category" — a practical approach to state-based system calculi. *JLAMP* 85(4) (2016), 449–474.
[19] T.B. Pedersen and C.S. Jensen. 2001. Multidimensional Database Technology. *Computer* 34 (December 2001), 40–46. Issue 12. http://dx.doi.org/10.1109/2.970558
[20] R. Pontes, M. Matos, J.N. Oliveira, and J.O. Pereira. 2017. Implementing a Linear Algebra Approach to Data Processing. In *GTTSE 2015 (LNCS)*, Vol. 10223. Springer-Verlag, 215–222.
[21] C.R. Rao and M.B. Rao. 1998. *Matrix algebra and its applications to statistics and econometrics.* World Scientific Pub Co Inc.
[22] W. E. Roth. 1934. On direct product matrices. Bulletin of the American Mathematical Society, 40 (1934), 461–468.
[23] J. Sun, D. Tao, S. Papadimitriou, P.S. Yu, and C. Faloutsos. 2008. Incremental tensor analysis: Theory and applications. *ACM Trans. Knowl. Discov. Data* 2, Article 11 (October 2008), 37 pages. Issue 3.
[24] P.L. Wadler. 1989. Theorems for Free!. In *4th International Symposium on Functional Programming Languages and Computer Architecture.* ACM, London, 347–359.

A APPENDIX

The following rules interfacing index-free and index-wise matrix notation, where f and g functional matrices, are used in this paper:

$$y\,(g^\circ \cdot M \cdot f)\,x \quad = \quad (g\ y)\,M\,(f\ x) \tag{50}$$

$$y\,(f \cdot M)\,x \quad = \quad \langle \sum\ z\ :\ y = f\ z\ :\ z\,M\,x \rangle \tag{51}$$

$$y\,(M \cdot f^\circ)\,x \quad = \quad \langle \sum\ z\ :\ x = f\ z\ :\ y\,M\,z \rangle \tag{52}$$

These rules are expressed in the style of the Eindhoven quantifier calculus (see e.g. [3]) and are convenient shorthands for the corresponding instances of matrix composition (6).

The rest of this appendix includes the proofs of the main results presented in this paper.

Proof of (32). Let us first draw a diagram depicting the types involved in (32):

$$\begin{tikzcd} B \times (A \times C) & A \times (B \times C) \arrow[l, "\mathsf{xng}"'] & & B \\ & C \arrow[ul, "\mathbf{vec}_B\,(M^\circ \otimes id_C)"] \arrow[u, "\mathbf{vec}_A\,(M \otimes id_C)"'] & & A \arrow[u, "M"'] \end{tikzcd}$$

The types of variables a, b, c and c' introduced below to both sides of the equality should be obvious from the diagram. We calculate the left hand

[6] Evidence of this is growing from on-going experimentation with TPC-H benchmarking (http://www.tpc.org/tpch/default.asp) [20].
[7] https://msdn.microsoft.com/en-us/library/ms144884.aspx.

side first:

$$
\begin{array}{ll}
& (b, (a, c'))\ (\mathrm{vec}\,(M^\circ \otimes id))\ c \\
= & \quad \{\ \text{vectorization (18)}\ \} \\
& (a, c')\ (M^\circ \otimes id)\ (b, c) \\
= & \quad \{\ \text{Kronecker product (4)}\ \} \\
& (a\ M^\circ\ b) \times (c'\ id\ c) \\
= & \quad \{\ \text{converse ; Kronecker product}\ \} \\
& (b, c')\ (M \otimes id)\ (a, c)
\end{array}
$$

Next we calculate the right hand side:

$$
\begin{array}{ll}
& (b, (a, c'))\ (\mathrm{xng} \cdot \mathrm{vec}\,(M \otimes id))\ c \\
= & \quad \{\ \text{rule (51)}\ \} \\
& \langle \sum x\ :\ (b, (a, c')) = \mathrm{xng}\ x :\ x\ \mathrm{vec}\,(M \otimes id)\ c \rangle \\
= & \quad \{\ \text{isomorphism xng (31)}\ \} \\
& \langle \sum x\ :\ x = (a, (b, c')) :\ x\ \mathrm{vec}\,(M \otimes id)\ c \rangle \\
= & \quad \{\ \text{one point rule of quantification}\ \} \\
& (a, (b, c')))\ (\mathrm{vec}\,(M \otimes id))\ c \\
= & \quad \{\ \text{vectorization (18)}\ \} \\
& (b, c')\ (M \otimes id)\ (a, c)
\end{array}
$$

Thus the equality holds.

□

Proof of (35). As above, we start by drawing a diagram showing the generic types involved in the equality we wish to calculate:

$$\begin{tikzcd}
 & D & B \times C \arrow[l, "M"'] & A \times C \arrow[l, "N \otimes id_C"'] \\
A \times D & B \times D \arrow[l, "N^\circ \otimes id_D"'] & C \arrow[l, "\mathbf{vec}_B\, M"'] \arrow[ll, bend left, "\mathbf{vec}_A\,(M \cdot (N \otimes id_C))"] &
\end{tikzcd}$$

Then we calculate the equality in the *pointfree* typed linear algebra style:

$$
\begin{array}{ll}
& \mathbf{vec}_A\,(M \cdot (N \otimes id_D)) = (N^\circ \otimes id_D) \cdot \mathbf{vec}_B\,M \\
\equiv & \quad \{\ \text{lhs: absorption (30) ; rhs: (34)}\ \} \\
& (id_A \otimes M) \cdot \mathbf{vec}_A\,(N \otimes id_D) = (N^\circ \otimes id_D) \cdot (id_B \otimes M) \cdot \eta_B \\
\equiv & \quad \{\ \text{Kronecker is a bifunctor [10]}\ \} \\
& (id_A \otimes M) \cdot \mathbf{vec}_A\,(N \otimes id_D) = (id_A \otimes M) \cdot (N^\circ \otimes id_{B\times D}) \cdot \eta_B \\
\Leftarrow & \quad \{\ \text{Leibniz}\ \} \\
& \mathbf{vec}_A\,(N \otimes id_D) = (N^\circ \otimes id_{B\times D}) \cdot \eta_B \\
\equiv & \quad \{\ (32)\ \} \\
& \mathrm{xng} \cdot \mathbf{vec}_B\,(N^\circ \otimes id_D) = (N^\circ \otimes id_{B\times D}) \cdot \eta_B \\
\equiv & \quad \{\ \text{lhs: (34)}\ \} \\
& \mathrm{xng} \cdot (id_B \otimes (N^\circ \otimes id_D)) \cdot \eta_B = (N^\circ \otimes id_{B\times D}) \cdot \eta_B \\
\equiv & \quad \{\ \text{naturality of xng ; } id_B \otimes id_D = id_{B\times D}\ \} \\
& (N^\circ \otimes id_{B\times D}) \cdot \mathrm{xng} \cdot \eta_B = (N^\circ \otimes id_{B\times D}) \cdot \eta_B \\
\equiv & \quad \{\ (33)\ \} \\
& \mathit{true}
\end{array}
$$

□

Proof of (36). Every matrix $B \xleftarrow{M} A$ is in 1-1 correspondence with a function $\mathbb{S} \xleftarrow{k} B \times A$, for $\mathbb{S}$ a fixed semiring:

$$[\![k]\!] = M \quad \equiv \quad k(b, a) = bMa \tag{53}$$

Cancellation yields:

$$k(b, a) = b[\![k]\!]a \tag{54}$$

A consequence of (53) is the equality

$$[\![h \cdot (g \times f)]\!] \quad = \quad g^\circ \cdot [\![h]\!] \cdot f \tag{55}$$

proved below:

$$
\begin{array}{ll}
& [\![h \cdot (g \times f)]\!] = M \\
\equiv & \quad \{\ (53)\ \} \\
& h((g \times f)(b, a)) = bMa \\
\equiv & \quad \{\ \text{product of two functions}\ \} \\
& h(g\ b, f\ a) = bMa \\
\equiv & \quad \{\ (54)\ \} \\
& (g\ b)[\![h]\!](f\ a) = bMa \\
\equiv & \quad \{\ \text{rule (50)}\ \} \\
& g^\circ \cdot [\![h]\!] \cdot f = M
\end{array}
$$

Thus the cube matrix operator

$$\mathrm{cube}\ : ((B+1) \leftarrow (A+1)) \leftarrow (B \leftarrow A)$$

corresponds to the polymorphic, higher order function

$$\mathit{cube}' : (\mathbb{S} \leftarrow ((B+1) \times (A+1))) \leftarrow (\mathbb{S} \leftarrow (B \times A))$$

in the sense that

$$[\![\mathit{cube}'\ f]\!] = \mathrm{cube}\ [\![f]\!] \tag{56}$$

The free theorem [24] of *cube′* is, for functions:

$$(\mathit{cube}'\ f) \cdot ((s + id) \times (r + id)) = \mathit{cube}'\ (f \cdot (s \times r))$$

Thus

$$
\begin{array}{ll}
& [\![(\mathit{cube}'\ f) \cdot ((s + id) \times (r + id))]\!] = [\![\mathit{cube}'\ (f \cdot (s \times r))]\!] \\
\equiv & \quad \{\ (55\ ; (56)\ \} \\
& (s \oplus id)^\circ \cdot [\![\mathit{cube}'\ f]\!] \cdot (r \oplus id) = \mathrm{cube}\ [\![f \cdot (s \times r)]\!] \\
\equiv & \quad \{\ (55\ ; (56)\ \} \\
& (s \oplus id)^\circ \cdot \mathrm{cube}\ [\![f]\!] \cdot (r \oplus id) = \mathrm{cube}\ (s^\circ \cdot [\![f]\!] \cdot r) \\
\equiv & \quad \{\ \text{change of variable: } M = [\![f]\!]\ \} \\
& (s \oplus id)^\circ \cdot \mathrm{cube}\ M \cdot (r \oplus id) = \mathrm{cube}\ (s^\circ \cdot M \cdot r)
\end{array}
$$

□

Proofs of (41) and (42). Calculating (41), the calculation of (42) being similar:

$\mathsf{fst} \cdot (\tau_A \otimes \tau_B)$

$=$ { definition of fst }

$(id \otimes !) \cdot (\tau_A \otimes \tau_B)$

$=$ { bifunctor $\otimes$; $! = \left[\ ! \mid !\ \right]$; $! : 1 \to 1 = id$ }

$\tau_A \otimes \left[\ ! \mid id\ \right] \cdot \left[\frac{id}{!}\right]$

$=$ { divide & conquer rule (11) }

$\tau_A \otimes (! + !)$

$=$ { $M + M = 2\ M$ }

$\left[\frac{id}{!}\right] \otimes (2\ !)$

$=$ { $\left[\frac{A}{B}\right] \otimes C = \left[\frac{A \otimes C}{B \otimes C}\right]$; $! \otimes ! = !$ }

$\left[\frac{id \otimes 2\ !}{(2\ !)}\right]$

$=$ { $(k\ A) \otimes B = A \otimes (k\ B) = k\ (A \otimes B)$ }

$2 \left[\frac{\mathsf{fst}}{!}\right]$

$=$ { (13) }

$2\ (\tau_A \cdot \mathsf{fst})$

□

Proof of (44). :

$\theta^\circ \cdot (\tau_A \otimes \tau_B)$

$=$ { definition of θ; converses }

$(i_1 \otimes i_1)^\circ \cdot (\tau_A \otimes \tau_B)$

$=$ { biproducts [10] }

$(\pi_1 \otimes \pi_1) \cdot (\tau_A \otimes \tau_B)$

$=$ { bifunctor $\otimes$ }

$(\pi_1 \cdot \tau_A \otimes \pi_1 \cdot \tau_B)$

$=$ { (14) ; bifunctor $\otimes$ }

id

□

Proof of (45).

$\alpha^\circ \cdot (\tau_A \otimes \tau_B) = \mathsf{fst}$

$\equiv$ { definition of α }

$(i_1 \triangledown i_2 \cdot !)^\circ \cdot (\tau_A \otimes \tau_B) = \mathsf{fst}$

$\equiv$ { taking converses }

$(\tau_A^\circ \otimes \tau_B^\circ) \cdot (i_1 \triangledown i_2 \cdot !) = \mathsf{fst}^\circ$

$\equiv$ { absorption (5) ; (14) ; (15) }

$(id \triangledown !^\circ \cdot !) = \mathsf{fst}^\circ$

$\equiv$ { absorption (5) }

$(id \otimes !^\circ) \cdot (id \triangledown !) = \mathsf{fst}^\circ$

$\equiv$ { $M \triangledown ! = M = ! \triangledown M$ [11] }

$(id \otimes !^\circ) = \mathsf{fst}^\circ$

$\equiv$ { definition of fst taking converses }

true

□

Proof of (47). :

$\omega^\circ \cdot (\tau_A \otimes \tau_B) = !$

$\equiv$ { taking converses }

$(\tau_A^\circ \otimes \tau_B^\circ) \cdot \omega = !^\circ$

$\equiv$ { definition of ω ; absorption (5) }

$\tau_A^\circ \cdot i_2 \triangledown \tau_B^\circ \cdot i_2 = !^\circ$

$\equiv$ { $\tau^\circ \cdot i_2 = !^\circ$, equivalent to (15) }

$!^\circ \triangledown !^\circ = !^\circ$

$=$ { $M \triangledown ! = M = ! \triangledown M$ [11] }

true

□