The Data Cube as a Typed Linear Algebra Operator

DBPL 2017 — 16th Symp. on DB Prog. Lang.

Technische Universität München (TUM), 1st Sep 2017

(Also: 38th InfoBlender Seminar, HASLab, 27th Sep 2017)

J.N. Oliveira

INESC TEC & U.Minho

(H2020-732051: CloudDBAppliance)

H.D. Macedo

SW Eng Group @ U.Aharus

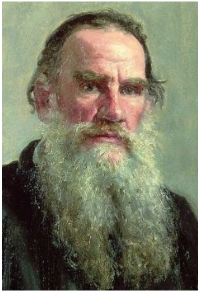
DBPL 2017


The 16th International Symposium on
Database Programming Languages
(DBPL 2017)
will be held in conjunction with VLDB 2017
on September 1st, 2017, in Munich, Germany.

Motivation

"Only by taking infinitesimally small units for observation (the ***differential*** *of history, that is, the individual tendencies of men) and attaining to the art of* ***integrating*** *them (that is, finding the sum of these infinitesimals) can we hope to arrive at the laws of history."*

Leo Tolstoy, "War and Peace"
- Book XI, Chap.II (1869)



L. Tolstoy (1828–1910)

150 years later, this is what we are trying to attain through **data-mining**.

But — how fit are our **maths** for the task?

Have we attained the "**art of integration**"?

Motivation

Since the early days of psychometrics in the **social sciences** (1970s), **linear algebra** (LA) has been central to data analysis (e.g. tensor decompositions etc)

We follow this trend but in a **typed** way, merging **LA** with polymorphic **type systems**, over a categorial basis.

We address a concrete example: that of studying the maths behind a well-known device in data analysis, the **data cube** construction.

We will define this construction as a **polymorphic LA** operator.

Typing **linear algebra** is proposed as a strategy for achieving such an “**art of integration**”.

Running example

Raw data:

$t =$

#	**Model**	**Year**	**Color**	**Sale**
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

Rows — records (n-many) — the *infinitesimals*
Columns — attributes — the *observables*

Column-orientation — each column (attribute) A represented by a function $t_A : n \to A$ such that $a = t_A\,(i)$ means "a is the value of attribute A in record nr i".

Records are tuples

Can records be rebuilt from such **attribute projection** functions?

Yes — by **tupling** them.

Tupling: *Given functions $f : A \to B$ and $g : A \to C$, their tupling is the function $f \triangledown g$ such that*

$$(f \triangledown g)\ a = (f\ a, g\ a)$$

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.

Inverting tuples

For the column-oriented model to work one will need to express *joins*, and these call for *"inverse"* functions, e.g.

$$(t_{Model} \triangledown t_{Year})^{\circ}\ (Ford, 1990) = \{3, 4\}$$

meaning that tuples nr 3 and nr 4 have the same model (*Ford*) and year (1990).

However, the type $f^{\circ} : A \to \mathcal{P}\ n$ is rather annoying, as it involves **sets** of tuple indices — these will add an extra layer of complexity.

Fortunately, there is a simpler way — **typed linear algebra**, also known as **linear algebra of programming** (**LAoP**).

The LAoP approach

Represent functions by Boolean matrices:

Given (finite) types A and B, any function

$$f : A \to B$$

can be represented by a matrix $[\![f]\!]$ with A-many columns and B-many rows such that, for any $b \in B$ and $a \in A$, the (b, a)-matrix-cell is

$$b\ [\![f]\!]\ a = \begin{cases} 1 \Leftarrow b = f\ a \\ 0\ \mathit{otherwise} \end{cases}$$

NB: 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.

The LAoP approach

One projection function (matrix) per **dimension** attribute:

t_{Model}	1	2	3	4	5	6
Chevy	1	1	0	0	0	0
Ford	0	0	1	1	1	1

t_{Year}	1	2	3	4	5	6
1990	1	1	1	1	0	0
1991	0	0	0	0	1	1

t_{Color}	1	2	3	4	5	6
Blue	0	1	0	1	0	1
Green	0	0	1	0	0	0
Red	1	0	0	0	1	0

#	**Model**	**Year**	**Color**	**Sale**
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

NB: we tend to abbreviate $[\![f]\!]$ by f when the context is clear.

The LAoP approach

Note how the inverse of a function is also represented by a Boolean matrix, e.g.

t°_{Model}	Chevy	Ford
1	1	0
2	1	0
3	0	1
4	0	1
5	0	1
6	0	1

versus

t_{Model}	1	2	3	4	5	6
Chevy	1	1	0	0	0	0
Ford	0	0	1	1	1	1

— no need for powersets.

Clearly,

$$j \; t^{\circ}_{Model} \; a = a \; t_{Model} \; j$$

Given a matrix M, M° is known as the **transposition** of M.

The LAoP approach

We **type** matrices in the same way as functions: $M : A \to B$ means a matrix M with A-many columns and B-many rows.

Matrices are arrows: $A \xrightarrow{M} B$ denotes a matrix from A (source) to B (target), where A, B are (finite) types.

Writing $B \xleftarrow{M} A$ means the same as $A \xrightarrow{M} B$.

Composition — *aka* matrix multiplication:

$$\begin{tikzcd} B & A \arrow[l, "M"'] & C \arrow[l, "N"'] \arrow[ll, bend left, "M\cdot N"] \end{tikzcd}$$

$$b(M \cdot N)c \;=\; \langle \sum a \;::\; (b\;M\;a) \times (a\;N\;c) \rangle$$

The LAoP approach

Function composition implemented by matrix multiplication, $[\![f \cdot g]\!] = [\![f]\!] \cdot [\![g]\!]$

Identity — the identity matrix *id* corresponds to the identity function and is such that

$$M \cdot id = M = id \cdot M \qquad (1)$$

Function tupling corresponds to the so-called **Khatri-Rao product** $M \triangledown N$ defined index-wise by

$$(b, c)\ (M \triangledown N)\ a \quad = \quad (b\ M\ a) \times (c\ N\ a) \qquad (2)$$

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

$$(y, x)\ (M \otimes N)\ (b, a) \quad = \quad (y\ M\ b) \times (x\ N\ a) \qquad (3)$$

Typing data

The raw data given above is represented in the LAoP by the expression

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

of type

$$v : 1 \to (Year \times (Color \times Model))$$

depicted aside.

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

v is a **multi-dimensional** column vector — a **tensor**. Datatype $1 = \{\text{ALL}\}$ is the so-called **singleton** type.

Dimensions and measures

Sale is a special kind of data — a **measure**. Measures are encoded as **row** vectors, e.g.

t^{Sale}	1	2	3	4	5	6
1	5	87	64	99	8	7

recall

#	**Model**	**Year**	**Color**	**Sale**
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

$$\begin{tikzcd} & Model & \\ Year & \#t \arrow[l, "t_{Year}"'] \arrow[u, "t_{Model}"'] \arrow[r, "t_{Color}"] \arrow[d, "t^{Sale}"] & Color \\ & 1 & \end{tikzcd}$$

Summary: **dimensions** *are* **matrices**, **measures** *are* **vectors**.

Measures provide for **integration** in Tolstoy's sense — *aka* **consolidation**

Totalisers

There is a unique function in type $A \to 1$, usually named $A \xrightarrow{!} 1$. This corresponds to a row vector wholly filled with 1s.

Example: $2 \xrightarrow{!} 1 = \begin{bmatrix} 1 & 1 \end{bmatrix}$

Given $M : B \to A$, the expression $! \cdot M$ (where $A \xrightarrow{!} 1$) is the row vector (of type $B \to 1$) that contains all column **totals** of M,

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 50 & 40 & 85 & 115 \\ 50 & 10 & 85 & 75 \end{bmatrix} = \begin{bmatrix} 100 & 50 & 170 & 190 \end{bmatrix}$$

Given type A, define its **totalizer** matrix $A \xrightarrow{\tau_A} A + 1$ by

$$\begin{aligned} \tau_A &: A \to A + 1 \\ \tau_A &= \begin{bmatrix} id \\ \hline ! \end{bmatrix} \end{aligned} \tag{5}$$

Thus $\tau_A \cdot M$ yields a copy of M on top of the corresponding totals.

Cubes

Data **cubes** are easily obtained from products of totalizers.

Recall the Kronecker (tensor) product $M \otimes N$ of two matrices:

$$\begin{array}{ccccc} A & & C & & A \times C \\ {\scriptstyle M}\downarrow & & {\scriptstyle N}\downarrow & & \downarrow{\scriptstyle M \otimes N} \\ B & & D & & B \times D \end{array}$$

The matrix

$$A \times B \xrightarrow{\tau_A \otimes \tau_B} (A+1) \times (B+1)$$

provides for totalization on the **two dimensions** A and B.

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 $\{A, B\}$".

Cube = muti-dimensional totalisation

Recalling

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

we build

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

This is the multidimensional vector (tensor) representing the **data cube** for

- **dimensions** *Year*, *Color*, *Model*
- **measure** *Sale*

depicted aside.

(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

Totalisers yield cubes

Thanks to $\times$-**absorption**

$$(M \otimes N) \cdot (P \triangledown Q) = (M \cdot P) \triangledown (N \cdot Q) \tag{6}$$

we can simplify the definition:

$$\begin{array}{rl}
c & = \ (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot v \\
& = \qquad \{ \ v = (t_{Year} \triangledown (t_{Color} \triangledown t_{Model})) \cdot (t^{Sale})^{\circ} \ \} \\
& \quad (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot (t_{Year} \triangledown (t_{Color} \triangledown t_{Model})) \cdot (t^{Sale})^{\circ} \\
& = \qquad \{ \ \textbf{absorption}\text{-law } (6) \ \} \\
& \quad ((\tau_{Year} \cdot t_{Year}) \triangledown ((\tau_{Color} \cdot t_{Color}) \triangledown ((\tau_{Model} \cdot t_{Model})))) \cdot (t^{Sale})^{\circ} \\
& = \qquad \{ \ \text{define } t'_A = \tau_A \cdot t_A \ \} \\
& \quad (t'_{Year} \triangledown (t'_{Color} \triangledown t'_{Model})) \cdot (t^{Sale})^{\circ}
\end{array}$$

Note that $t'_A = \left[\frac{t_A}{!}\right]$, since t_A is a function.

Generalizing data cubes

In our approach a **cube** is not necessarily one such column vector.

The key to **generic** data cubes is (generalized) **vectorization**, a kind of "**matrix currying**": given $A \times B \xrightarrow{M} C$ with $A \times B$-many columns and C-many rows, reshape M into its **vectorized** version $B \xrightarrow{\mathbf{vec}_A\, M} A \times C$ 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, cf

$$c\ M\ (a, b) \quad = \quad (a, c)\ (\mathbf{vec}_A\, M)\ b \tag{7}$$

which holds for every a, b, c.

Generalizing data cubes

Vectorization thus has an inverse operation — **unvectorization**:

$$
\begin{tikzcd}
A \times B \to C \arrow[rr, bend left, "\mathbf{vec}_A"] \arrow[rr, phantom, "\cong"] & & B \to A \times C \arrow[ll, bend left, "\mathbf{unvec}_A"]
\end{tikzcd}
$$

That is, M can be retrieved back from $\mathbf{vec}_A\, M$ by unvectorizing it:

$$N = \mathbf{vec}_A\, M \quad \Leftrightarrow \quad \mathbf{unvec}_A\, N = M \tag{8}$$

Vectorization has a rich algebra, e.g. a **fusion**-law

$$(\mathbf{vec}\, M) \cdot N \quad = \quad \mathbf{vec}\,(M \cdot (id \otimes N)) \tag{9}$$

and an **absorption**-law:

$$\mathbf{vec}\,(M \cdot N) \quad = \quad (id \otimes M) \cdot \mathbf{vec}\, N \tag{10}$$

(Un)vectorization

Let us unvectorize our starting (data) tensor, across dimension *Year*:

$$
\begin{array}{c}
\mathit{Year} \times (\mathit{Color} \times \mathit{Model}) \longleftarrow 1 \\
\boldsymbol{unvec}_{\mathit{Year}} \left(\begin{array}{lll|c}
 & & & \text{ALL} \\ \hline
1990 & \mathit{Blue} & \mathit{Chevy} & 87 \\
 & & \mathit{Ford} & 99 \\
 & \mathit{Green} & \mathit{Chevy} & 0 \\
 & & \mathit{Ford} & 64 \\
 & \mathit{Red} & \mathit{Chevy} & 5 \\
 & & \mathit{Ford} & 0 \\ \hline
1991 & \mathit{Blue} & \mathit{Chevy} & 0 \\
 & & \mathit{Ford} & 7 \\
 & \mathit{Green} & \mathit{Chevy} & 0 \\
 & & \mathit{Ford} & 0 \\
 & \mathit{Red} & \mathit{Chevy} & 0 \\
 & & \mathit{Ford} & 8
\end{array} \right)
\end{array}
=
\begin{array}{c}
\mathit{Color} \times \mathit{Model} \longleftarrow \mathit{Year} \\
\begin{array}{ll|cc}
 & & 1990 & 1991 \\ \hline
\mathit{Blue} & \mathit{Chevy} & 87 & 0 \\
 & \mathit{Ford} & 99 & 7 \\ \hline
\mathit{Green} & \mathit{Chevy} & 0 & 0 \\
 & \mathit{Ford} & 64 & 0 \\ \hline
\mathit{Red} & \mathit{Chevy} & 5 & 0 \\
 & \mathit{Ford} & 0 & 8
\end{array}
\end{array}
$$

There is room for further unvectorizing the outcome, this time across *Color* — next slide:

(De)vectorization

Further unvectorization:

$$Color \times Model \longleftarrow Year \qquad\qquad Model \longleftarrow Color \times Year$$

$$\mathbf{unvec}_{Color}\left(\begin{array}{ll|cc} & & 1990 & 1991 \\ \hline Blue & Chevy & 87 & 0 \\ & Ford & 99 & 7 \\ \hline Green & Chevy & 0 & 0 \\ & Ford & 64 & 0 \\ \hline Red & Chevy & 5 & 0 \\ & Ford & 0 & 8 \end{array}\right) =$$

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

and so on.

Generic cubes

It turns out **that** cubes can be calculated for any such two-dimensional versions of our original data tensor, for instance,

$$\textbf{cube } N : \ \mathit{Model} + 1 \longleftarrow (\mathit{Color} + 1) \times (\mathit{Year} + 1)$$
$$\textbf{cube } N = \tau_{\mathit{Model}} \cdot N \cdot (\tau_{\mathit{Color}} \otimes \tau_{\mathit{Year}})^{\circ}$$

where N stands for the second matrix of the previous slide, yielding

	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

The 36 entries of the original cube have been rearranged in a 3*12 rectangular layout, as dictated by the **dimension** cardinalities.

The **cube** (LA) operator

Definition (Cube)

Let M be a matrix of type

$$\Pi_{j=1}^{n} B_j \xleftarrow{M} \Pi_{i=1}^{m} A_i \tag{11}$$

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

$$\textbf{cube}\ M \;=\; (\bigotimes_{j=1}^{n} \tau_{B_j}) \cdot M \cdot (\bigotimes_{i=1}^{m} \tau_{A_i})^{\circ} \tag{12}$$

where $\bigotimes$ is finite Kronecker product.

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

□

Properties of data cubing

Linearity:

$$\mathbf{cube}\ (M + N) \quad = \quad \mathbf{cube}\ M + \mathbf{cube}\ N \tag{13}$$

Proof: Immediate by bilinearity of matrix composition:

$$M \cdot (N + P) \quad = \quad M \cdot N + M \cdot P \tag{14}$$
$$(N + P) \cdot M \quad = \quad N \cdot M + P \cdot M \tag{15}$$

This can be taken advantage of not only in **incremental** data cube construction but also in **parallelizing** data cube generation.

Properties of data cubing

Updatability: by Khatri-Rao product linearity,

$$(M+N) \triangledown P = M \triangledown P + N \triangledown P$$
$$P \triangledown (M+N) = P \triangledown M + P \triangledown N$$

the **cube** operator commutes with the usual CRUDE operations, namely with record **updating**. For instance, suppose record

#	Model	Year	Color	Sale
5	*Ford*	1991	*Red*	8

cf

t_{Model}	1	2	3	4	5	6
Chevy	1	1	0	0	0	0
Ford	0	0	1	1	1	1

is updated to

#	Model	Year	Color	Sale
5	*Chevy*	1991	*Red*	8

cf

t'_{Model}	1	2	3	4	5	6
Chevy	1	1	0	0	1	0
Ford	0	0	1	1	0	1

Properties of data cubing

One just has to compute the “delta” projection,

$$\delta_{Model} = t'_{Model} - t_{Model} =$$

	1	2	3	4	5	6
Chevy	0	0	0	0	1	0
Ford	0	0	0	0	-1	0

then the “delta cube”,

$$d = (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot v'$$

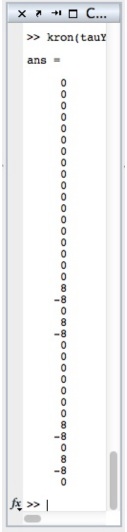
where

$$v' = (t_{Year} \triangledown (t_{Color} \triangledown \delta_{Model})) \cdot (t^{Sale})^{\circ}$$

and finally **add** the “delta cube” to the original cube:

$$c' = c + d.$$

— see the simulation aside (MATLab).



Properties of data cubing

Theorem (Cube commutes with vectorization)

Let $X \xleftarrow{M} Y \times C$ *and* $Y \times X \xleftarrow{\mathbf{vec}\,M} C$ *be its* Y*-vectorization. Then*

$$\mathbf{vec}\,(\mathbf{cube}\ M) = \mathbf{cube}\ (\mathbf{vec}\,M) \tag{16}$$

holds.

□

The proof (in the paper) relies on the type diagrams:

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

Properties of data cubing

The following theorem shows that changing the dimensions of a data cube does not change its totals.

Theorem (Free theorem)

Let $B \xleftarrow{M} A$ *be cubed into* $B+1 \xleftarrow{\mathbf{cube}\ M} A+1$, *and* $r : C \to A$ *and* $s : D \to B$ *be arbitrary functions. Then*

$$\mathbf{cube}\ (s^\circ \cdot M \cdot r) \quad = \quad (s^\circ \oplus id) \cdot (\mathbf{cube}\ M) \cdot (r \oplus id) \tag{17}$$

holds, where $M \oplus N = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix}$ *is matrix* **direct sum**.

□

The proof given in the paper resorts to the **free theorem** of polymorphic operators popularized by Wadler (1989) under the heading *Theorems for free!*.

Cube universality — slicing

Slicing is a specialized filter for a particular value in a dimension.

Suppose that from our starting cube

$$c : 1 \rightarrow (Year + 1) \times ((Color + 1) \times (Model + 1))$$

one is only interested in the data concerning year 1991.

It suffices to regard data values as (categorial) **points**: given $p \in A$, constant function $\underline{p} : 1 \rightarrow A$ is said to be a *point* of A, for instance

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

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

Cube universality — slicing

Example:

$$1 \xrightarrow{c} (\mathit{Year}+1)\times((\mathit{Color}+1)\times(\mathit{Model}+1)) \xrightarrow{\underline{1991}^\circ \otimes \mathit{id}} 1\times((\mathit{Color}+1)\times(\mathit{Model}+1)) \quad = \quad \begin{bmatrix}0\\7\\7\\0\\0\\0\\0\\8\\8\\0\\15\\15\end{bmatrix}$$

Cube universality — rolling-up

Gray et al. (1997) say that *going up the levels [of aggregated data] is called rolling-up*.

In this sense, a **roll-up** operation over dimensions A, B and C could be the following form of (increasing) summarization:

$$A \times (B \times C)$$

$$A \times B$$

$$A$$

$$1$$

How does this work over a data cube? We take the simpler case of two dimensions A, B as example.

Cube universality — rolling-up

The dimension powerset for A, B is captured by the corresponding matrix **injections** onto the cube target type $(A+1)\times(B+1)$:

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

where

$$\theta = i_1 \otimes i_1$$
$$\alpha = i_1 \triangledown i_2 \cdot !$$
$$\beta = i_1 \cdot ! \triangledown i_2$$
$$\omega = i_2 \triangledown i_2$$

NB: the injections i_1 and i_2 are such that $[i_1|i_2] = id$, where $[M|N]$ denotes the horizonal gluing of two matrices.

Cube universality — rolling-up

One can build compound injections, for instance

$$\rho : (A+1) \times (B+1) \leftarrow A \times B + (A+1)$$
$$\rho = [\theta| [\alpha|\omega]]$$

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

$$\rho^\circ \cdot (\textbf{cube}\ M) = \left[\frac{M}{\left[\frac{fst \cdot M}{! \cdot M} \right]} \right] \cdot \tau_C^\circ$$

extracts from **cube** M the corresponding **roll-up**.

The next slides give a concrete example.

Cube universality — rolling-up

Let M be the (generalized) data cube

		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

Cube universality — rolling-up

Building the injection matrix $\rho = [\theta | \, [\alpha | \omega]]$ for types $\mathit{Color} \times \mathit{Model} + \mathit{Color} + 1 \rightarrow (\mathit{Color} + 1) \times (\mathit{Model} + 1)$ we get the following matrix (already transposed):

		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

Cube universality — rolling-up

Then

$$\rho^\circ \cdot \textbf{cube}\ M \;=\; \begin{array}{ll|ccc} & & 1990 & 1991 & \textsc{all} \\ \hline \multirow{2}{*}{\textit{Blue}} & \textit{Chevy} & 87 & 0 & 87 \\ & \textit{Ford} & 99 & 7 & 106 \\ \hline \multirow{2}{*}{\textit{Green}} & \textit{Chevy} & 0 & 0 & 0 \\ & \textit{Ford} & 64 & 0 & 64 \\ \hline \multirow{2}{*}{\textit{Red}} & \textit{Chevy} & 5 & 0 & 5 \\ & \textit{Ford} & 0 & 8 & 8 \\ \hline & \textit{Blue} & 186 & 7 & 193 \\ & \textit{Green} & 64 & 0 & 64 \\ & \textit{Red} & 5 & 8 & 13 \\ \hline & \textsc{all} & 255 & 15 & 270 \end{array}$$

Note how a roll-up is a particular "subset" of a cube.

Matrix ρ° performs the (quantitative) selection of such a subset.

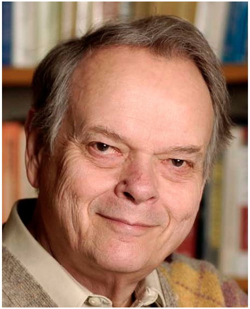
Summing up

Data science seems to be ignoring the role of **types** and **type parametricity** in software — one of the most significant advances in CS.

Nice theory called **parametric polymorphism** (John Reynolds, CMU).

So nice that you can derive **properties** of your operations **solely** by looking at their **types** 🙂

As Kurt Lewin (1890-1947) once write it: *"There is nothing more practical than a good theory"*.



J.C. Reynolds
(1935–2013)

Summing up

Abadir and Magnus (2005) stress on the need for a **standardized** notation for **linear algebra** in the field of econometrics and **statistics**.

This talk suggests such a notation should be **polymorphically typed**.

Since (Macedo and Oliveira, 2013) the author has invested in **typing** linear algebra in a way that makes it closer to modern **typed** languages.

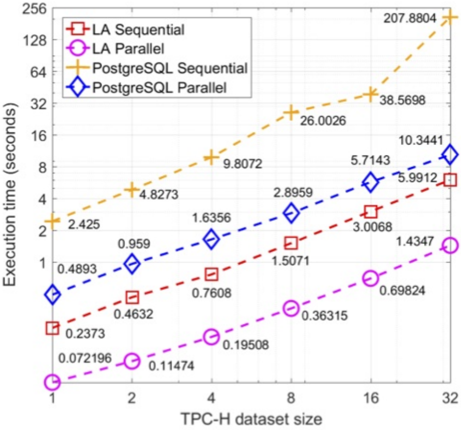
This extends previous efforts on applying LA to **OLAP** (Macedo and Oliveira, 2015)

(Still not convinced? Peek the next slide.)

Annex

(For those who care mostly about efficiency)

Aside: Plot taken from a recent MSc report on TPC-H benchmarking LA approach to analytical querying (on-going work).



HIGH-ASSURANCE SOFTWARE LABORATORY
IMPROVING PRACTICE THROUGH THEORY

References

K.M. Abadir and J.R. Magnus. *Matrix algebra. Econometric exercises 1*. C.U.P., 2005.

J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Mining and Knowledge Discovery*, 1(1):29–53, 1997. URL `citeseer.nj.nec.com/article/gray95data.html`.

H.D. Macedo and J.N. Oliveira. Typing linear algebra: A biproduct-oriented approach. *SCP*, 78(11):2160–2191, 2013.

H.D. Macedo and J.N. Oliveira. A linear algebra approach to OLAP. *FAoC*, 27(2):283–307, 2015.

J. N. Oliveira and H. D. Macedo. The data cube as a typed linear algebra operator. In *Proc. of the 16th Int. Symposium on Database Programming Languages*, DBPL '17, pages 6:1–6:11, New York, NY, USA, 2017. ACM.

P.L. Wadler. Theorems for free! In *4th International Symposium on Functional Programming Languages and Computer Architecture*, pages 347–359, London, Sep. 1989. ACM.