On the meaning of $curry(M)$ for M a matrix

J.N. Oliveira
(joint work with H. Macedo)

HASLab/Universidade do Minho
Braga, Portugal

IFIP WG2.1 meeting #67
May 2011
Reykjavik, Iceland

Quotation

"Using matrix notation such a set of simultaneous equations takes the form $A \cdot x = b$ *where* x *is the vector of unknown values,* A *is the matrix of coefficients and* b *is the vector of values on the right side of the equation."*

"In this way a set of equations has been reduced to a **single** *equation."*

"This is a tremendous improvement in **concision** *that does not incur any loss of* **precision!**"

Roland Backhouse (2004)

Concision — is computer science "concise enough"?

Perhaps not: we write long-winded formulæ such as eg.

$$\langle \forall\ y, x\ ::\ \langle \exists\ z\ ::\ r(y,z) \wedge p(z,x) \rangle \Rightarrow \langle \exists\ w\ ::\ s(y,w) \wedge q(w,x) \rangle \rangle$$

for something which could be written, in "matrix format", as

$$R \cdot P \ \subseteq \ S \cdot Q$$

Question:

- "Matrix"? Are there matrices in predicate logic?

Answer:

- Yes — **binary relations**, which are Boolean matrices.

The function-relation-matrix hierarchy

- **Relations** — are everywhere, eg.

  $$y \text{ likes } x$$
  $$y \leq x$$

  (graphs, etc)

- **Functions** — deterministic and total relations, eg.

  $$y = ax + b$$
  $$y = \text{birthplace of } x$$

  ("left-linear" graphs, etc)

- **Matrices** — quantified relations, cf.

  $$y \; M \; x = k$$

  further to

  $$y \; M \; x = \text{true}$$

  (weighted graphs, etc)

Applications in computer science

- **Functions** — functional programming, an advanced discipline strongly rooted on mathematics.
- **Relations** — ubiquitous (eg. graphs) but still under the atavistic *set of pairs* interpretation.
- **Matrices** — key concept in mathematics as a whole, many tools (eg. MATLAB, Mathematica) but still "untyped".

Typed versus untyped

- **Functions** — polymorphically typed, cf. triples of the form $B \xleftarrow{f} A$, they form a category under composition $(f \cdot g)$, etc.

- **Relations** — polymorphically typed if understood in the **Rel** allegory, where $R \cdot S$ means relation composition and $R \subseteq S$ means a partial order (one per type, ie. homset).

- **Matrices** — our "Matrices as arrows!" campaign promotes a typed approach to linear algebra (LA) rooted on categories of matrices.

Matrices are Arrows

Given

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}_{m \times n}$$

$$m \xleftarrow{A} n$$

$$B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}_{n \times k}$$

$$n \xleftarrow{B} k$$

define

$$m \xleftarrow{A} n \xleftarrow{B} k \qquad m \xleftarrow{A \cdot B} k$$

Category of matrices

Under $A \cdot B$ (vulg. MMM)

- matrices form a **category** whose **objects** are matrix dimensions and whose **morphisms** $m \xleftarrow{A} n$, $n \xleftarrow{B} k$ are the matrices themselves;
- every identity $n \xleftarrow{id} n$ is the diagonal of size n, that is, $id(r, c) \triangleq r = c$ under the $(0, 1)$ encoding of the Booleans:

$$id_n = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{n \times n} \qquad n \xleftarrow{id_n} n$$

Vectors

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

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \qquad \text{and} \qquad w = \begin{bmatrix} w_1 & \dots & w_n \end{bmatrix}$$

Column vector v is of type $m \longleftarrow 1$ (m rows, one column) and **row** vector w is of type $1 \longleftarrow n$ (one row, n columns).

Our convention is that lowercase letters (eg. v, w) denote vectors and uppercase letters (eg. A, M) denote arbitrary matrices.

Categories of Matrices

Very rarely used in computing, to the best of our knowledge.

Notable exception

Work on **linear algebras of machines** by Bloom et al. (1996), who

- develop a "machines as matrices" theory of concurrency where the underlying matrix-cell algebra is the semiring $\mathcal{L}(X^*)$ of subsets of words on a finite alphabet X.
- They define machine **composition**, machine target and source **tupling**, and machine **product**.

Via these constructions they obtain a compositional and modular approach to build complex machines.

Our aim

To enrich relation and linear algebra with ingredients which have proved effective in functional programming, namely

- types
- polymorphism
- type checking
- calculational techniques.

Common ground?

- Mild use of category theory.

In this talk we focus on one such good ingredient — **transposition**.

Curry/uncurry

Functional programming's "Swiss army knife" with two blades,

```
curry ::  ((a,b) -> c) -> (a -> b -> c)
curry f x y = f(x,y)
```

and

```
uncurry ::  (a -> b -> c) -> ((a,b) -> c)
uncurry f (a,b) = f a b
```

These higher-order functions enable programmers to

- defer **input** to **output**;
- fit a **binary** function where a **unary** one is expected;
- temporarily *freeze* parameters of a binary function while these don't play a role in the computation.

Exponentials

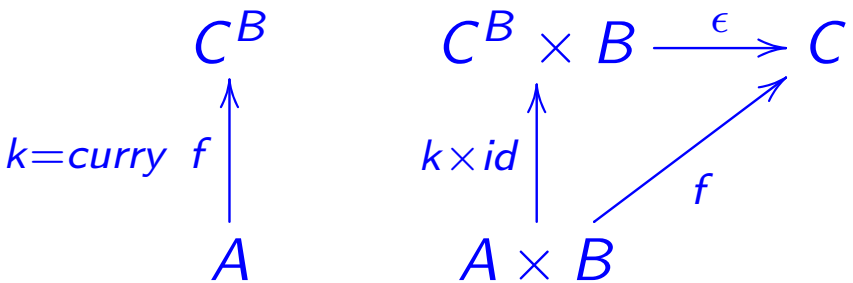
No loss of information (isomorphisms)

$$A \times B \to C \quad \underset{\text{uncurry}}{\overset{\text{curry}}{\cong}} \quad A \to C^B$$

thanks to universal property

$$curry\ f = k \ \Leftrightarrow\ f = \epsilon \cdot (k \times id)$$

where ϵ is the `eval` function, cf. diagram

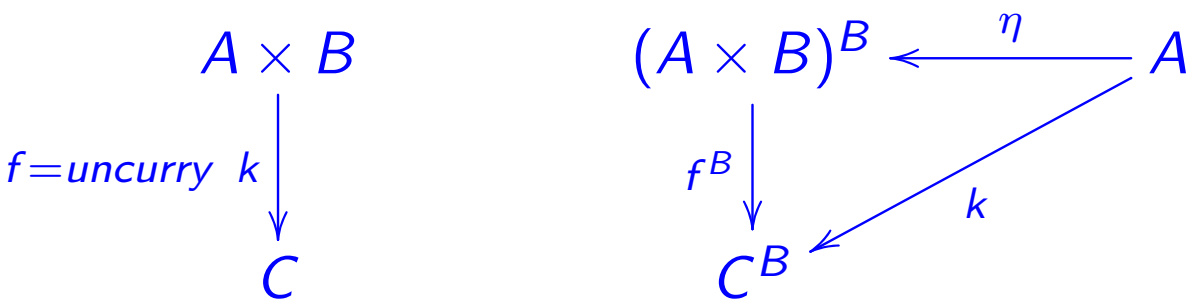


Alternative (with a monad inside)

Universal property:

$$uncurry\ k = f \ \Leftrightarrow\ k = f^B \cdot \eta$$

cf. diagram



where η is the unit (`return` function) of the state **monad** on B.

Products — "where" exponentials come from

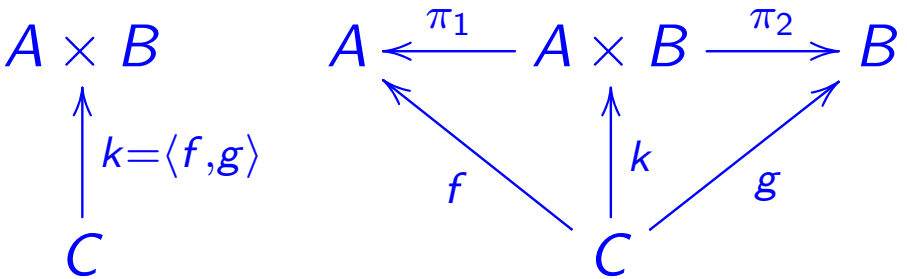
Exponentials thus explained by adjunction

$$(\times B) \dashv (\_^B)$$

where product is another universal construct:

$$k = \langle f, g \rangle \quad \Leftrightarrow \quad \begin{cases} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{cases}$$

cf. diagram

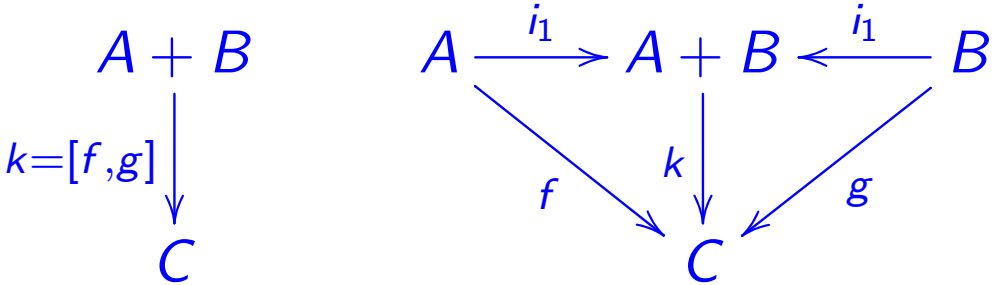


Coproducts — sums

Dual concept expressing alternation:

$$k = [f, g] \quad \Leftrightarrow \quad \begin{cases} k \cdot i_1 = f \\ k \cdot i_2 = g \end{cases}$$

cf. diagram



where $A + B$ expresses disjoint union.

1st step in the generalization

From functions to relations:

- Relational type $A \to B$ (homset) far larger than the same restricted to functions.
- Pairing not injective — $\langle R, P \rangle = \langle S, Q \rangle$ no longer entails $R = S$ and $P = Q$ — so product is gone!
- Coproduct survives :-)
- A new isomorphic transformation — converse $B \xrightarrow{R^\circ} A$ given $A \xrightarrow{R} B$ — since relations are invertible.

Biproducts

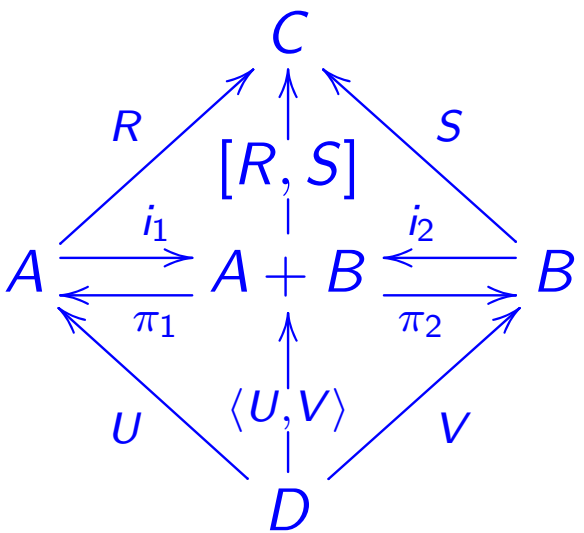
Products still to be found in the category, but elsewhere — in fact "attached" to coproducts, as converse duals:

$$\langle R, S\rangle^\circ \quad = \quad [R^\circ, S^\circ]$$

Altogether, they form a **biproduct**:

$$\begin{array}{lcl} [R, S] & = & R \cdot \pi_1 \cup S \cdot \pi_2 \\ \langle U, V\rangle & = & i_1 \cdot U \cup i_2 \cdot V \end{array}$$

where $\pi_1 = i_1^\circ$
$\pi_2 = i_2^\circ$



Biproducts in general

In an Abelian category, a *biproduct* diagram for the objects m, n is a diagram of shape

$$m \underset{i_1}{\overset{\pi_1}{\leftrightarrows}} r \underset{i_2}{\overset{\pi_2}{\leftrightarrows}} n$$

whose arrows π_1, π_2, i_1, i_2 satisfy the identities which follow:

$$\pi_1 \cdot i_1 = id_m \quad (1)$$

$$\pi_2 \cdot i_2 = id_n \quad (2)$$

$$i_1 \cdot \pi_1 + i_2 \cdot \pi_2 = id_r \quad (3)$$

Two orthogonality properties follow:

$$\pi_1 \cdot i_2 = 0 \quad , \quad \pi_2 \cdot i_1 = 0 \quad (4)$$

"Standard LA biproduct"

Among the many solutions to the biproduct equations we select the following

$$\pi_1 = m \xleftarrow{[id_m \mid 0]} m+n \quad , \quad \pi_2 = n \xleftarrow{[0 \mid id_n]} m+n$$

$$i_1 = m+n \xleftarrow{\left[\frac{id_m}{0}\right]} m \quad , \quad i_2 = m+n \xleftarrow{\left[\frac{0}{id_m}\right]} n$$

which are made of fragments of id

$$[i_1|i_2] = id \tag{5}$$

$$\left[\frac{\pi_1}{\pi_2}\right] = id \tag{6}$$

and enough to explain "block notation" used in LA.

Blocked LA

Thanks to universal constructs:

- Structural equality laws (over the same biproduct):

$$[A|B] = [C|D] \quad \Leftrightarrow \quad A = C \wedge B = D \tag{7}$$

$$\left[\frac{A}{B}\right] = \left[\frac{C}{D}\right] \quad \Leftrightarrow \quad A = C \wedge B = D \tag{8}$$

- Injections and projections in a biproduct are unique. For instance,

$$P \cdot \left[\frac{U}{V}\right] = U \ \wedge \ Q \cdot \left[\frac{U}{V}\right] = V \quad \Leftrightarrow \quad P = \pi_1 \wedge Q = \pi_2 \tag{9}$$

"Forks"

Still room for "old products" — renamed "forks".

The relational fork operator implicit in Tarski's work

$$(a, b)(R \vartriangle S)c \quad \Leftrightarrow \quad aRc \wedge bSc$$

is a conservative extension of functional pairing

$$(f \vartriangle g)c \quad = \quad (f\ c, g\ c)$$

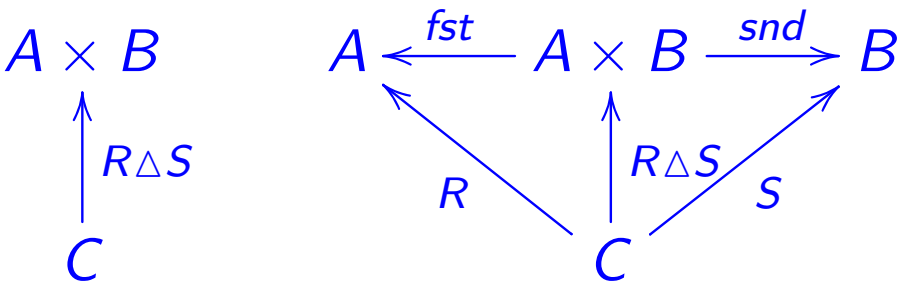
but no longer a categorial product.

"Forks"

In fact:

$$X \subseteq R \vartriangle S \quad \Leftrightarrow \quad \begin{cases} fst \cdot X \subseteq R \\ snd \cdot X \subseteq S \end{cases}$$

cf. diagram

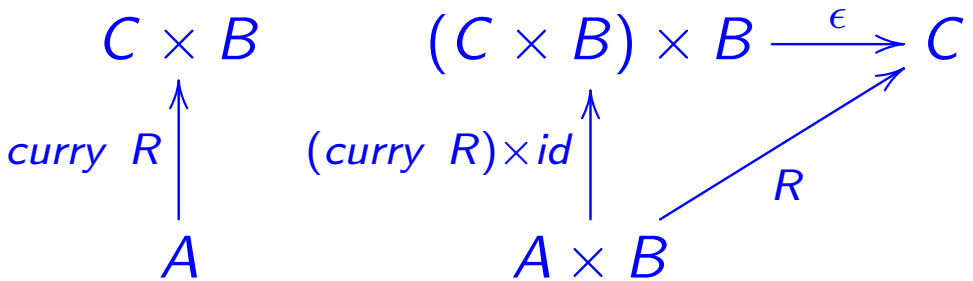


Dramatic impact on "currying"

Isomorphism now is

$$A \times B \to C \quad \underset{\text{uncurry}}{\overset{\text{curry}}{\cong}} \quad A \to C \times B$$

Diagram:



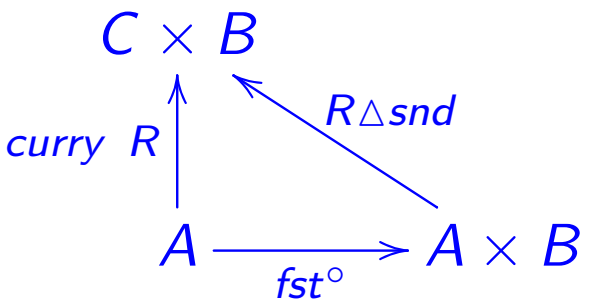
Self-adjunction

Relational currying thus explained by **self**-adjunction

$$(\times B) \dashv (\times B)$$

Details:

$$curry\ R \triangleq (R \triangle snd) \cdot fst^{\circ}$$



that is

$$(c, b)(curry\ R)a \Leftrightarrow c\ R\ (a, b)$$

Moral: every n-ary relation is a binary relation; where (input/output) you place the attributes is irrelevant.

Converse duality

Relational uncurrying:

$$\begin{aligned} uncurry\ R &\triangleq curry(R^\circ)^\circ \\ &= fst \cdot \langle R^\circ, snd \rangle^\circ \end{aligned}$$

Then

$$\epsilon = uncurry\ id = fst \cdot \langle id, snd \rangle^\circ$$

With points:

$$c_2\ \epsilon\ ((c_1, b_1), b_2) \quad \Leftrightarrow \quad c_2 = c_1 \wedge b_1 = b_2$$

2nd step in the generalization

From relations to matrices:

- What about *curry* M for M a matrix?
- Types are dimensions which can be factored, for instance

$$k \times m \to n \quad \underset{\text{uncurry}_k}{\overset{\text{curry}_k}{\cong}} \quad m \to k \times n$$

- "Deja vu"? **Yes!** See next slide on matrix vectorization.

Vectorization

Example: matrix

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad \text{converted to} \quad \mathbf{vec}\, A = \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \\ a_{13} \\ a_{23} \end{bmatrix}$$

Type-wise:

$$3 \times 1 \to 2 \quad \underset{\textit{uncurry}_3}{\overset{\textit{curry}_3}{\cong}} \quad 1 \to 3 \times 2$$

Transformations which preserve the "area" of the original matrix.

Parallel

Intuition — vectorization is akin to exponentiation:

> *While currying "thins" the input of a given binary function* $m \times k \xrightarrow{f} n$ *by converting it into its unary (higher-order) counterpart* $m \xrightarrow{curry\ f} n^k$ *, so does vectorization by thinning a given matrix* $k \times m \xrightarrow{M} n$ *into* $m \xrightarrow{\mathbf{vec}\, M} k \times n$ *, where* k *is the "thinning factor".*

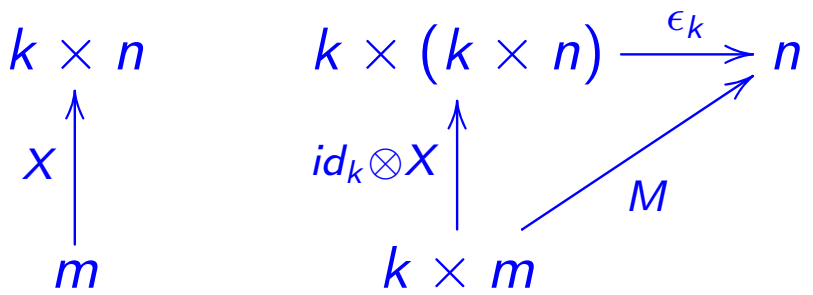
(For $m = 1$, $\mathbf{vec}\, M$ will be a column vector.)

Self-adjunction

Vectorization with thinning factor k (in general) is captured by self-adjunction

$$(k\times) \dashv (k\times)$$

cf. diagram



and universal property

$$X = \mathbf{vec}_k\, M \;\Leftrightarrow\; M = \epsilon_k \cdot (id_k \otimes X) \tag{10}$$

where $\otimes$ is the Kronecker product (a bifunctor in the category).

Vectorization refactoring

Macedo and Oliveira (2011) show how standard vectorization theory arises from this universal property and biproduct algebra, namely the following relationship

$$\mathbf{vec}\,(A \cdot B \cdot C) \quad = \quad (C^{\circ} \otimes A) \cdot \mathbf{vec}\,B \tag{11}$$

which Abadir and Magnus (2005) attribute to Roth (1934) and regard as the **fundamental result** of the whole theory.

Such theory is often clumsy, full of index-wise *"..."-built* matrices hard to grasp.

Thanks to typed (block) LA combinators and biproduct algebra one can infer closed, polymorphic formulæ for such matrices. Let us infer that of the counit ϵ_k.

LA refreshing — Direct sum

Direct sum

$$A \oplus B = [i_1 \cdot A | i_2 \cdot B] = \left[\begin{array}{c|c} A & 0 \\ \hline 0 & B \end{array}\right] \quad (12)$$

is a bifunctor, cf.

$$\begin{array}{ccc} n & m & n+m \\ {\scriptstyle A}\downarrow & {\scriptstyle B}\downarrow & \downarrow{\scriptstyle A \oplus B} \\ k & j & k+j \end{array}$$

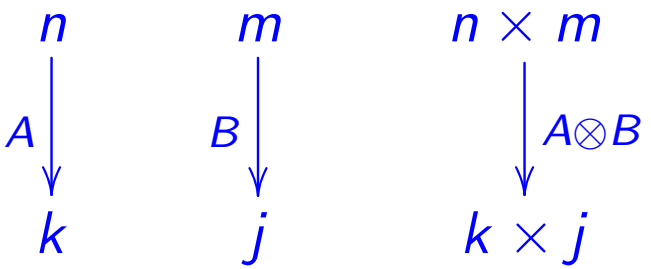
such that

$$id_2 \otimes A = A \oplus A \quad (13)$$

$$[A|B] \cdot (C \oplus D) = [A \cdot C | B \cdot D] \quad (14)$$

Kronecker product

Another bifunctor



whose fusion laws

$$[A|B] \otimes C = [A \otimes C|B \otimes C] \tag{15}$$

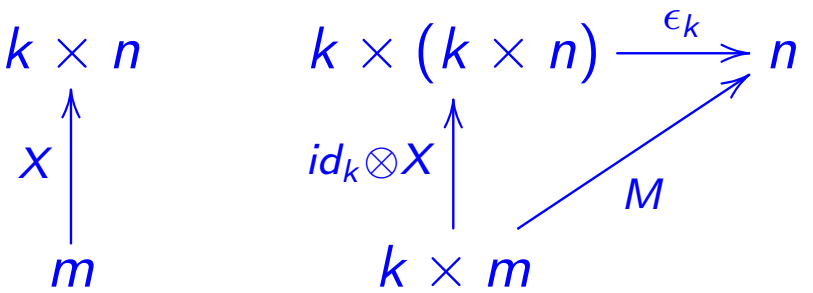
$$\left[\frac{A}{B}\right] \otimes C = \left[\frac{A \otimes C}{B \otimes C}\right] \tag{16}$$

capture its meaning block-wise. Another property relevant for our purposes is simplification rule

$$\pi_j \otimes id = \pi_j \quad (j = 1, 2) \tag{17}$$

Chasing ϵ_k

- We want to characterize ϵ_k in



  for any k, so that a generic theory of "blocked vectorization" becomes available.

- For $k = 2$, the smallest possible case happens for $m = n = 1$, where one expects $\mathbf{vec}_2\ [x\ y]$ to be $\begin{bmatrix} x \\ y \end{bmatrix}$, for x and y elementary data.

- Scaling up: replace x and y by blocks $n \xleftarrow{A} m$ and $n \xleftarrow{B} m$, respectively, and reason:

Reasoning

$$\mathbf{vec}_2\ [A|B] \;=\; \left[\frac{A}{B}\right]$$

$\Leftrightarrow$ { (10) }

$$[A|B] \;=\; \epsilon_2 \cdot (id_2 \otimes \left[\frac{A}{B}\right])$$

$\Leftrightarrow$ { (13) ; unjunc ϵ_2 into ϵ_{21} and ϵ_{22} }

$$[A|B] \;=\; [\epsilon_{21}|\epsilon_{22}] \cdot (\left[\frac{A}{B}\right] \oplus \left[\frac{A}{B}\right])$$

$\Leftrightarrow$ { $\oplus$-absorption (14) }

$$[A|B] \;=\; \left[\epsilon_{21} \cdot \left[\frac{A}{B}\right] | \epsilon_{22} \cdot \left[\frac{A}{B}\right]\right]$$

Reasoning continued

$$
\begin{aligned}
[A|B] &= \left[\epsilon_{21} \cdot \left[\frac{A}{B}\right] | \epsilon_{22} \cdot \left[\frac{A}{B}\right]\right] \\
\Leftrightarrow & \quad \{\ (7)\ ;\ (9)\ \} \\
& \epsilon_{21} = \pi_1 \wedge \epsilon_{22} = \pi_2 \\
\Leftrightarrow & \quad \{\ \text{junc } \epsilon_{21} \text{ and } \epsilon_{22} \text{ back into } \epsilon_2\ \} \\
& \epsilon_2 = [\pi_1|\pi_2]
\end{aligned}
$$

Thus we get ϵ_2 with type

$$
n \xleftarrow{\epsilon_2=[\pi_1|\pi_2]} 2n + 2n \tag{18}
$$

expressing ϵ_k (for $k = 2$) in terms of the standard biproduct projections.

Example

Check the following instance of cancellation law,

$$M = \epsilon_k \cdot (id_k \otimes \mathbf{vec}_k\, M) \tag{19}$$

for $k = m = n = 2$:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot (id_2 \otimes \begin{bmatrix} a_{11} \\ a_{21} \\ a_{12} \\ a_{22} \end{bmatrix})$$

Clearly,

$$\epsilon = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \end{bmatrix} \otimes id_2$$

A general fact revealing the polymorphism of ϵ:

$$\epsilon \otimes id \quad = \quad \epsilon \tag{20}$$

Example

Calculation:

$$
\begin{aligned}
& \epsilon \otimes id \\
= & \quad \{ \text{ definition (18) } \} \\
& [\pi_1 | \pi_2] \otimes id \\
= & \quad \{ \text{ distribution (15) } \} \\
& [\pi_1 \otimes id | \pi_2 \otimes id] \\
= & \quad \{ \pi_j \otimes id = \pi_j \text{ (17) twice; (18) } \} \\
& \epsilon
\end{aligned}
$$

Typewise:

$$
(\; k^2 n \xrightarrow{\epsilon_k} n \;) \otimes (\; j \xrightarrow{id} j \;) \quad = \quad k^2(nj) \xrightarrow{\epsilon_k} nj
$$

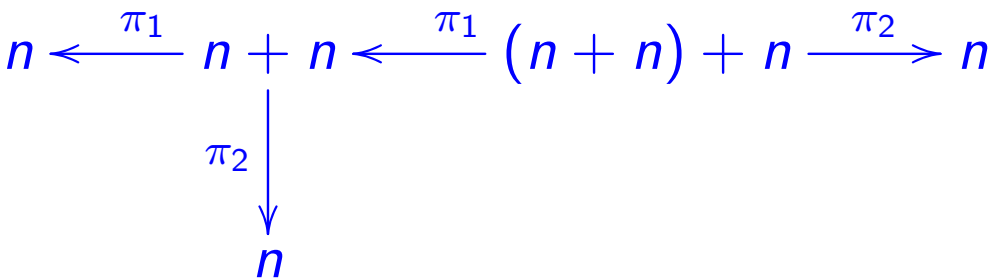
This provides an elegant explanation for the index-wise construction of ϵ given in Došen and Petrić (2003).

Generic counit

Doing a similar exercise for $k = 3$, one would obtain

$$3^2 n \xrightarrow{\epsilon_3} n \quad = \quad [[\pi_1 \cdot \pi_1 | \pi_2 \cdot \pi_1] | \pi_2]$$

with types as in diagram:



Note that

$$\begin{aligned} \epsilon_3 &= [[\pi_1 | \pi_2] \cdot (\pi_1 \oplus \pi_1) | \pi_2] \\ &= [\epsilon_2 \cdot (id_2 \otimes \pi_1) | \pi_2] \end{aligned}$$

thanks to absorption laws (14), (13) and ϵ_2 (18).

Generic counit

This provides a hint of the general case:

$$\epsilon_{k+1} = [\epsilon_k \cdot (id_k \otimes \pi_1)|\pi_2] \quad (21)$$

$$\epsilon_1 = id \quad (22)$$

No explicit dimensions – is this "well-formed"?

Let us calculate its "principal type" using Hindley-Milner's algorithm *modulo* some arithmetics.

Polymorphic matrix type inference

To typecheck

$$\epsilon_{k+1} \quad = \quad [\epsilon_k \cdot (id_k \otimes \pi_1) | \pi_2]$$

we proceed by unification, starting from completely independent types as starting point:

$$j \xleftarrow{\epsilon_{k+1}} (k+1)^2 j$$

$$i \xleftarrow{\epsilon_k} k^2 i$$

$$k \xleftarrow{id_k} k$$

$$n \xleftarrow{\pi_1} n + m$$

$$b \xleftarrow{\pi_2} a + b$$

Type equations $a = n$ and $b = m$ follow from π_1 and π_2 belonging to the same biproduct.

Polymorphic matrix type inference

Term $\epsilon_k \cdot (id_k \otimes \pi_1)$ forces type equation ("unification")

$$k^2 i = kn$$

that is, $n = ki$. Term $[\epsilon_k \cdot (id_k \otimes \pi_1)|\pi_2]$ entails $i = m$. Finally, the whole equality forces:

$$\begin{aligned} j &= m \\ (k+1)^2 j &= k(ki+m) + m + ki \end{aligned}$$

Unfolding and substituting, $k^2 m + 2km + m = k^2 i + km + i + ki$ yields $i = m$. Principal type of (21) thus is:

$$m \xleftarrow{\epsilon_{k+1}} (k+1)^2 m$$

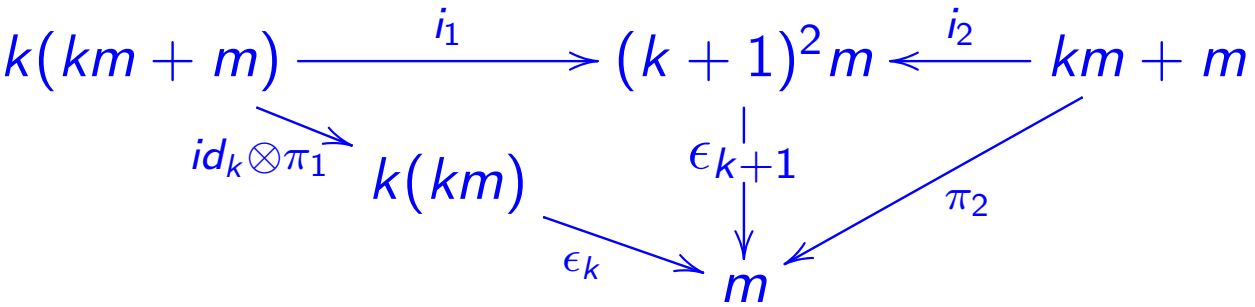
$$m \xleftarrow{\epsilon_k} k^2 m$$

$$k \xleftarrow{id_k} k$$

$$km \xleftarrow{\pi_1} km + m$$

$$m \xleftarrow{\pi_2} km + m$$

We could have used a diagram instead



Why explicit dimensions in LA?

Cf. the following piece of MATLAB code for ϵ_k

```
function E = epsilon(k,m)
   if k==1
      E = eye(m)
   else
      n=k-1;
      p1 = eye(n*m,k*m);
      p2 = jay(m,k*m);
      E = [ (epsilon(n,m) * kron(eye(n),p1)) p2 ]
   end
end
```

Explicit dimensions are a burden, cause mistakes and reduce polymorphism (writer unaware that there is a more general type...)

Summing up

- Linear algebra is polymorphically typed.
- Improvement over the loose *"valid only for matrices of the same order"* (Abadir and Magnus, 2005) attitude found in the literature.
- The prospect of building biproduct-based type checkers for computer algebra systems such as MATLAB within reach.
- This seems to be already the approach in Cryptol (Browning, 2010), a Haskell based DSL for cryptography — can we improve it?

Advertisement: talk proposed about OLAP stuff illustrates a quite practical application of typed LA.

K.M. Abadir and J.R. Magnus. *Matrix algebra. Econometric exercises 1*. Cambridge , United Kingdom: Cambridge University Press, 2005.

R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

Stephen L. Bloom, N. Sabadini, and R. F. C. Walters. Matrices, machines and behaviors. *Applied Categorical Structures*, 4: 343–360, 1996. ISSN 0927-2852. URL `http://dx.doi.org/10.1007/BF00122683`. 10.1007/BF00122683.

Sally Browning. Cryptol, a DSL for cryptographic algorithms. In *ACM SIGPLAN Commercial Users of Functional Programming*, CUFP '10, pages 9:1–9:1, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0516-7.

K. Došen and Z. Petrić. Self-adjunctions and matrices. *Journal of Pure and Applied Algebra*, 184:7–39, 2003. doi: http://doi.acm.org/10.1016/S0022-4049(03)00084-7.

H.D. Macedo and J.N. Oliveira. Typing linear algebra: A

biproduct-oriented approach, 2011. (Submitted to Science of Computer Programming).

W. E. Roth. On direct product matrices. Bulletin of the American Mathematical Society(40):461–468, 1934.