

An AoP approach to typed linear algebra

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

Dept. Informática,
Universidade do Minho
Braga, Portugal

IFIP WG2.1 meeting #65
27th January 2010
Braga, Portugal

Context and Motivation

- The advent of on-chip **parallelism** poses many challenges to current programming languages.
- Traditional approaches (compiler + hand-coded optimization) are giving place to trendy DSL-based generative techniques.
- In areas such as scientific computing, image/video processing, the bulk of the work performed by so-called **kernel** functions.
- Examples of kernels are matrix-matrix multiplication (MMM), the discrete Fourier transform (DFT), etc.
- Kernel **optimization** is becoming very difficult due to the complexity of current computing platforms.

Teaching computers to write fast numerical code

In the SPIRAL Group (CMU), a DSL has been defined (**OL**) (Franchetti et al., 2009) to specify kernels in a data-independent way.

- **Divide-and-conquer** algorithms are described as **OL** breakdown rules.
- By recursively applying these rules a space of algorithms for a desired kernel can be generated.

Rationale behind SPIRAL:

- Target imperative code is too late for numeric processing kernel optimization.
- Such optimization can be elegantly and efficiently performed well above in the design chain once the maths themselves are expressed in an **index-free** style.

Starting point

Synergy:

- Parallel between the pointfree notation of **OL** and **relational algebra** (relations are Boolean matrices)
- Rich calculus of algebraic rules.

Observation:

- Relational calculus is typed once relations are regarded as **arrows** in the **Rel** allegory.
- What about the matrix calculus?

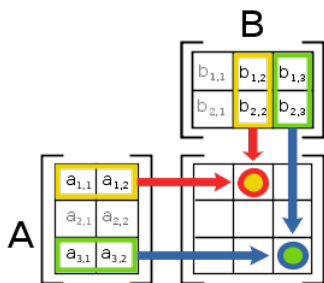
OL Sample (Franchetti et al., 2009)

name	definition
<i>Linear, arity (1,1)</i>	
identity	$I_n : \mathbb{C}^n \rightarrow \mathbb{C}^n; \mathbf{x} \mapsto \mathbf{x}$
vector flip	$J_n : \mathbb{C}^n \rightarrow \mathbb{C}^n; (x_i) \mapsto (x_{n-i})$
transposition of an $m \times n$ matrix	$L_m^{mn} : \mathbb{C}^{mn} \rightarrow \mathbb{C}^{mn}; \mathbf{A} \mapsto \mathbf{A}^T$
matrix $M \in \mathbb{C}^{m \times n}$	$M : \mathbb{C}^n \rightarrow \mathbb{C}^m; \mathbf{x} \mapsto M\mathbf{x}$
<i>Multilinear, arity (2,1)</i>	
Point-wise product	$P_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n; ((x_i), (y_i)) \mapsto (x_i y_i)$
Scalar product	$S_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}; ((x_i), (y_i)) \mapsto \Sigma(x_i y_i)$
Kronecker product	$K_{m \times n} : \mathbb{C}^m \times \mathbb{C}^n \rightarrow \mathbb{C}^{mn}; ((x_i), \mathbf{y}) \mapsto (x_i \mathbf{y})$
<i>Others</i>	
Fork	$\text{Fork}_n : \mathbb{C}^n \rightarrow \mathbb{C}^n \times \mathbb{C}^n; \mathbf{x} \mapsto (\mathbf{x}, \mathbf{x})$
Split	$\text{Split}_n : \mathbb{C}^n \rightarrow \mathbb{C}^{n/2} \times \mathbb{C}^{n/2}; \mathbf{x} \mapsto (\mathbf{x}^U, \mathbf{x}^L)$
Concatenate	$\oplus_n : \mathbb{C}^n \times \mathbb{C}^m \rightarrow \mathbb{C}^{n+m}; (\mathbf{x}, \mathbf{y}) \mapsto \mathbf{x} \oplus \mathbf{y}$
Duplication	$\text{dup}_n^m : \mathbb{C}^n \rightarrow \mathbb{C}^{nm}; (\mathbf{x} \mapsto \mathbf{x} \otimes I_m)$
Min	$\min_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n; (\mathbf{x}, \mathbf{y}) \mapsto (\min(x_i, y_i))$
Max	$\max_n : \mathbb{C}^n \times \mathbb{C}^n \rightarrow \mathbb{C}^n; (\mathbf{x}, \mathbf{y}) \mapsto (\max(x_i, y_i))$

Table 1. Definition of basic operators. The operators are assumed to operate on complex numbers but other base sets are possible. Boldface fonts represent vectors or matrices linearized in memory. Superscripts U and L represent the upper and lower half of a vector. A vector is sometimes written as $\mathbf{x} = (x_i)$ to identify the components.

MMM as inspiration about what to do

From the Wikipedia:



Index-wise definition

$$C_{ij} = \sum_{k=1}^{2,3} A_{ik} \times B_{kj}$$

Hiding indices i, j, k :

$$3 \xleftarrow{A} 2 \xleftarrow{B} 3$$

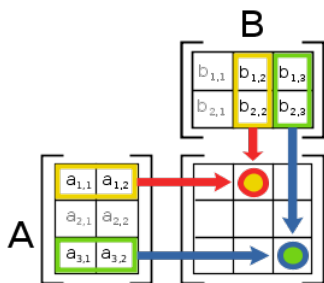
$$\xleftarrow{A \cdot B} 3$$

Index-free

$$C = A \cdot B$$

MMM as inspiration about what to do

From the Wikipedia:



Index-wise definition

$$C_{ij} = \sum_{k=1}^{2,3} A_{ik} \times B_{kj}$$

Hiding indices i, j, k :

$$\begin{array}{c}
 3 \xleftarrow{A} 2 \xleftarrow{B} 3 \\
 \xleftarrow{A \cdot B}
 \end{array}$$

Index-free

$$C = A \cdot B$$

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

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

$$\xleftarrow{A \cdot B} m$$

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

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

$$\xleftarrow{A \cdot B}$$

Category of matrices

As guessed above:

- Under MMM ($A \cdot B$), 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:

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

Transposition (converse)

As happens with relations, given $n \xrightarrow{A} m$ define

$$A^\circ = \begin{bmatrix} a_{11} & \dots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \dots & a_{mn} \end{bmatrix} \quad n \xleftarrow{A^\circ} m$$

Instead of telling how transposition is carried out index-wise, let us stress on its (index-free) properties such as, eg.

$$(A^\circ)^\circ = A \tag{1}$$

$$(A \cdot B)^\circ = B^\circ \cdot A^\circ \tag{2}$$

Bilinearity

Categories of matrices are **Abelian** — every homset forms an additive Abelian group (**Ab**-category) such that composition is bilinear relative to $+$:

$$M \cdot (N + L) = M \cdot N + M \cdot L \quad (3)$$

$$(N + L) \cdot K = N \cdot K + L \cdot K \quad (4)$$

Moreover, it has **biproducts**, where a biproduct diagram

$$\begin{array}{ccccc}
 & & \xleftarrow{\pi_1} & & \xrightarrow{\pi_2} \\
 a & & & c & & b \\
 & \xrightarrow{i_1} & & \xleftarrow{i_2} & &
 \end{array} \quad (5)$$

is such that

$$\pi_1 \cdot i_1 = id_a \quad (6)$$

$$\pi_2 \cdot i_2 = id_b \quad (7)$$

$$i_1 \cdot \pi_1 + i_2 \cdot \pi_2 = id_c \quad (8)$$

Deja vu?

In fact, within relations (where $+$ is \cup , π_1 is i_1° and π_2 is i_2°):

$$i_1^\circ \cdot i_1 = id$$

$$i_2^\circ \cdot i_2 = id$$

meaning that $i_{k=1,2}$ are injections (**kernels** both reflexive and coreflexive) and

$$i_1 \cdot i_1^\circ \cup i_2 \cdot i_2^\circ = id$$

meaning that they are jointly surjective (**images** together are reflexive and coreflexive).

Orthogonality

Projections and injections are orthogonal to each other:

$$\pi_1 \cdot i_2 = 0 \quad , \quad \pi_2 \cdot i_1 = 0 \quad (9)$$

Again something we can translate to relational algebra, for instance (recalling that $\pi_1 = i_1^\circ$, $\pi_2 = i_2^\circ$):

$$\begin{aligned} i_1^\circ \cdot i_2 &= \perp \\ \Leftrightarrow \quad &\{ \text{go pointwise and simplify} \} \\ &\neg \langle \exists b, a :: \langle \exists c :: i_1 c = i_2 c \rangle \rangle \end{aligned}$$

(injections are range-disjoint).

In linear algebra, however, **biproducts** are far many and more interesting!
Let us see why.

Biproduct = product + coproduct

Quoting MacLane (1971), pg. 194:

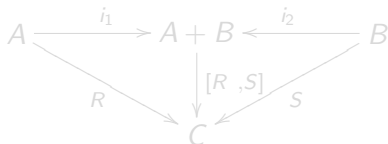
Theorem:

“Two objects a and b in Ab-category A have a **product** in A iff they have a biproduct in A . Specifically, given a biproduct diagram, the object c with the projections π_1 and π_2 is a product of a and b , while, dually, c with i_1 and i_2 is a **coproduct**.”

How do we build (c)products from biproducts?

The parallel with relation algebra helps once again (for $\pi_1 = i_1^\circ$ and $\pi_2 = i_2^\circ$):

$$[R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \quad \text{cf.}$$



Biproduct = product + coproduct

Quoting MacLane (1971), pg. 194:

Theorem:

“Two objects a and b in Ab-category A have a **product** in A iff they have a biproduct in A . Specifically, given a biproduct diagram, the object c with the projections π_1 and π_2 is a product of a and b , while, dually, c with i_1 and i_2 is a **coproduct**.”

How do we build (c)products from biproducts?

The parallel with relation algebra helps once again (for $\pi_1 = i_1^\circ$ and $\pi_2 = i_2^\circ$):

$$[R, S] = (R \cdot i_1^\circ) \cup (S \cdot i_2^\circ) \quad \text{cf.}$$

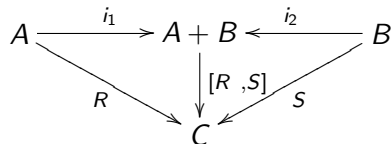
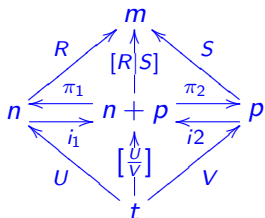


Diagram for (co)products

Diagram and definitions below depict how products and coproducts arise from biproducts:



$$[R|S] = R \cdot \pi_1 + S \cdot \pi_2 \quad (10)$$

$$\begin{bmatrix} U \\ V \end{bmatrix} = i_1 \cdot U + i_2 \cdot V \quad (11)$$

These are in fact families of (co)products, as there are many solutions to the biproduct equations. How do we go about such a *variety of solutions*?

Chasing biproducts

Hugo sought help from *Mathematica* by reducing dimensions as much as possible

$$1 \begin{array}{c} \xleftarrow{\pi_1} \\ \xrightarrow{i_1} \end{array} 1 + 1 \begin{array}{c} \xrightarrow{\pi_2} \\ \xleftarrow{i_2} \end{array} 1$$

thus leading to a more manageable “puzzle”

$$\begin{cases} \pi_1 \cdot i_1 & = & [1] \\ \pi_2 \cdot i_2 & = & [1] \\ i_1 \cdot \pi_1 + i_2 \cdot \pi_2 & = & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{cases}$$

(which still has several solutions)

Chasing biproducts

Fragment of *Mathematica* script:

```
sol = Solve[{pi1.i1 == I1, pi2.i2 == I1, i1.pi1 + i2.pi2 == I2}]
```

```
Solve::svars :
```

```
Equations may not give solutions for all "solve" variables. >>
```

$$\left\{ \left\{ w[1][1] \rightarrow \frac{1}{y[1][1]}, w[1][2] \rightarrow -\frac{x[1][1]z[1][2]}{y[1][1]}, x[1][2] \rightarrow \frac{1}{z[1][2]}, z[1][1] \rightarrow 0, y[1][2] \right. \right.$$

$$\left. \left\{ w[1][1] \rightarrow -\frac{x[1][2]z[1][1]}{y[1][2]}, w[1][2] \rightarrow \frac{y[1][2]+x[1][2]y[1][1]z[1][1]}{y[1][2]^2}, \right. \right.$$

$$\left. \left. x[1][1] \rightarrow \frac{y[1][2]+x[1][2]y[1][1]z[1][1]}{y[1][2]z[1][1]}, z[1][2] \rightarrow -\frac{y[1][1]z[1][1]}{y[1][2]} \right\} \right\}$$

Chasing biproducts

Among solutions given by *Mathematica* we picked

$$\begin{aligned}\pi_1 &= \begin{bmatrix} 1 & 0 \end{bmatrix} & \pi_2 &= \begin{bmatrix} 0 & 1 \end{bmatrix} \\ i_1 &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} & i_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}\end{aligned}$$

which purport an intuitive reading of *either* and *split*:

- $[A|B]$ glues two matrices horizontally
- $\begin{bmatrix} A \\ B \end{bmatrix}$ glues two matrices vertically

In general:

$$\begin{aligned}\pi_1 &= m \xleftarrow{[id_m \mid 0]} m+n & , & & \pi_2 &= n \xleftarrow{[0 \mid id_n]} m+n \\ i_1 &= m+n \xleftarrow{\begin{bmatrix} id_m \\ 0 \end{bmatrix}} m & , & & i_2 &= m+n \xleftarrow{\begin{bmatrix} 0 \\ id_m \end{bmatrix}} n\end{aligned}$$

The “standard” biproduct

Rephrased using (10) and (11) just defined, biproduct axiom (8) rewrites to both

$$[i_1 | i_2] = id \quad (12)$$

$$\begin{bmatrix} \pi_1 \\ \pi_2 \end{bmatrix} = id \quad (13)$$

telling how the two injections and the two projections “decompose” the identity matrix.

Moreover, (12,13) look like **reflection** laws (AoP terminology).

Thus the universal properties which follow:

The “standard” biproduct

Universal properties (familiar to the AoP practitioner) — one for “either”,

$$X = [R|S] \Leftrightarrow \begin{cases} X \cdot i_1 = R \\ X \cdot i_2 = S \end{cases} \quad (14)$$

another for “split”:

$$X = \begin{bmatrix} U \\ V \end{bmatrix} \Leftrightarrow \begin{cases} \pi_1 \cdot X = U \\ \pi_2 \cdot X = V \end{cases} \quad (15)$$

Converse duality

$$\begin{bmatrix} A \\ B \end{bmatrix} = [A^\circ | B^\circ]^\circ \quad (16)$$

Block notation

Block notation is nothing but but packaging products and coproducts together:

$$X = \left[\begin{array}{c|c} A & C \\ \hline B & D \end{array} \right] \Leftrightarrow \begin{cases} \pi_1 \cdot X \cdot i_1 = A \\ \pi_1 \cdot X \cdot i_2 = C \\ \pi_2 \cdot X \cdot i_1 = B \\ \pi_2 \cdot X \cdot i_2 = D \end{cases} \quad (17)$$

As expected, projection indices identify *lines*, injection indices identify *columns*.

Triggering the AoP panoply

(Besides reflection laws already mentioned)

Two **fusion** laws:

$$\left[\begin{array}{c} A \\ B \end{array} \right] \cdot C = \left[\begin{array}{c} A \cdot C \\ B \cdot C \end{array} \right] \quad (18)$$

$$C \cdot [A|B] = [C \cdot A|C \cdot B] \quad (19)$$

Four **cancellation**-laws:

$$[A|B] \cdot i_1 = A \quad , \quad [A|B] \cdot i_2 = B \quad (20)$$

$$\pi_1 \cdot \left[\begin{array}{c} A \\ B \end{array} \right] = A \quad , \quad \pi_2 \cdot \left[\begin{array}{c} A \\ B \end{array} \right] = B \quad (21)$$

Abide laws

The *either/split* **exchange** law:

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] = \left[\left[\begin{array}{c} A \\ C \end{array} \right] \mid \left[\begin{array}{c} B \\ D \end{array} \right] \right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] \quad (22)$$

— tells the equivalence between *row-major* and *column-major* matrix construction.

Two **blocked addition** laws,

$$[A|B] + [C|D] = [A + C|B + D] \quad (23)$$

$$\left[\begin{array}{c} A \\ B \end{array} \right] + \left[\begin{array}{c} C \\ D \end{array} \right] = \left[\begin{array}{c} A + C \\ B + D \end{array} \right] \quad (24)$$

for suitably typed A , B , C and D .

Putting things in motion

Elementary **divide and conquer** matrix multiplication:

$$[R|S] \cdot \begin{bmatrix} U \\ V \end{bmatrix} = R \cdot U + S \cdot V \quad (25)$$

Calculation:

$$\begin{aligned}
 & [R|S] \cdot \begin{bmatrix} U \\ V \end{bmatrix} \\
 = & \quad \{ (11) \} \\
 & [R|S] \cdot (i_1 \cdot U + i_2 \cdot V) \\
 = & \quad \{ \text{bilinearity (3)} \} \\
 & [R|S] \cdot i_1 \cdot U + [R|S] \cdot i_2 \cdot V \\
 = & \quad \{ \text{+-cancellation (20) twice} \} \\
 & R \cdot U + S \cdot V
 \end{aligned}$$

Putting things in motion

Blockwise MMM:

$$\left[\begin{array}{c|c} R & S \\ \hline U & V \end{array} \right] \cdot \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array} \right] = \left[\begin{array}{c|c} RA + SC & RB + SD \\ \hline UA + VC & UB + VD \end{array} \right] \quad (26)$$

Calculation:

$$\begin{aligned} & \left[\left[\begin{array}{c} R \\ \hline U \end{array} \right] \mid \left[\begin{array}{c} S \\ \hline V \end{array} \right] \right] \cdot \left[\left[\begin{array}{c} A \\ \hline C \end{array} \right] \mid \left[\begin{array}{c} B \\ \hline D \end{array} \right] \right] \\ = & \quad \{ \textit{either-fusion (19)} \} \\ & \left[\left[\left[\begin{array}{c} R \\ \hline U \end{array} \right] \mid \left[\begin{array}{c} S \\ \hline V \end{array} \right] \right] \cdot \left[\begin{array}{c} A \\ \hline C \end{array} \right] \mid \left[\left[\begin{array}{c} R \\ \hline U \end{array} \right] \mid \left[\begin{array}{c} S \\ \hline V \end{array} \right] \right] \cdot \left[\begin{array}{c} B \\ \hline D \end{array} \right] \right] \end{aligned}$$

Putting things in motion

= { divide and conquer (25) twice }

$$\left[\left[\frac{R}{U} \right] \cdot A + \left[\frac{S}{V} \right] \cdot C \mid \left[\frac{R}{U} \right] \cdot B + \left[\frac{S}{V} \right] \cdot D \right]$$

= { split-fusion (19) four times }

$$\left[\left[\frac{R \cdot A}{U \cdot A} \right] + \left[\frac{S \cdot C}{V \cdot C} \right] \mid \left[\frac{R \cdot B}{U \cdot B} \right] + \left[\frac{S \cdot D}{V \cdot D} \right] \right]$$

= { blocked addition (24) twice }

$$\left[\left[\frac{R \cdot A + S \cdot C}{U \cdot A + V \cdot C} \right] \mid \left[\frac{R \cdot B + S \cdot D}{U \cdot B + V \cdot D} \right] \right]$$

= { the same in block notation (22) }

$$\left[\frac{RA + SC}{UA + VC} \mid \frac{RB + SD}{UB + VD} \right]$$

No indices messing around :-)

Exploiting the biproduct space

What about other solutions to the biproduct equations? What can we expect from them?

Think of **Gaussian elimination**, for instance: main steps are row-switching, row-multiplication and row-addition, eg. transformation t for a given α :

$$t : (n \longleftarrow n) \times (n + n \longleftarrow m) \rightarrow (n + n \longleftarrow m)$$

$$t(\alpha, \begin{bmatrix} A \\ B \end{bmatrix}) = \begin{bmatrix} A \\ \alpha A + B \end{bmatrix}$$

(arrow $n \xleftarrow{\alpha} n$ means $n \xleftarrow{id} n$ with all 1s replaced by α .)

Another biproduct

Let us “reverse specify” t :

$$\begin{aligned}
 t(\alpha, \begin{bmatrix} A \\ B \end{bmatrix}) &= \begin{bmatrix} A \\ \alpha A + B \end{bmatrix} \\
 &= \{ \text{(26) in reverse order} \} \\
 &= \left[\begin{bmatrix} 1 \\ \alpha \end{bmatrix} \mid \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right] \cdot \begin{bmatrix} A \\ B \end{bmatrix} \\
 &= \{ \text{divide-and-conquer (25)} \} \\
 &= \begin{bmatrix} 1 \\ \alpha \end{bmatrix} \cdot A + \begin{bmatrix} 0 \\ 1 \end{bmatrix} \cdot B \tag{27}
 \end{aligned}$$

It can be shown that (27) is the *split* combinator of another biproduct, the one capturing such a step of Gaussian elimination:

$$\begin{aligned}
 \pi'_1 &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad , \quad \pi'_2 = \begin{bmatrix} -\alpha & 1 \\ 0 & 0 \end{bmatrix} \\
 i'_1 &= \begin{bmatrix} 1 \\ \alpha \end{bmatrix} \quad , \quad i'_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}
 \end{aligned}$$

Gaussian elimination “hylomorphism”

Structured and (“polymorphically”) typed:

$$ge : (1 + n \longleftarrow 1 + m) \rightarrow (1 + n \longleftarrow 1 + m)$$

$$ge \left[\begin{array}{c|c} x & M \\ \hline N & Q \end{array} \right] = \left[\begin{array}{c|c} x & M \\ \hline 0 & ge(Q - \frac{N}{x} \cdot M) \end{array} \right]$$

$$ge \ x = x$$

However: what’s the specification of ge ?

Currently studying its specification and correctness proof.

Last but not least — vectorization

Refinement step: linearization of an arbitrary matrix into a vector mapped on linear storage.

Refactoring SPIRAL's **OL** means studying the refinement of all matrix operations into vectorial form.

A foretaste of what is to come: DFT as an OL breakdown rule,

$$DFT_n \rightarrow (DFT_k \otimes I_m) \circ D_{k,m} \circ (I_k \otimes DFT) \circ L_k^{km}, \quad n = km$$

MMM as an OL breakdown rule,

$$\begin{aligned} \text{MMM}_{m,k,n} \rightarrow & (I_{m/m_b} \otimes L_{m_b}^{m_b n/n_b} \otimes I_{n_b}) \circ (\text{MMM}_{m/m_b, k/k_b, n/n_b} \otimes \text{MMM}_{m_b, k_b, n_b}) \\ & \circ ((I_{m/m_b} \otimes L_{k/k_b}^{m_b k/k_b} \otimes I_{k_b}) \times (I_{k/k_b} \otimes L_{n/n_b}^{k_b n/n_b} \otimes I_{n_b})) \end{aligned}$$

Strategy?

Last but not least — vectorization

Main observation — vectorization is akin to exponentiation:

While currying “things” the input of a given binary function $n \xleftarrow{f} mk$ by converting it into its unary (higher-order) counterpart $n^k \xleftarrow{\text{curry } f} m$, so does vectorization by thinning a given matrix $n \xleftarrow{M} km$ into $kn \xleftarrow{\text{vec } M} m$, where k is the “thinning factor”.

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

Last but not least — vectorization

Once again, let us rely on capturing such a relationship by an universal property:

$$X = \mathbf{vec} M \Leftrightarrow M = \epsilon \cdot (id \otimes X)$$

cf. diagram (analogue to that of *curry*)

$$\begin{array}{ccc}
 & k \times n & k \times (k \times n) \xrightarrow{\epsilon} n \\
 \mathbf{vec} M \uparrow & & \uparrow \\
 m & & id_k \otimes (\mathbf{vec} M) \\
 & & \uparrow \\
 & & k \times m \xrightarrow{M} n
 \end{array}$$

where \otimes denotes Kronecker product.

Last but not least — vectorization

This grants **vec** as a bijective transformation. So its converse **unvec** is also a bijection, whereby $\epsilon = \mathbf{unvec} \, id$, etc, etc

In other words, we are in presence of an adjunction between functor $FX = id_k \otimes X$ and itself.

Categories of matrices are not CCC but they are CSM (closed symmetric monoidal), yielding a tensor product (\otimes) which is a bifunctor with a monoidal structure

$$\otimes : Mat_k \times Mat_k \rightarrow Mat_k$$

Exploring all this in calculating the whole algebra of OL vectorized operations will keep us (HM+JNO) busy for a while.

Franz Franchetti, Frédéric de Mesmay, Daniel McFarlin, and Markus Püschel. Operator language: A program generation framework for fast kernels. In *IFIP Working Conference on Domain Specific Languages (DSL WC)*, 2009.

S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New-York, 1971.