Towards a Linear Algebra of Programming

J.N. Oliveira

HASLab — High Assurance Software Lab
INESC TEC and University of Minho, Portugal

IFIP WG2.1 meeting #68
February 2012
Rome, Italy

Motivation

Formal methods are going quantitative — further to predicting "**it may happen**" one wants to know "**how often it will happen**"

As happened with physics in the past, computer science is becoming **probabilistic**

Probability theory particularly relevant in **security** analysis of information flow

Haskell library **PFP** for probabilistic functional programming — distribution monad, etc (Erwig and Kollmansberger, 2006)

A new probability perspective

However, traditional notation for probabilities is too **descriptive** and not meant for calculation as we understand it today.

Quoting Hehner (2011) — also wg2.1#64:

> *Perhaps a thousand years ago the philosophers of the time [might give] reasons why their answer is right. Now we* **don't argue***; we formalize,* **calculate***, and unformalize.*

From a functional programming perspective, one may identify

probabilistic functions

as something half way between **relations** and traditional **functions**: they express the propensity, or likelihood of ambiguous (or "bad") outputs.

Example: fault-injected multiplication

Safe multiplication (over $\mathbb{N}_0$):

$$(a*) = \textbf{for}\ (a+)\ 0$$

that is,

$$a * 0 = 0$$
$$a * (n + 1) = a + a * n$$

Bad multiplication, **fault-injected** — 5% probability of a wrong base case (in extended functional notation):

$$a * 0 =_{.95} 0$$
$$a * 0 =_{.05} a$$
$$a * (n + 1) =_1 a + a * n$$

Example: fault-injected multiplication

The same in Haskell's PFP library (Erwig and Kollmansberger, 2006):

```
a * 0 = D [(0,0.95),(a,0.05)]
a * (n+1) = do x <- a * n
               return $ a + x
```

monadic over distributions:

```
newtype Dist a = D {unD :: [(a,ProbRep)]}

instance Monad Dist where
  return x = D [(x,1)]
  d >>= f  = D [(y,q*p) | (x,p) <- unD d, (y,q) <- unD (f x)]
  fail _   = D []
```

Question: does the fault in the base case carry over to the overall function? In what extent? (**Quantify** fault propagation.)

Non-deterministic functions

Nondeterministic outputs — set-valued functions are relations

$$f = \Lambda R \quad \Leftrightarrow \quad \langle \forall\ b, a\ ::\ b\ R\ a \Leftrightarrow b \in f\ a \rangle \tag{1}$$

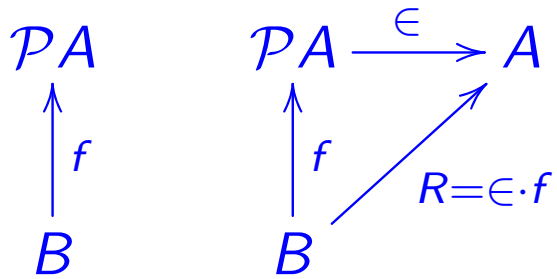
that is,

$$A \to \mathcal{P} B \underset{\Lambda}{\overset{(\in \cdot)}{\cong}} A \to B \tag{2}$$

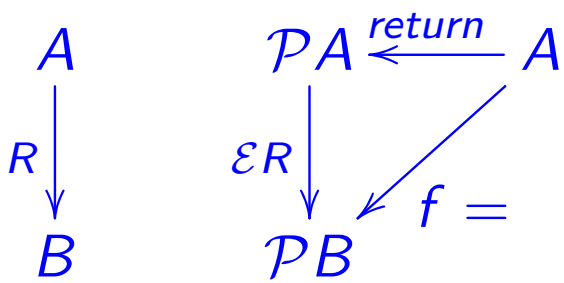
where $A \to B$ on the right hand side is the **relational type** $A \to B$ of all relations $R \subseteq B \times A$.

Nondeterministic functions

An adjunction, offering two ways for reasoning — one relational (**Rel**)



the other monadic (**Set**):



$$\mathcal{E}R \cdot return$$

$$where\ (\mathcal{E}R)s = \{b | a \leftarrow s; bRa\}$$

The same choice in going probabilistic — **monadic**? **what else**?

Probabilistic functions

Outputs become distributions,

$$A \to \mathcal{D}B \quad \cong \quad A \to B \tag{3}$$

where $\mathcal{D}B$ is the B-distribution data type

$$\mathcal{D}B \quad = \quad \{\mu \in [0,1]^B \mid \sum_{b \in B} \mu\ b = 1\} \tag{4}$$

and where $[0,1]$ is the interval of all non-negative reals at most 1.

However, what does $A \to B$ on the right hand side of (3) mean?

Probabilistic functions

One has:

$$
\begin{array}{cc}
 & A \to [0,1]^B \\
\Leftrightarrow & \{ \text{ uncurrying } \} \\
 & A \times B \to [0,1] \\
\Leftrightarrow & \{ \text{ swapping } \} \\
 & B \times A \to [0,1]
\end{array}
$$

where $B \times A \to [0,1]$ can be identified with the set of all **matrices** taking elements from $[0,1]$ with as many **columns** (resp. **rows**) as elements in A (resp. B).

Left stochastic matrices

In fact:

$$A \to \mathcal{D}B \quad \cong \quad A \to_{LS} B \tag{5}$$

where LS denotes the **category** of **left-stochastic** matrices (columns in such matrices add up to 1).

Such a **matrix**-transform is captured by the universal property, for all $f :: A \to \mathcal{D}B$ and LS-matrix M:

$$M = [\![f]\!] \quad \Leftrightarrow \quad \langle \forall\ b, a\ ::\ M(b, a) = (f\ a)b \rangle \tag{6}$$

Research question:

> *Is LS "as useful" to probabilistic reasoning as Rel is to non-deterministic reasoning in the AoP (Bird and de Moor, 1997) ?*

Towards a LAoP

My answer:

I believe so.

But — many things to be explained:

- **categories of matrices** — what's this?
- category of **LS matrices** — what's this?
- the **AoP** is pointfree — universal property (6) above is pointwise...

Answering these questions will generalize the **AoP** into something one may identify as a **Linear** Algebra of Programming (**LAoP**).

Linear algebra for software verification

Could not agree more on...

> *"(...) our key idea is to adopt linear algebra as the lingua franca of software verification"*

quoted from

> *LAP: Linear Algebra of bounded resources Programs*

— a project of SQIG at the Telecommunications Institute (IT) in Lisbon (`http://sqig.math.ist.utl.pt/work/LAP`).

However — **old-fashioned** matrix calculus needs to be **spruced up**... **;-)**

Uups!

"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)

"I cannot believe that anything so **ugly** *as multiplication of matrices is an essential part of the scheme of nature"*

Sir Arthur Eddington (1936)

— Thanks, Jeremy, for this quote :-)

Linear algebra in computer science

Trend towards **quantitative methods** in computer science (using **LA** in particular):

- Read Baroni and Zamparelli (2010) suggestive paper: *Nouns are vectors, adjectives are matrices* in semantics of natural language.
- "Quantum inspiration" in Sernadas et al. (2008) who regard probabilistic programs as linear transformations over suitable vector spaces.

Our own trend: PhD thesis by Hugo Macedo (2012) — just submitted — entitled

"Matrices as Arrows" — Why Categories of Matrices Matter"

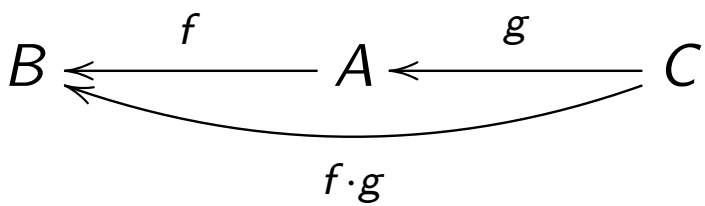
Arrow notation for functions

Used everywhere for declaring **functions**, eg.

$$\begin{array}{lcl} f & : & \mathbb{N} \to \mathbb{R} \\ n & \mapsto & \frac{n}{\pi} \end{array}$$

The first line is the **type** of the function (**syntax**) and the second line is the rule of correspondence (**semantics**).

Compositionality — functions compose with each other:



(7)

$$b = f(g\ c)$$

Arrow notation for (binary) relations

Binary relations are typed: arrow $A \xrightarrow{R} B$ denotes a binary relation from A (source) to B (target).

A, B are types. Writing $B \xleftarrow{R} A$ means the same as $A \xrightarrow{R} B$.

Compositionality — relations compose with each other:

$$B \xleftarrow{R} A \xleftarrow{S} C \qquad R \cdot S \tag{8}$$

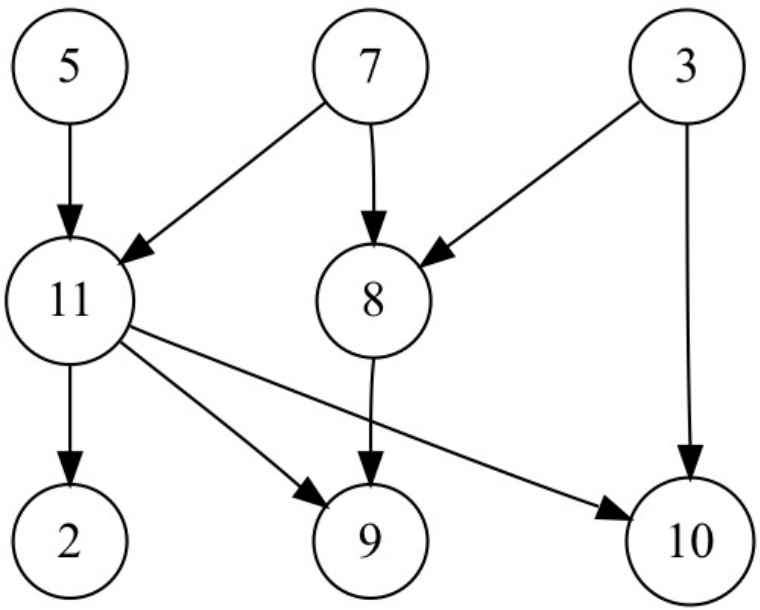
$$b(R \cdot S)c \;\Leftrightarrow\; \langle \exists\ a \;::\; b\ R\ a \wedge a\ S\ c \rangle \tag{9}$$

Example: $uncle = brother \cdot parent$

From binary relations to matrices

As binary relations are Boolean matrices, eg.

Relation R:



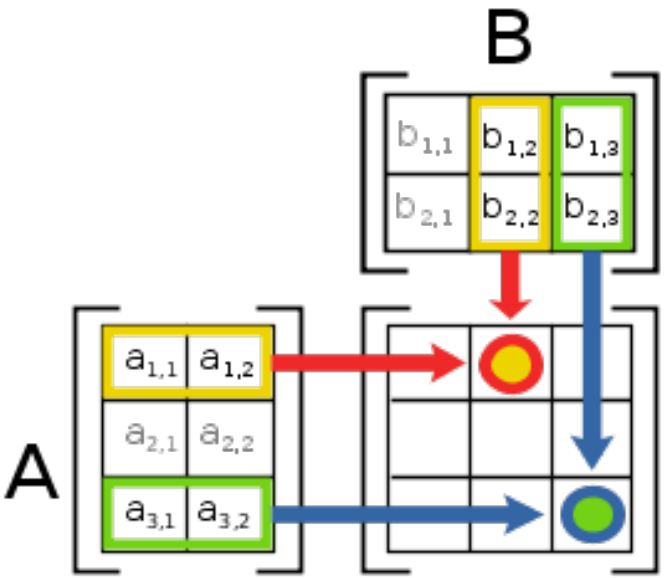
Matrix M:

	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	1
10	0	0	1	0	0	0	0	0	0	0	1
11	0	0	0	0	1	0	1	0	0	0	0

why not represent matrices as arrows too, cf.

$$11 \xleftarrow{M} 11 \ ?$$

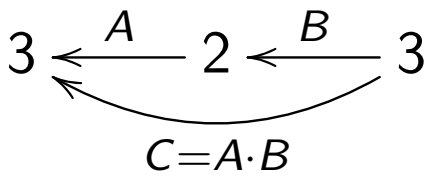
Compositionality — matrix-matrix multiplication



Index-wise definition

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

Arrows hide indices i, j, k:

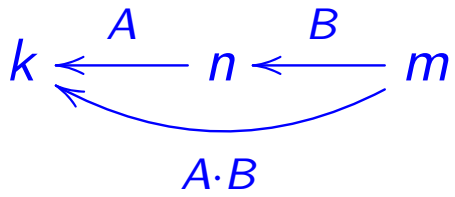


Index-free

$$C = A \cdot B$$

Typed linear algebra

Composition of matrices obeys to the **typing rule**



For matrices A and B **of the same type** $n \longleftarrow m$, we can extend cell level algebra to matrix level, eg. by **adding** and **subtracting** matrices,

$$A + B \quad , \quad A - B$$

multiplying matrices (the Hadamard product)

$$A \times B$$

and so on.

Typed linear algebra

Expressions such as eg. $A + B$, $A \times B$ for A and B of different types **won't typecheck**.

The underlying type system is **polymorphic** and type inference proceeds by **unification**, as in programming languages.

For instance, the **identity matrix**

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

is polymorphic on type n. This could help in equipping tools such as MATLAB and MATHEMATICA with a type system saving the burden of always checking for matrix dimensions.

Converse

Given matrix $n \xleftarrow{M} m$, notation $m \xleftarrow{M^\circ} n$ denotes its transpose, or converse.

Operation whereby M changes shape by turning its rows into columns and vice-versa.

The following idempotence and contravariance laws hold:

$$(M^\circ)^\circ = M \tag{10}$$

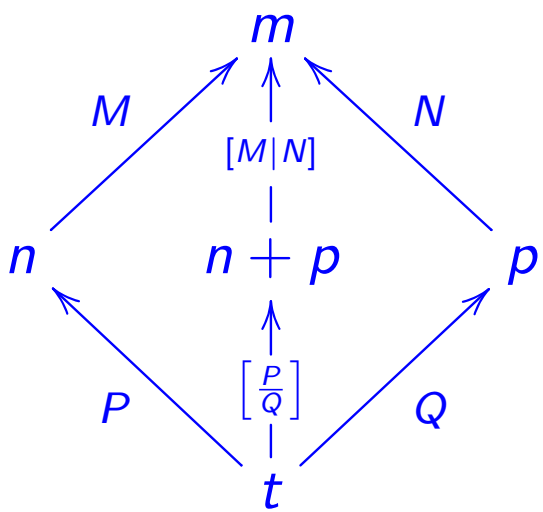
$$(M \cdot N)^\circ = N^\circ \cdot M^\circ \tag{11}$$

Polymorphic (block) combinators

Two ways of putting matrices together to build larger ones:

- $X = [M|N]$ — M and N side by side (“‘junc”)
- $X = \left[\frac{P}{Q}\right]$ — P on top of Q (“‘split”).

Mind the (polymorphic) types:



(A biproduct)

Blocked linear algebra

Rich set of laws, for instance

- **Divide-and-conquer:**

$$[A|B] \cdot \left[\frac{C}{D}\right] = A \cdot C + B \cdot D \tag{12}$$

- **"Fusion"-laws:**

$$C \cdot [A|B] = [C \cdot A|C \cdot B] \tag{13}$$

$$\left[\frac{A}{B}\right] \cdot C = \left[\frac{A \cdot C}{B \cdot C}\right] \tag{14}$$

Block-matrix-algebra is at the heart of **parallelism** in **LA** — this is why **LA**-based approaches are so important today.

Vectors

Vectors are special cases of matrices in which one of the types 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.

Special matrices

The following (Boolean) matrices are relevant:

- The **bottom** matrix $n \xleftarrow{\bot} m$ — wholly filled with 0s
- The **top** matrix $n \xleftarrow{\top} m$ — wholly filled with 1s
- The **identity** matrix $n \xleftarrow{id} n$ — diagonal of 1s
- The **bang** (row) vector $1 \xleftarrow{!} m$ — wholly filled with 1s

Thus, (typewise) **bang** matrices are special cases of **top** matrices:

$$1 \xleftarrow{\top} m \quad = \quad !$$

Also note that, on type $1 \xleftarrow{} 1$:

$$\top \;=\; ! \;=\; id$$

Type generalization

As is standard is **relational mathematics** (Schmidt, 2010), matrix types can be generalized from numeric dimensions (n, $m \in \boldsymbol{N}_0$) to arbitrary denumerable types (X, Y), taking **disjoint union** $X + Y$ for $m + n$, Cartesian product $X \times Y$ for mn, etc.

We will restrict ourselves to matrices taking elements from $\boldsymbol{R}_0^+$, the non-negative reals.

The interval $[0, 1]$ will be particularly at target in the case of **probabilistic** functions.

Doing elementary set theory with LA

Clearly, any **Boolean** vector of type $1 \xleftarrow{!} X$ for some type X represents a subset of X, for instance

$$\begin{array}{c} \\ \mathbf{1} \end{array} \begin{array}{c} \begin{array}{ccc} \mathbf{John} & \mathbf{Mary} & \mathbf{Henry} \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \end{pmatrix} \end{array}$$

represents set $\{John, Henry\} \subseteq \{John, Mary, Henry\}$.

Let $[\![A]\!]$ denote the vector which represents $A \subseteq X$. Then:

$$\begin{aligned} [\![\emptyset]\!] = \bot \quad &, \quad [\![A]\!] = ! \\ [\![A \cap B]\!] &= [\![A]\!] \times [\![B]\!] \\ [\![\neg A]\!] &= ! - [\![A]\!] \\ [\![A \cup B]\!] &= [\![A]\!] + [\![B]\!] - [\![A \cap B]\!] \end{aligned} \tag{15}$$

Example of LA set theory calculation

State that A and B form a **partition** of X simply by writing

$$[\![A]\!] + [\![B]\!] = !$$

Infer that, necessarily, A and B are disjoint:

$$[\![A]\!] \times [\![B]\!] = \bot$$

For this, we only need to recall why **LA** is referred to as **linear**:

$$M \times (N + P) = M \times N + M \times P \quad (16)$$

$$M \cdot (N + P) = M \cdot N + M \cdot P \quad (17)$$

$$(N + P) \cdot M = N \cdot M + P \cdot M \quad (18)$$

etc, and drop the $[\![\_]\!]$ parentheses for saving ink:

Example of LA set theory calculation

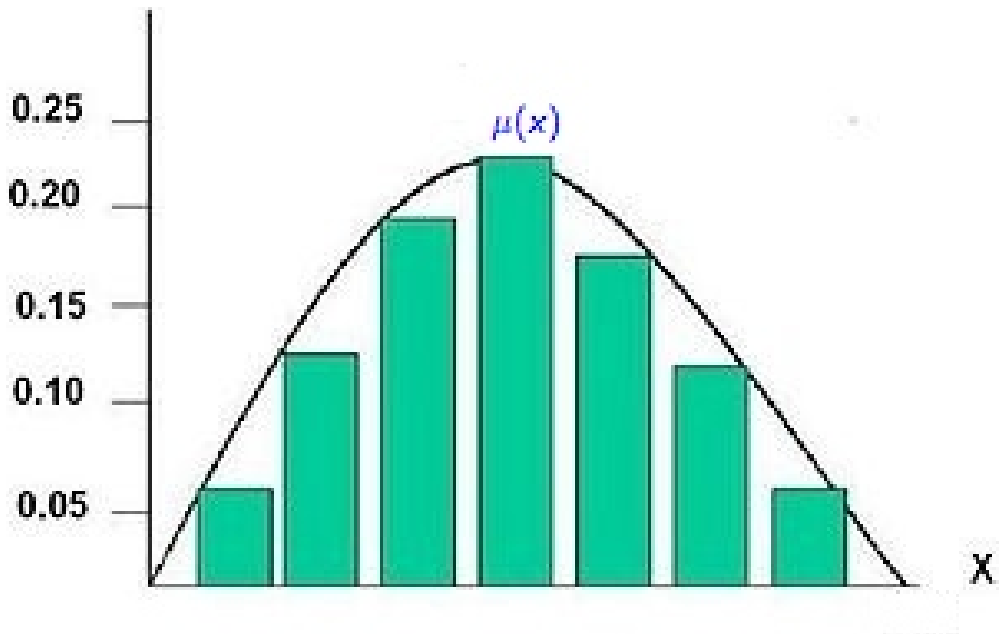
$$
\begin{array}{ll}
 & A + B = ! \\
\Rightarrow & \quad \{ \text{ multiply both sides by } A \text{ (Leibniz) } \} \\
 & A \times (A + B) = A \times ! \\
\Leftrightarrow & \quad \{ \text{ Hadamard: linearity (16), } A \times A = A \cap A = A \text{ and unit } ! \ \} \\
 & A + A \times B = A \\
\Leftrightarrow & \quad \{ \text{ subtract } A \text{ from both sides of the equation } \} \\
 & A \times B = A - A \\
\Leftrightarrow & \quad \{ \text{ cancellation of inverses } \} \\
 & A \times B = \bot \\
\Leftrightarrow & \quad \{ \text{ definition } \} \\
 & A \text{ and } B \text{ are disjoint}
\end{array}
$$

Doing elementary probability theory in LA

Recall standard definition of a **distribution**:

$$\mathcal{D}X \quad = \quad \{\mu : X \to [0,1] \mid \sum_{x \in X} \mu(x) = 1\} \tag{19}$$

as, for example,



Elementary probability theory in LA

The same in matrix format (for **discrete** μ) is column vector

$$X \xleftarrow{[\![\mu]\!]} 1$$

(over non-negative reals) such that $! \cdot [\![\mu]\!] = !$.

Why? The **sum** of all entries of vector $X \xleftarrow{v} 1$ is vector (cell)

$$1 \xleftarrow{!} X \xleftarrow{v} 1 \qquad !\cdot v$$

and $1 \xleftarrow{!} 1$ is real number 1.

Thus the pointfree style starts to happen (no need for $\sum$ etc)

Elementary probability theory in LA

Let $X \xrightarrow{\mu} [0,1]$ be a **discrete** probability **distribution** over sample space X.

The probability $P(S)$ of event $S \subseteq X$ under μ will be given by

$$P(S) = 1 \xleftarrow{[\![S]\!]} S \xleftarrow{[\![\mu]\!]} 1 \qquad (20)$$

$$[\![S]\!] \cdot [\![\mu]\!]$$

Then a *random variable* $T \xleftarrow{v} X$ on probability space (X, μ) induces a new probability distribution on T by composition,

$$[\![\mu']\!] = [\![v]\!] \cdot [\![\mu]\!] \qquad (21)$$

generating a new probability space (T, μ').

Elementary probability theory in LA

That μ' is indeed a distribution can be easily calculated:

$$
\begin{array}{ll}
 & ! \cdot [\![\mu']\!] \\
= & \quad \{ \ (21) \ \} \\
 & ! \cdot ([\![v]\!] \cdot [\![\mu]\!]) \\
= & \quad \{ \ v \text{ is a function, therefore Dirac-probabilistic} \ \} \\
 & ! \cdot [\![\mu]\!] \\
= & \quad \{ \ \mu \text{ is a distribution} \ \} \\
 & !
\end{array}
$$

Elementary probability theory in LA

Easy to get basic probability theory from the **LA** encoding, for instance the **addition law** of probability

$$P(A \cup B) \quad = \quad P(A) + P(B) - P(A \cap B)$$

coming straight from vectorial set union (15) and linearity (18).

Example: calculating the law of **total probability**

$$P(A) \quad = \quad P(A \cap B_1) + P(A \cap B_2)$$

where $\{B_1, B_2\}$ form a partition ($B_1 + B_2 = !$) of the sample space X (next slide).

Law of total probability

$$
\begin{aligned}
& P(A \cap B_1) + P(A \cap B_2) \\
= & \quad \{ \text{ (20) twice } \} \\
& (A \times B_1) \cdot \mu + (A \times B_2) \cdot \mu \\
= & \quad \{ \text{ composition and Hadamard product are bilinear } \} \\
& (A \times (B_1 + B_2)) \cdot \mu \\
= & \quad \{ \text{ partition : } B_1 + B_2 = ! \ \} \\
& (A \times !) \cdot \mu \\
= & \quad \{ \ ! = \top \text{ is the unit of Hadamard } \times \text{; definition (20) } \} \\
& P(A)
\end{aligned}
$$

Matrix transformed probabilistic functions

Recall that, given probabilistic function $A \xrightarrow{f} \mathcal{D}B$, its matrix **transform** is a matrix of type $A \xrightarrow{[\![f]\!]} B$ such that

$$! \cdot [\![f]\!] \quad = \quad ! \tag{22}$$

that is, all **columns** of $[\![f]\!]$ add up to one.

For $A = B$, probabilistic function f can be regarded as a **Markov chain**.

Example — probabilistic **negation**:

$$\begin{array}{c} \\ \textbf{True} \\ \textbf{False} \end{array} \begin{array}{c} \begin{array}{cc} \textbf{True} & \textbf{False} \end{array} \\ \begin{pmatrix} 0.1 & 0.8 \\ 0.9 & 0.2 \end{pmatrix} \end{array}$$

Linear algebra of probabilistic functions

Every sharp function is probabilistic — it offers the **Dirac distribution** for every input. This includes the identity function id represented by the identity matrix $[\![id]\!]$.

Compositionality: probabilistic functions compose, under monad-flavoured definition

$$[\![f \bullet g]\!] \quad = \quad [\![f]\!] \cdot [\![g]\!] \tag{23}$$

In monad-speak:

$$[\![\lambda \mathtt{a}.\ \mathbf{do}\ \{\mathtt{b} \leftarrow\ \mathtt{g}\ \mathtt{a}; \mathtt{f}\ \mathtt{b}\}\ ]\!] \quad = \quad [\![f]\!] \cdot [\![g]\!]$$

Let us not forget checking the 100% constraint (22).

$$
\begin{array}{ll}
 & f \bullet g \text{ is a probabilistic function} \\
\Leftrightarrow & \quad \{ \ (22) \ \} \\
 & ! \cdot [\![f \bullet g]\!] = ! \\
\Leftrightarrow & \quad \{ \ \text{definition (23)} \ \} \\
 & ! \cdot [\![f]\!] \cdot [\![g]\!] = ! \\
\Leftrightarrow & \quad \{ \ f \text{ is probabilistic (22)} \ \} \\
 & ! \cdot [\![g]\!] = ! \\
\Leftrightarrow & \quad \{ \ g \text{ is probabilistic (22)} \ \} \\
 & ! = !
\end{array}
$$

Probabilistic "junc"

Probabilistic $A + B \xrightarrow{[f,g]} \mathcal{D}C$ — run either f or g — transposes into

$$[\![[f,g]]\!] \quad = \quad [[\![f]\!]|[\![g]\!]] \tag{24}$$

where (recall) $[M|N]$ denotes M and N put **side by side**.

Checking the 100% constraint (22):

$$\begin{array}{ll} & !\cdot[[\![f]\!]|[\![g]\!]] \\ \Leftrightarrow & \quad \{ \text{ fusion-+ (13) } \} \\ & [!\cdot[\![f]\!]|!\cdot[\![g]\!]] \\ \Leftrightarrow & \quad \{ \text{ } f \text{ and } g \text{ probabilistic (22) } \} \\ & [!|!] \\ \Leftrightarrow & \quad \{ \text{ } [!|!] = ! \text{ } \} \\ & ! \end{array}$$

Probabilistic choice

In their programming language *pGCL*, McIver and Morgan (2005) introduce notation

$$prog \ {}_p\!\diamond \ prog'$$

as a form of **probabilistic choice** between two branches of a program, $prog$ chosen with probability p and $prog'$ with probability $1 - p$.

This corresponds to the choice between two probabilistic functions f and g **of the same type** defined by

$$[\![f \ {}_p\!\diamond \ g]\!] \quad = \quad p[\![f]\!] + (1-p)[\![g]\!] \tag{25}$$

Probabilistic choice

Probabilistic choice "is probabilistic":

$$
\begin{aligned}
& ! \cdot [\![f \ {}_p\diamond\ g]\!] \\
= & \quad \{ \text{ definition (25) ; bilinearity } \} \\
& ! \cdot (p[\![f]\!]) + ! \cdot ((1-p)[\![g]\!]) \\
= & \quad \{ \text{ } p \text{ is a scalar } \} \\
& p(! \cdot [\![f]\!]) + (1-p)(! \cdot [\![g]\!]) \\
= & \quad \{ \text{ } f \text{ and } g \text{ are probabilistic } \} \\
& p! + (1-p)! \\
= & \quad \{ \text{ bilinearity } \} \\
& (p + 1 - p)! \\
= & \quad \{ \text{ cancellation } \} \\
& !
\end{aligned}
$$

Properties

Probabilistic choice enjoys many properties easy to derive from the definition, eg. basic

$$f \;{}_p\!\diamond\; f = f \tag{26}$$
$$f \;{}_0\!\diamond\; g = g \tag{27}$$
$$f \;{}_p\!\diamond\; g = g \;{}_{1-p}\!\diamond\; f \tag{28}$$

fusion-laws

$$(f \;{}_p\!\diamond\; g) \bullet h = (f \bullet h) \;{}_p\!\diamond\; (g \bullet h) \tag{29}$$
$$h \bullet (f \;{}_p\!\diamond\; g) = (h \bullet f) \;{}_p\!\diamond\; (h \bullet g) \tag{30}$$

and the **exchange** law:

$$[f, g] \;{}_p\!\diamond\; [h, k] = [f \;{}_p\!\diamond\; h, g \;{}_p\!\diamond\; k] \tag{31}$$

Probabilistic sums

The **direct sum** of two matrices,

$$M \oplus N = [i_1 \cdot M | i_2 \cdot N] = \left[\frac{M \cdot \pi_1}{N \cdot \pi_2}\right] = \left[\begin{array}{c|c} M & 0 \\ \hline 0 & N \end{array}\right] \quad (32)$$

which has type $\begin{array}{ccc} A & B & A+B \\ {\scriptstyle M}\downarrow & {\scriptstyle N}\downarrow & \downarrow{\scriptstyle M \oplus N} \\ C & D & C+D \end{array}$ (a **bifunctor**) enables us to sum probabilistic functions:

$$[\![f \oplus g]\!] = [\![f]\!] \oplus [\![g]\!]$$

Distribution over choice

$$h \oplus (f \; {}_p\diamond \; g) = (h \oplus f) \; {}_p\diamond \; (h \oplus g) \quad (33)$$

is central to probabilistic function calculation.

Probabilistic recursion

Recall the algorithm for performing **multiplication** by **iterated addition** (over $\mathbb{N}_0$),

$$(a*) = \textbf{for}\ (a+)\ 0$$

which we wrote in two clauses

$$a * 0 = 0$$
$$a * (n + 1) = a + a * n$$

earlier on, only to inject a fault into the base case:

$$a * 0 =_{.95} 0$$
$$a * 0 =_{.05} a$$
$$a * (n + 1) =_{1} a + a * n$$

Probabilistic recursion

Now that we have probabilistic choice (${}_p\diamond$) we can encode the faulty version as follows (also parametric on the probability p of the fault),

$$faulty = \textbf{for}\ (a+)\ (0\ {}_p\diamond\ a)$$

and **try and quantify** how bad it is if compared to the original version,

$$good = \textbf{for}\ (a+)\ 0$$

or to one definitely bad:

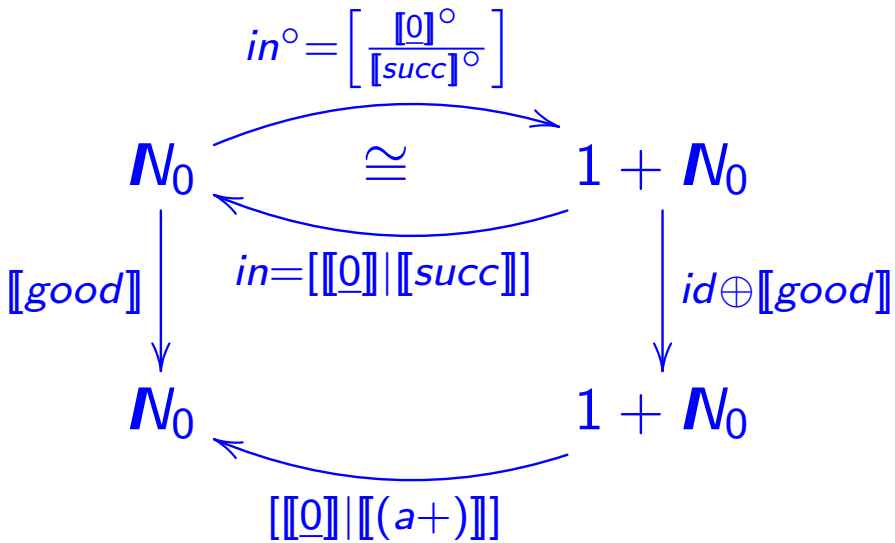
$$bad = \textbf{for}\ (a+)\ a$$

(Matrix) catamorphisms

Note the slight abuse of notation, as we don't have a choice for values — only for functions. It should be:

$$faulty = (\!|[\underline{0}\ {}_p\diamond\ \underline{a}|(a+)]|\!)$$
$$bad = (\!|[\underline{a}|(a+)]|\!)$$
$$good = (\!|[\underline{0}|(a+)]|\!)$$

Here is how *good* looks like in a diagram:



(Matrix) catamorphisms

Parsing the diagram, one has:

$$[\![good]\!] = [[\![\underline{0}]\!]|[\![(a+)]\!]] \cdot (id \oplus [\![good]\!] \cdot \left[\frac{[\![\underline{0}]\!]^\circ}{[\![succ]\!]^\circ}\right]$$

$$= \quad \{ \text{ absorption (34) ; dropping parentheses for better parsing } \}$$

$$good = [\underline{0}|(a+) \cdot good] \cdot \left[\frac{\underline{0}^\circ}{succ^\circ}\right]$$

$$= \quad \{ \text{ divide and conquer (12) } \}$$

$$good = \underline{0} \cdot \underline{0}^\circ + (a+) \cdot good \cdot succ^\circ$$

Note how the matrix for *good* is recursively filled up: first the outer-product $\underline{0} \cdot \underline{0}^\circ$ (that is, the everywhere-0 matrix apart from the 1 in cell $(0,0)$), which is added to $(a+) \cdot \underline{0} \cdot \underline{0}^\circ \cdot succ^\circ = \underline{a} \cdot \underline{1}^\circ$ (matrix with a 1 in cell labeled $(a,1)$ and 0 otherwise), and so on.

Checking assertion

Conjecture: is it true that

$$faulty = good \ {}_p\diamond \ bad$$

We reason:

$$\begin{array}{ll} & faulty = good \ {}_p\diamond \ bad \\ \Leftrightarrow & \quad \{ \text{ definition of } faulty \ \} \\ & (\!|[\underline{0} \ {}_p\diamond \ \underline{a}|(a+)]|\!) = good \ {}_p\diamond \ bad \\ \Leftrightarrow & \quad \{ \text{ cata-universal property, for } \mathsf{F}X = id \oplus X \ \} \\ & (good \ {}_p\diamond \ bad) \cdot [\underline{0}|succ] = [\underline{0} \ {}_p\diamond \ \underline{a}|(a+)] \cdot \mathsf{F}(good \ {}_p\diamond \ bad) \end{array}$$

The calculation of this equality will resort to $\oplus$-absorption:

$$[M|N] \cdot (P \oplus Q) \quad = \quad [M \cdot P|N \cdot Q] \qquad (34)$$

Checking assertion

$$
\begin{aligned}
& (good \ {}_p\diamond \ bad) \cdot [\underline{0}|succ] \\
= & \quad \{ \ {}_p\diamond\text{-fusion (29)} \ \} \\
& (good \cdot [\underline{0}|succ]) \ {}_p\diamond \ (bad \cdot [\underline{0}|succ]) \\
= & \quad \{ \ \text{cata-cancellation (twice: } good \text{ and } bad) \ \} \\
& ([\underline{0}|(a+)] \cdot \mathsf{F}good) \ {}_p\diamond \ ([\underline{a}|(a+)] \cdot \mathsf{F}bad) \\
= & \quad \{ \ \text{absorption (34) over } \mathsf{F}X = id \oplus X \ \} \\
& [\underline{0}|(a+) \cdot good] \ {}_p\diamond \ [\underline{a}|(a+) \cdot bad] \\
= & \quad \{ \ \text{exchange-law: } [f,g] \ {}_p\diamond \ [h,k] = [f \ {}_p\diamond \ h, g \ {}_p\diamond \ k] \ (31) \ \} \\
& [\underline{0} \ {}_p\diamond \ \underline{a} \mid ((a+) \cdot good) \ {}_p\diamond \ ((a+) \cdot bad)]
\end{aligned}
$$

Checking assertion

$$
\begin{array}{ll}
= & \{\ \text{fusion (30)}\ \} \\
& [\underline{0}\ {}_p\diamond\ \underline{a} \mid (a+)\cdot(good\ {}_p\diamond\ bad)] \\
= & \{\ \oplus\text{-absorption (34), in the reverse direction}\ \} \\
& [\underline{0}\ {}_p\diamond\ \underline{a}|(a+)]\cdot \mathsf{F}(good\ {}_p\diamond\ bad) \\
\square &
\end{array}
$$

Note how exchange-law (31)

$$[f, g]\ {}_p\diamond\ [h, k] = [f\ {}_p\diamond\ h, g\ {}_p\diamond\ k]$$

plays the central role above, on top of bilinearity.

Closing

The research question which motivated this talk splits in two other questions, in fact two sides of the same coin:

(a) Can **the** AoP be extended quantitatively in some useful way?
(b) What happens to the discipline once we generalize from relations to matrices?

The answer leads us into **linear algebra**, which eventually provides a surprisingly simple framework for calculating with **set-theory**, **probabilities**, functions and relations, provided it is **typed** — as advocated by Macedo (2012).

Closing

The comment by Sir Arthur Eddington in his *Relativity Theory of Electrons and Protons* (already quoted) can be understood as a call for better laid out **linear algebra** — perhaps **typed** :-)?

Is this kind of foundation sought in 1967, in the Garmisch NATO workshop:

> *In late 1967 the Study Group recommended the holding of a working conference on Software Engineering. The phrase 'software engineering' was deliberately chosen as being* **provocative**, *in implying the need for software manufacture to be based on the types of* **theoretical foundations** *and practical disciplines, that are traditional in the established branches of engineering. (Naur and Randell, 1969)*

Only **time** and **experience** will tell.

Future work

Topic in its very start (very recent research, a couple of months), much to be done:

- **LAoP** needs to be pushed forward to realistic case-studies.
- Typed LA framework applied to **OLAP** and data-mining — follow up of work already carried out by (Macedo and Oliveira, 2011). Links to **parallel programming**
- Extension of the **GNU Octave** (`http://www.gnu.org/software/octave`) matrix sub-language with **types** and a **type checker** along the lines of this talk.

References

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

M. Baroni and R. Zamparelli. Nouns are vectors, adjectives are matrices: representing adjective-noun constructions in semantic space. In *Proceedings*, EMNLP '10, pages 1183–1193, Morristown, NJ, USA, 2010. Association for Computational Linguistics.

R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

Arthur Eddington. *Relativity Theory of Electrons and Protons*. Cambridge University Press, 1936.

M. Erwig and S. Kollmansberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16: 21–34, January 2006.

E. Hehner. A probability perspective. *Formal Aspects of Computing*, 23:391–419, 2011.

H. Macedo. *Matrices as Arrows — Why Categories of Matrices Matter*. PhD thesis, University of Minho, 2012. (Submitted Jan. 2012).

H.D. Macedo and J.N. Oliveira. Do the middle letters of "OLAP" stand for linear algebra ("LA")? Technical Report TR-HASLab:04:2011, INESC TEC and University of Minho, Gualtar Campus, Braga, 2011.

A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems*. Monographs in Computer Science. Springer-Verlag, 2005. ISBN 0387401156.

P. Naur and B. Randell, editors. *Software Engineering: Report on a conference sponsored by the NATO SCIENCE COMMITTEE, Garmisch, Germany, 7th to 11th October 1968*, 1969. Scientific Affairs Division, NATO. URL `http://www.cs.ncl.ac.uk/people/brian.randell/home.formal/NATO/`.

G. Schmidt. *Relational Mathematics*. Number 132 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, November 2010. ISBN 9780521762687.

A. Sernadas, J. Ramos, and P. Mateus. Linear algebra techniques for deciding the correctness of probabilistic programs with bounded resources. Technical report, SQIG - IT and IST - TU

Lisbon, 1049-001 Lisboa, Portugal, 2008. Short paper presented at LPAR 2008, Doha, Qatar. November 22-27.