Calculating risk in functional programming

J.N. Oliveira
(joint work with D. Murta)

IFIP WG2.1 MEETING #70 - ULM, JULY 2013

QAIS Project - Grant PTDC/EIA-CCO/122240/2010

INESC TEC & University of Minho

Motivation

Two-sided motivation:

- **Practical** Software **safety** and **certification** standards concerned with calculating **risk** involved in safety-critical software.
- **Theoretical** Quantitative methods in the algebra of programming (**AoP**) lead to a **LAoP** ("L" for *linear*).

Question:

Can we **transform** *(functional) programs so as to mitigate unexpected faults better than the original ones?*

Safety and certification

Formal method bias:

Interested in the opportunities open for Formal Methods by RTCA DO 178C for certifying airborne software.

Challenged by

> *the use of formal methods to be "at least as good as" a conventional approach that does not use formal methods.* *(Joyce, 2011)*

[... "at least as good" ? ...]

Qualitative vs quantitative

Quoting Jackson (2009):

> *A* ***dependable system*** *is one (..) in which you can place your reliance or trust. A rational person or organization only does this with* ***evidence*** *that the system's* ***benefits*** *outweigh its* ***risks****.*

In formula

$$\mathit{dependable\ system} = \mathit{benefit} + \mathit{risk}$$

we identify:

- **benefit** = qualitative
- **risk** = quantitative.

Safety cases

MOD Defence Standard 00-56:

> ***9.1*** *The Contractor shall produce a* ***Safety Case*** *for the system [which] shall [provide] a* ***compelling****,* ***comprehensible*** *and* ***valid*** *case that a system is safe for a given application in a given environment.*

DS 00-56 (contd.):

> ***10.5.4*** *All assumptions, data, judgements and calculations underpinning the* ***Risk Estimation*** *shall be recorded in the* ***Safety Case****, such that the risk estimates can be reviewed and reconstructed.*

Risk estimation? Calculations? How, when and where is this performed in a **FM** life-cycle?

P(robabilistic)R(isk)A(nalysis)

NASA/SP-2011-3421 (Stamatelatos and Dezfuli, 2011):

> *1.2.2 A PRA characterizes risk in terms of three basic questions: (1) What can* ***go wrong****? (2) How* ***likely*** *is it? and (3) What are the* ***consequences****?*

The PRA process

> *answers these questions by systematically (...) identifying, modeling, and* ***quantifying*** *scenarios that can lead to undesired consequences*

Moreover,

> *1.2.3 (...) The* ***total probability*** *from the set of scenarios modeled may also be non-negligible even though the probability of each scenario is small.*

Doesn't work in FMs — why?

Program semantics are usually **qualitative** — how does one **quantify** risk in standard denotational semantics?

PRA performed **a posteriori** — we've seen this before, eg. in *'a posteriori'* program correctness.

Need for a change:

> *Programming should incorporate risk as the rule rather than the exception (absence of risk =* **ideal** *case).*

Need for **combinators** expressing risk of failure, eg. **probabilistic choice** (McIver and Morgan, 2005)

> $bad\ {}_p\!\diamond\ good$

between **expected behaviour** and **misbehaviour**.

Quantitative semantics

Program semantics denoted by (typed) stochastic **matrices**.

Semantics of language constructs modelled by **linear algebra** operators — for instance,

$$[\![P_1; P_2]\!] = [\![P_2]\!] \cdot [\![P_1]\!]$$

where the dot means matrix **multiplication** — including **recursion**.

Laws of the **LAoP** enable the **calculation of risk** (eg. fault propagation).

Simulation easy to perform in a monadic language such as Haskell (distribution monad).

Quantitative functional programming

Monadic code is in general ready to accommodate **PRA** simulation in functional programming. Example: a lossy channel

$$fcat\ p = \mathbf{fold}\ (lose\ {}_p\!\diamond\ send)\ nil$$

(for $send = cons$ and $lose = snd$) in which we express the fact that, with probability p, $fcat$ fails to pass data from input to output.

For $p = 0.1$, for instance, distribution $fcat\ p$ `"abc"` will range from **perfect copy** (72.9%) to **complete loss** (0.1%) — cf. "quantified suffix":

"abc"	72.9%
"ab"	8.1%
"ac"	8.1%
"bc"	8.1%
"a"	0.9%
"b"	0.9%
"c"	0.9%
""	0.1%

Details

Nothing special, just a monadic variant of *foldr*

$$\begin{array}{l}\textbf{fold} :: \mathit{Monad}\ m \Rightarrow ((a,b) \rightarrow m\ b) \rightarrow m\ b \rightarrow [a] \rightarrow m\ b \\ \textbf{fold}\ f\ d\ [\,] = d \\ \textbf{fold}\ f\ d\ (h : t) = \textbf{do}\ \{x \leftarrow \textbf{fold}\ f\ d\ t; f\ (h,x)\}\end{array}$$

which switches to distributions or lists (cf. the suffix view above) as you wish.

Later we will need **for**-loops, so we anticipate this combinator:

$$\begin{array}{l}\textbf{for} :: (\mathit{Monad}\ m, \mathit{Integral}\ t) \Rightarrow (b \rightarrow m\ b) \rightarrow b \rightarrow t \rightarrow m\ b \\ \textbf{for}\ b\ i\ 0 = \mathit{return}\ i \\ \textbf{for}\ b\ i\ (n+1) = \textbf{do}\ \{x \leftarrow \textbf{for}\ b\ i\ n; b\ x\}\end{array}$$

Quantitative functional programming

Another example:

$$fcount\ q = \mathbf{fold}\ ((id\ {}_q\!\diamond\ succ) \cdot snd)\ \underline{0}$$

is a risky *length* function: with probability q, it doesn't count. For instance, for $q = 0.15$, distribution $fcount\ q$ `"abc"` will be:

```
3  61.4%
2  32.5%
1   5.7%
0   0.3%
```

However, we are **simulating** — not predicting!

Question: what can we predict about $(fcount\ q) \cdot (fcat\ p)$? Can we fuse this?

Quantitative functional programming

Another example:

$$fcount\ q = \mathbf{fold}\ ((id\ {}_q\!\diamond\ succ) \cdot snd)\ \underline{0}$$

is a risky *length* function: with probability q, it doesn't count. For instance, for $q = 0.15$, distribution $fcount\ q$ `"abc"` will be:

```
3   61.4%
2   32.5%
1    5.7%
0    0.3%
```

However, we are **simulating** — not predicting!

Question: what can we predict about $(fcount\ q) \cdot (fcat\ p)$? Can we fuse this?

Fault fusion

Fold-fusion law in the **LAoP**

$$k \cdot (\textbf{fold}\ g\ e) = \textbf{fold}\ f\ d \quad \Leftarrow \quad k \cdot [e|g] = [d|f] \cdot (F\ k) \qquad (1)$$

holds in the probabilistic setting, but now

- regard function variables (eg. k, g, e etc) as (column) stochastic **typed** matrices;
- such matrices represent the **Kleisli** category of the distribution monad;
- $F\ k = id \oplus (id \otimes k)$ is the base functor, where $\cdot \otimes \cdot$ denotes **Kronecker** product and $\cdot \oplus \cdot$ denotes **direct sum** of matrices;
- $[f|g]$ glues f and g horizontally (**coproduct** combinator).

Fault fusion

We want to solve equation $(fcount\ q) \cdot (fcat\ p) = fold\ x\ y$ for unknowns x and y:

$$
\begin{array}{ll}
 & (fcount\ q) \cdot (fcat\ p) = fold\ x\ y \\
\Leftarrow & \quad \{\ fold \text{ fusion (1) ; definition of } fcat\ \} \\
 & (fcount\ q) \cdot [nil|(lose\ {}_p\diamond\ send)] = [x|y] \cdot (F\ (fcount\ q)) \\
\Leftrightarrow & \quad \{\ \text{coproduct fusion (2) ; definition of } F\text{; (3) ; (4)}\ \} \\
 & \left\{ \begin{array}{l} (fcount\ q) \cdot nil = x \\ (fcount\ q) \cdot (lose\ {}_p\diamond\ send) = y \cdot (id \otimes (fcount\ q)) \end{array} \right.
\end{array}
$$

where (LAoP):

$$
\begin{aligned}
P \cdot [M|N] &= [P \cdot M|P \cdot N] && (2)\\
[M|N] \cdot (P \oplus Q) &= [M \cdot P|N \cdot Q] && (3)\\
[M|N] = [P|Q] &\Leftrightarrow M = P \wedge N = Q && (4)
\end{aligned}
$$

Fault fusion (cntd.)

From $(fcount\ q) \cdot nil = \underline{0}$ we obtain $x = \underline{0}$.

We are left with the second equality, which we solve for y knowing that **choice-fusion** laws

$$h \cdot (f \ {}_p\!\diamond\ g) = (h \cdot f) \ {}_p\!\diamond\ (h \cdot f) \tag{5}$$

$$(f \ {}_p\!\diamond\ g) \cdot h = (f \cdot h) \ {}_p\!\diamond\ (g \cdot h) \tag{6}$$

hold:

$$
\begin{array}{ll}
& ((fcount\ q) \cdot (snd \ {}_p\!\diamond\ cons) = y \cdot (id \otimes (fcount\ q)) \\
\Leftrightarrow & \qquad \{\ \text{choice fusion (5)}\ \} \\
& ((fcount\ q) \cdot snd) \ {}_p\!\diamond\ ((fcount\ q) \cdot cons) = y \cdot (id \otimes (fcount\ q)) \\
\Leftrightarrow & \qquad \{\ \text{unfolding } (fcount\ q) \cdot cons\ \} \\
& ((fcount\ q) \cdot snd) \ {}_p\!\diamond\ ((id \ {}_q\!\diamond\ succ) \cdot snd \cdot (id \otimes (fcount\ q))) \\
& = y \cdot (id \otimes (fcount\ q))
\end{array}
$$

Fault fusion (cntd.)

The free theorem of *snd*

$$snd \cdot (f \otimes g) = g \cdot snd \tag{7}$$

helps in the next step:

$$
\begin{array}{ll}
 & ((fcount\ q) \cdot snd)\ {}_{p}\!\diamond\ ((id\ {}_{q}\!\diamond\ succ) \cdot (fcount\ q) \cdot snd) \\
 & = y \cdot (id \otimes (fcount\ q)) \\
\Leftrightarrow & \qquad \{\ \text{choice fusion (6)}\ \} \\
 & (id\ {}_{p}\!\diamond\ (id\ {}_{q}\!\diamond\ succ)) \cdot (fcount\ q) \cdot snd = y \cdot (id \otimes (fcount\ q)) \\
\Leftrightarrow & \qquad \{\ \text{free theorem (7) again}\ \} \\
 & (id\ {}_{p}\!\diamond\ (id\ {}_{q}\!\diamond\ succ)) \cdot snd \cdot (id \otimes (fcount\ q)) = y \cdot (id \otimes (fcount\ q)) \\
\Leftarrow & \qquad \{\ \text{Leibniz — cancel } id \otimes (fcount\ q) \text{ from both sides}\ \} \\
 & y = (id\ {}_{p}\!\diamond\ (id\ {}_{q}\!\diamond\ succ)) \cdot snd
\end{array}
$$

Fault fusion (conc.)

Putting everything together, we have **consolidated** the risk of pipeline $(\mathit{fcount}\ q) \cdot (\mathit{fcat}\ p)$ into

$$\begin{array}{l} \mathit{fold}\ y\ \underline{0}\ \textbf{where} \\ \quad y = ((p+q-pq)\ \mathit{id} + (1-p)\ (1-q)\ \mathit{succ}) \cdot \mathit{snd} \end{array}$$

using definition

$$f\ {}_{p}\!\diamond g \quad = \quad p \otimes f + (1-p) \otimes g \tag{8}$$

Higher p, q reduce the probability of succ taking place.

FAULT FUSION: the risk of the **whole** expressed in terms of the risk of the **parts**.

Fault fusion (conc.)

From the calculation we can infer eg.

$$(fcount\ 0) \cdot (fcat\ p) \quad = \quad (fcount\ p) \cdot (fcat\ 0)$$

since terms

$$(0 + p - 0\ p)\ id + (1 - 0)\ (1 - p)\ succ)$$
$$(p + 0 - p0)\ id + (1 - p)\ (1 - 0)\ succ$$

are the same. In words:

> *(for the same probabilities), a* **perfect** *counter reading from a* **faulty** *channel is indistinguishable from a* **faulty** *counter reading from a* **perfect** *channel.*

Clearly, black-box **testing** and **simulation** wouldn't be able to spot where the fault is.

LAoP vs AoP

Summing up,

- in the same way **relations** are needed in standard AoP for calculating **functions**,
- so are (typed) **matrices** in the LAoP for calculating **probabilistic functions**.
- AoP extends smoothly to the LAoP, but not the whole of it.
- A significant difference can be found in **pairing** (**tupling** in general) and **mutual recursion**.

Thus we focus on **probabilistic pairing** in the sequel.

Running examples

Consider two little programs in C, one which supposedly computes the square of a non-negative integer n,

```
int sq(int n) {
  int s=0; int o=1; int i;
  for (i=1;i<n+1;i++) {s+=o; o+=2;}
  return s;
};
```

and the other

```
int fib(int n) {
  int x=0; int y=1; int i;
  for (i=1;i<=n;i++) {int a=y; y=y+x; x=a;}
  return x;
};
```

which supposedly computes the n-th entry in the Fibonacci series, for n positive.

Running examples

Both programs are **for**-loops whose bodies rely on the same operation: **addition** of natural numbers.

Suppose one knows that, in the machine where such programs will run, there is the risk of addition misbehaving in some known way: with probability p, $x + y$ may evaluate to y, in which case $(x+) = id$.

Or one might know that, in some unfriendly environment, the processor's ALU may reset addition output to 0, with probability q.

Question: how do the above programs "react" to such faults?

Simulation

As before, we may encode the two programs in Haskell using the **for** combinator,

$$
\begin{array}{l}
\mathit{sq}\ n = \\
\quad \mathbf{do}\ \{(s, o) \leftarrow \mathbf{for}\ \mathit{loop}\ (0, 1)\ n; \mathit{return}\ s\}\ \mathbf{where} \\
\qquad \mathit{loop}\ (s, o) = \mathbf{do}\ \{z \leftarrow \mathit{fadd}\ 0.1\ s\ o; \mathit{return}\ (z, o + 2)\} \\
\mathit{fib}\ n = \\
\quad \mathbf{do}\ \{(x, y) \leftarrow \mathbf{for}\ \mathit{loop}\ (0, 1)\ n; \mathit{return}\ x\}\ \mathbf{where} \\
\qquad \mathit{loop}\ (x, y) = \mathbf{do}\ \{z \leftarrow \mathit{fadd}\ 0.1\ x\ y; \mathit{return}\ (y, z)\}
\end{array}
$$

both calling

$$\mathit{fadd}\ p\ a = \underline{0}\ {}_{p}\!\diamond (a+)$$

— a risky addition which resets with probability p.

Simulation

Then we may simulate, for instance ($p = 0.1$)

$sq\ 6 =$

11	10%
20	9%
27	8%
32	7%
35	7%
36	59%

and, for instance:

$fib\ 6 =$

1	0%
2	0%
3	2%
4	2%
5	15%
6	15%
8	66%

Simulation

However, this does not tell anything special about what's happening.

We know that both programs can be derived from their specification (resp. $sq\ n = n^2$ and the *binary recursive* definition of Fibonacci) using the **mutual-recursion** law, vulg. **tupling** (Hu et al., 1997).

One way to compare the two implementations would be to check how far they are from their specifications (under the same faults).

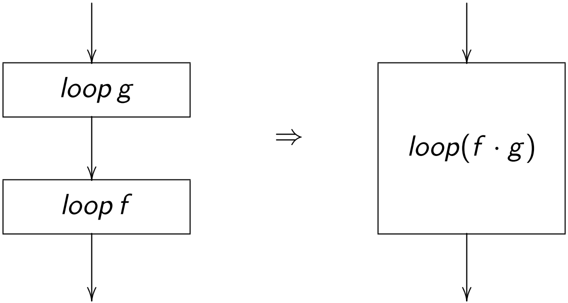
By experimentation, we observed that spec + imp of sq seem probabilistically indistinguishable, while **Fibonacci** does not: the **linear** version is (as much as we could test) **less** risky — next slide:

Simulation (faulty Fibonacci)

n	*recursive spec*		**for** *loop implementation*	
$\leqslant 4$	*the same*			
5	5	65.6%	5	72.9%
	4	21.9%	3	16.2%
	3	10.5%	4	8.1%
	2	1.9%	2	2.7%
	1	0.1%	1	0.1%
6	8	47.8%	8	65.6%
	7	26.6%	6	14.6%
	6	11.8%	5	14.6%
	5	9.8%	3	2.4%
	4	2.7%	4	2.4%
	3	1.1%	2	0.4%
	2	0.2%	1	0.0%
	1	0.0%		

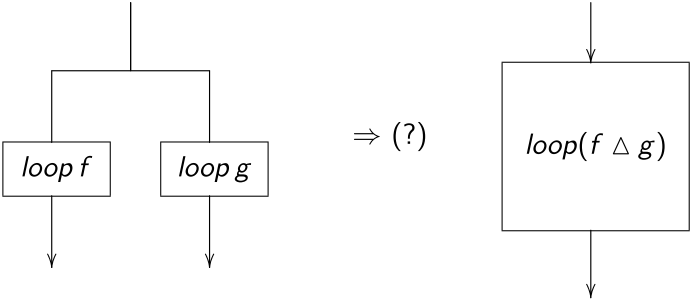
Mutual recursion?

These experiments pointed towards checking the validity of **tupling** in the LAoP: while "vertical" (sequential) loop-fusion laws hold,



Mutual recursion?

... "horizontal loop-fusion"



does not seem to hold in general. Why?

Probabilistic pairing

Pairing the outputs of two probabilistic functions f and g is captured by their **Khatri-Rao** matrix product (keep thinking in terms of matrices)

$$
\begin{tikzcd}
A \times B \\
C \arrow[u, "f \triangle g"']
\end{tikzcd}
\qquad
\begin{tikzcd}
A & A \times B \arrow[l, "fst"'] \arrow[r, "snd"] & B \\
& C \arrow[ul, "f"] \arrow[u, "f \triangle g"'] \arrow[ur, "g"'] &
\end{tikzcd}
$$

but (important!) this is a **weak** categorial product:

$$
k = f \triangle g \quad \Rightarrow \quad \left\{ \begin{array}{r} fst \cdot k = f \\ snd \cdot k = g \end{array} \right. \tag{9}
$$

cf. the $\Rightarrow$ in (9) — Khatri-Rao is injective but not surjective (unlike pairing in Sets).

Probabilistic pairing

Weak product (9) still grants the **cancellation** rule,

$$fst \cdot (f \triangle g) = f \ \wedge \ snd \cdot (f \triangle g) = g \tag{10}$$

$$fst = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix} \qquad snd = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{tikzcd}
2 & 2 \times 3 \arrow[l, "fst"'] \arrow[r, "snd"] & 3 \\
& 4 \arrow[ul, "f"] \arrow[u, "f \triangle g"] \arrow[ur, "g"'] &
\end{tikzcd}$$

$$f = \begin{bmatrix} 0.5 & 0.3 & 0 & 0.75 \\ 0.5 & 0.7 & 1 & 0.25 \end{bmatrix} \qquad f \triangle g = \begin{bmatrix} 0.15 & 0.12 & 0 & 0 \\ 0.35 & 0.06 & 0 & 0.75 \\ 0 & 0.12 & 0 & 0 \\ 0.15 & 0.28 & 0.1 & 0 \\ 0.35 & 0.14 & 0.2 & 0.25 \\ 0 & 0.28 & 0.7 & 0 \end{bmatrix} \qquad g = \begin{bmatrix} 0.3 & 0.4 & 0.1 & 0 \\ 0.7 & 0.2 & 0.2 & 1 \\ 0 & 0.4 & 0.7 & 0 \end{bmatrix}$$

Probabilistic pairing

... but **fusion** becomes side-conditioned

$$(f \vartriangle g) \cdot h = (f \cdot h) \vartriangle (g \cdot h) \quad \Leftarrow \quad h \text{ is "sharp" } (100\%) \tag{11}$$

and **reconstruction** doesn't hold in general

$$k = (fst \cdot k) \vartriangle (snd \cdot k)$$

cf. eg.

$$k : 2 \to 2 \times 3$$

$$k = \begin{bmatrix} 0 & 0.4 \\ 0.2 & 0 \\ 0.2 & 0.1 \\ 0.6 & 0.4 \\ 0 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad (fst \cdot k) \vartriangle (snd \cdot k) = \begin{bmatrix} 0.24 & 0.4 \\ 0.08 & 0 \\ 0.08 & 0.1 \\ 0.36 & 0.4 \\ 0.12 & 0 \\ 0.12 & 0.1 \end{bmatrix}$$

(k is not recoverable from its projections — Khatri-Rao not surjective).

Asymmetric Khatri-Rao fusion

Another side condition granting fusion is

$$(f \vartriangle g) \cdot h = (f \cdot h) \vartriangle (g \cdot h) \quad \Leftarrow \quad f \cdot h \text{ or } g \cdot h \text{ is } 100\% \tag{12}$$

which enables the following **probabilistic** mutual recursion law (**tupling**):

$$\left\{ \begin{array}{l} f \cdot \mathbf{in} = h \cdot F\,(f \vartriangle g) \\ g \cdot \mathbf{in} = k \cdot F\,(f \vartriangle g) \end{array} \right. \quad \Leftrightarrow \quad f \vartriangle g = (\!| h \vartriangle k |\!) \tag{13}$$

provided one of

$$h \cdot F\,(f \vartriangle g) \text{ or } k \cdot F\,(f \vartriangle g)$$

is 100% — generic statement for polynomial F.

Asymmetric tupling

The calculation of (13) uses the two conditioned pairing fusion laws in different places:

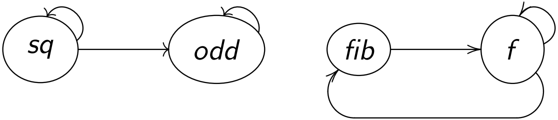
$$
\begin{array}{ll}
& f \vartriangle g = (\!| h \vartriangle k |\!) \\
\Leftrightarrow & \qquad \{ \text{ cata (fold) universal property } \} \\
& (f \vartriangle g) \cdot \mathbf{in} = (h \vartriangle k) \cdot F\ (f \vartriangle g) \\
\Leftrightarrow & \qquad \{ \text{ “sharp” fusion (11) ; asymmetric fusion (12) } \} \\
& (f \cdot \mathbf{in}\ ) \vartriangle (g \cdot \mathbf{in}\ ) = (h \cdot F\ (f \vartriangle g)) \vartriangle (k \cdot F\ (f \vartriangle g)) \\
\Leftrightarrow & \qquad \{ \text{ Khatri-Rao equality } \} \\
& \left\{ \begin{array}{l} f \cdot \mathbf{in} = h \cdot F\ (f \vartriangle g) \\ g \cdot \mathbf{in} = k \cdot F\ (f \vartriangle g) \end{array} \right.
\end{array}
$$

Back to square and Fibonacci

Standard tupling derivations,

$$\begin{array}{ll}
sq\ 0 = 0 & fib\ 0 = 0 \\
sq\ (n+1) = sq\ n + odd\ n & fib\ (n+1) = f\ n \\
odd\ 0 = 1 & f\ 0 = 1 \\
odd\ (n+1) = 2 + odd\ n & f\ (n+1) = fib\ n + f\ n
\end{array}$$

show why $sq \vartriangle odd$ and $fib \vartriangle f$ react differently to faulty addition, cf.



— odd does not depend on sq and therefore remains 100% — as opposed to fib and f, which contaminate each other.

Probabilistic banana-split

This also helps to see why **banana-split** still holds for f and g probabilistic:

$$(\!| f |\!) \vartriangle (\!| g |\!) \quad = \quad (\!| (f \otimes g) \cdot (\underbrace{F\ fst \vartriangle F\ snd}_{unzip_{\mathsf{F}}}) |\!) \tag{14}$$

— the two computations go side-by-side and don't interfere with each other.

This time the proof relies in something I've been using only recently: **free theorems** in linear algebra, in this case

$$(F\ f \otimes F\ g) \cdot unzip_{\mathsf{F}} \quad = \quad unzip_{\mathsf{F}} \cdot F\ (f \otimes g)$$

derived using hom-functors in matrix categories — inspired by Hinze (2012).

Wrapping up

First round of AoP extension towards **LAoP** (folds)

Probabilistic **unfolds** require sub-distributions while computing fixpoints (current work)

Currently using them in checking fault propagation in Barbosa (2001)'s **components as coalgebras** (probabilistic automata networks)

Probabilistic **hylomorphisms** are next.

Wrapping up

Weak tupling has opened new perspectives, namely in relation to **Rel** and to categorial quantum physics, under the umbrella of **monoidal** categories.

In fact, these also include *FdHilb*, the category of finite dimensional Hilbert spaces. — thus the remarks by Coecke and Paquette, in their *Categories for the Practising Physicist* (Coecke, 2011):

> *Rel [the category of relations] possesses more 'quantum features' than the category Set of sets and functions [...] The categories FdHilb and Rel moreover admit a categorical matrix calculus.*

I agree: *Set* is too perfect to "belong to reality"...

L.S. Barbosa. *Components as Coalgebras*. University of Minho, December 2001. Ph. D. thesis.

B. Coecke, editor. *New Structures for Physics*. Number 831 in Lecture Notes in Physics. Springer, 2011. doi: 10.1007/978-3-642-12821-9.

Ralf Hinze. Adjoint folds and unfolds—an extended study. *Science of Computer Programming*, (0):–, 2012. ISSN 0167-6423. doi: http://dx.doi.org/10.1016/j.scico.2012.07.011. URL http://www.sciencedirect.com/science/article/pii/S0167642312001396. DOI: 10.1016/j.scico.2012.07.011. In press.

Z. Hu, H. Iwasaki, M. Takeichi, and A. Takano. Tupling calculation eliminates multiple data traversals. In *In ACM SIGPLAN International Conference on Functional Programming*, pages 164–175. ACM Press, 1997.

D. Jackson. A direct path to dependable software. *Commun. ACM*, 52(4):78–88, 2009.

J. Joyce. Proposed Formal Methods Supplement for RTCA DO

178C, 2011. High Confidence Software and Systems, 11th Annual Conference, 3-6 May 2011, Annapolis.

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

M. Stamatelatos and H. Dezfuli. Probabilistic Risk Assessment Procedures Guide for NASA Managers and Practitioners, 2011. NASA/SP-2011-3421, 2nd edition, December 2011.