Software components as monadic, weighted Mealy machines in typed linear algebra

J.N. Oliveira
(joint work with L.S. Barbosa and D. Murta)

DCC/FCUP, PORTO, 19TH SEP., 2013

Grant PTDC/EIA-CCO/122240/2010

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

INESC TEC & University of Minho

Motivation

Safety and **certification**

> *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 as" ? ...]

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

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

one finds:

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

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** — Hmmm... we've seen this mistake before, eg. in 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**.

In this talk

Interested in reasoning about the risk of **faults propagating** in **component**-based software (**CBS**) systems.

Traditional CBS **risk analysis** relies on *semantically weak* CBS models — eg. over component call-graphs as in (Cortellessa and Grassi, 2007).

Our starting point is the **coalgebraic** semantics of Barbosa (2001) for (safe) CBS systems, under the lemma:

> *"Components as coalgebras"*

Main ideas

Component = (Monadic) Mealy machine (MMM), that is, an $\mathfrak{F}$ -branching transition structure of type:

$$S \times I \to \mathfrak{F}(S \times O)$$

where $\mathfrak{F}$ is a monad.

Component-oriented design = Algebra of MMM combinators

Semantics = Coalgebraic, calculational

To this framework we want to add calculation of

Risk = Probability of faulty (catastrophic) behaviour

Mealy machines in various guises

$\mathfrak{F}$ -branching transition structure:

$$S \times I \rightarrow \mathfrak{F}(S \times O)$$

Coalgebra:

$$S \rightarrow (\mathfrak{F}(S \times O))^I$$

State-monadic:

$$I \rightarrow (\mathfrak{F}(S \times O))^S$$

All versions useful in component algebra.

Example — stack

A **stack** at functional level

$$
\begin{aligned}
push &= flip\ (:) \\
pop &= tail \\
top &= head \\
empty &= (0{=}) \cdot length
\end{aligned}
$$

is a collection of **partial** functions on the free monoid of finite sequences.

Each such partial function gives rise to a **method**, ie. an (elementary) Mealy machine.

The stack component will arise as the **sum** of its methods.

Individual Mealy (Maybe) machines

Example of a method

$$push' :: ([b], b) \rightarrow ([b], 1)$$
$$push' = \widehat{push} \vartriangle !$$

which resorts

(a) to the **uncurry** operator,

$$\widehat{f}\ (a, b) = f\ a\ b$$

(b) to the **pairing** operator,

$$(f \vartriangle g)\ x = (f\ x, g\ x)$$

(c) and to uniquely defined (total) function $! :: b \rightarrow 1$ ('bang').

Partiality — rule rather than exception

Partiality, however, requires 'Maybe' ($\mathfrak{M}$) Mealy machines, one per totalized (partial) function, eg.:

$$
\begin{aligned}
&pop' :: ([a], 1) \rightarrow \mathfrak{M}\ ([a], a)\\
&pop' = (pop \vartriangle top) \Leftarrow (\neg \cdot empty) \cdot \pi_1
\end{aligned}
$$

where $\cdot \Leftarrow \cdot$ totalizes a partial function by fusion with a **precondition**,

$$
\begin{aligned}
&\cdot \Leftarrow \cdot :: (a \rightarrow b) \rightarrow (a \rightarrow \mathbb{B}) \rightarrow a \rightarrow \mathfrak{M}\ b\\
&f \Leftarrow p = p \rightarrow (\eta \cdot f)\ ,\ \bot
\end{aligned}
$$

where unit η (of $\mathfrak{M}$) means **success** and 'zero' element $\bot$ means **failure**.

Standard stack methods

$$empty' :: ([a], 1) \rightarrow \mathfrak{M}\ ([a], \mathbb{B})$$
$$empty' = \eta \cdot (id \vartriangle empty) \cdot \pi_1$$

$$top' :: ([a], 1) \rightarrow \mathfrak{M}\ ([a], a)$$
$$top' = (id \vartriangle top \ \Leftarrow \ (\neg \cdot empty)\ ) \cdot \pi_1$$

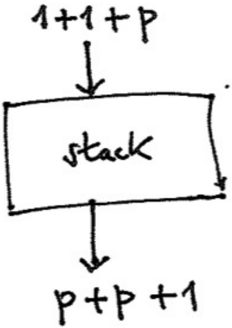
$$push' :: ([b], b) \rightarrow \mathfrak{M}\ ([b], 1)$$
$$push' = \eta \cdot (\widehat{push} \vartriangle !)$$

$$pop' :: ([a], 1) \rightarrow \mathfrak{M}\ ([a], a)$$
$$pop' = (pop \vartriangle top \ \Leftarrow \ (\neg \cdot empty)\ ) \cdot \pi_1$$

Component $= \sum$ methods

The stack **component**

$$\begin{array}{l} stack :: ([p], (1+1)+p) \rightarrow \mathfrak{M}\ ([p], (p+p)+1) \\ stack = pop' \oplus top' \oplus push' \end{array}$$



is built thanks to the MMM **sum** combinator

$$\begin{array}{l} \cdot \oplus \cdot :: (Functor\ \mathfrak{F}) \Rightarrow \\ \quad \text{-- input machines} \\ \quad ((s,i) \rightarrow \mathfrak{F}\ (s,o)) \rightarrow \\ \quad ((s,j) \rightarrow \mathfrak{F}\ (s,p)) \rightarrow \\ \quad \text{-- output machine} \\ \quad (s, i+j) \rightarrow \mathfrak{F}\ (s, o+p) \\ \quad \text{-- definition} \\ m_1 \oplus m_2 = (\mathfrak{F}\ dr^\circ) \cdot \Delta \cdot (m_1 + m_2) \cdot \mathrm{dr} \end{array}$$

where (next slide)

Component $= \sum$ methods

- $m_1 + m_2$ is categorial sum (coproduct);
- isomorphism

$$
\begin{array}{l}
\mathrm{dr} :: (a, c + b) \rightarrow (a, c) + (a, b) \\
\mathrm{dr}\ (a, i_1\ b) = i_1\ (a, b) \\
\mathrm{dr}\ (a, i_2\ c) = i_2\ (a, c)
\end{array}
$$

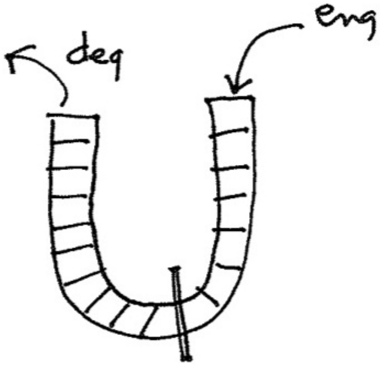
  (resp. dr°) distributes (resp. factorizes) the shared state across the sum of inputs (resp. outputs)
- "Cozip" operator

$$
\begin{array}{l}
\Delta :: (Functor\ \mathfrak{F}) \Rightarrow (\mathfrak{F}\ a) + (\mathfrak{F}\ b) \rightarrow \mathfrak{F}\ (a + b) \\
\Delta = [(\mathfrak{F}\ i_1), (\mathfrak{F}\ i_2)]
\end{array}
$$

  promotes coproducts through $\mathfrak{F}$.

Combining components

Queue = two stacks:



out stack (left) interacting with **in** stack (right)

More MMM combinators

For the two stacks to interact we need to be able to **compose** two MMM,

$$
\begin{array}{l}
\cdot \,;\, \cdot :: (\mathit{Strong}\ \mathfrak{F}, \mathit{Monad}\ \mathfrak{F}) \Rightarrow \\
\quad \text{-- input machines} \\
\quad ((s,i) \to \mathfrak{F}\ (s,j)) \to \\
\quad ((r,j) \to \mathfrak{F}\ (r,k)) \to \\
\quad \text{-- output machine} \\
\quad ((s,r),i) \to \mathfrak{F}\ ((s,r),k) \\
\quad \text{-- definition} \\
m_1 \,;\, m_2 = ((\mathfrak{F}\ \mathrm{a}^\circ) \cdot \tau_l \cdot (id \times m_2) \cdot \mathrm{xl}) \bullet \tau_r \cdot (m_1 \times id) \cdot \mathrm{xr}
\end{array}
$$

which is the $\mathfrak{F}$-Kleisli composition $(\bullet)$ of suitably wrapped, "paired" m_1 and m_2 .

(Explanatory diagrams in the following slide.)

More MMM combinators

Kleisli composition for monad $\mathfrak{F}$

$$
\begin{tikzcd}
\mathfrak{F}\,(\mathfrak{F}\;C) \arrow[d, "\mu"'] & \mathfrak{F}\;B \arrow[l, "\mathfrak{F}\;f"'] \arrow[d, dash, dotted] & A \arrow[l, "g"'] \arrow[dll, bend left, "f \bullet g"] \\
\mathfrak{F}\;C & B \arrow[l, "f"'] &
\end{tikzcd}
$$

which is the composition in the corresponding **Kleisli category**, with η as identity:

$$f \bullet (g \bullet h) = (f \bullet g) \bullet h$$
$$f \bullet \eta = f = \eta \bullet f$$

Conceptually, it is as if one (typewise) drops the $\mathfrak{F}$'s from f and g in the diagram above.

More MMM combinators

Wrapping of m_1 :

$$\mathfrak{F}\,((s,o),r) \xleftarrow{\tau_r} (\mathfrak{F}\,(s,o),r) \xleftarrow{m_1 \times id} ((s,i),r) \xleftarrow{\mathsf{xr}} ((s,r),i)$$

Wrapping of m_2 :

$$\begin{tikzcd}
 & (s,\mathfrak{F}\,(r,k)) \arrow[d, "\tau_l"'] & (s,(r,o)) \arrow[l, "id \times m_2"'] & ((s,o),r) \arrow[l, "\mathsf{xl}"'] \\
\mathfrak{F}\,((s,r),k) & \mathfrak{F}\,(s,(r,k)) \arrow[l, "\mathfrak{F}\,\mathsf{a}^\circ"'] & &
\end{tikzcd}$$

Mind the **strength** operators:

$$\tau_r :: (\mathfrak{F}\,a,b) \rightarrow \mathfrak{F}\,(a,b)$$
$$\tau_l :: (b,\mathfrak{F}\,a) \rightarrow \mathfrak{F}\,(b,a)$$

More MMM combinators

We will also need **interface**-wrapping

$$
\begin{array}{l}
\cdot_{\{\cdot \to \cdot\}} :: (\mathit{Functor}\ \mathfrak{F}) \Rightarrow \\
\quad \text{-- input machine} \\
((a, e) \to \mathfrak{F}\ (a, c)) \to \\
\quad \text{-- input wrapper} \\
(i \to e) \to \\
\quad \text{-- output wrapper} \\
(c \to d) \to \\
\quad \text{-- output machine} \\
(a, i) \to \mathfrak{F}\ (a, d) \\
\quad \text{-- definition} \\
m_{\{f \to g\}} = \mathfrak{F}\ (id \times g) \cdot m \cdot (id \times f)
\end{array}
$$

for $I\ /\ O$ "*wiring*" (analogy with hardware).

Queue component

We build a **queue** out of two **stacks**

$$queue :: (([q],[q]), q+1) \rightarrow \mathfrak{M}\ (([q],[q]), 1+q)$$
$$queue = enq \oplus deq$$

by providing two methods: **enqueueing**

$$enq = nop \,;\, stack_{\{pushIn \rightarrow pushOut\}}$$

which does nothing (nop) on the output stack and pushes onto the input stack (wrapping selects the $push$ method), and **dequeueing**,

$$deq = (stack_{\{popIn \rightarrow popOut\}} \,;\, nop) \bullet check$$

which pops from the output stack, preceded by flushing the input stack should the former be empty:

$$check = (empty' \,;\, nop) \rightarrow (write \overleftarrow{;} flush')\ ,\ nop$$

where $m \overleftarrow{;} n = rev\ (n \,;\, m)$.

Wiring

Helper functions

$$\begin{aligned} popIn &= first \\ topIn &= second \\ pushIn &= third \end{aligned}$$

are convenient renamings of generic *wiring* functions

$$\begin{aligned} first &= i_1 \cdot i_1 \\ second &= i_1 \cdot i_2 \\ third &= i_2 \end{aligned}$$

which route data into place through the parameter sums.

Enriched stacks

NB: our stack component had meanwhile to be enriched with two more methods,

$$
\begin{aligned}
&\mathit{stackpp} :: ([p], (((1+1)+p)+1)+1) \\
&\qquad \rightarrow \mathfrak{M}\ ([p], (((p+p)+1)+\mathbb{B})+[p]) \\
&\mathit{stackpp} = \mathit{stack} \oplus \mathit{empty}' \oplus \mathit{flush}'
\end{aligned}
$$

where empty' checks for stack emptiness and

$$
\begin{aligned}
&\mathit{flush}' :: \mathit{Monad}\ \mathfrak{F} \Rightarrow ([b], 1) \rightarrow \mathfrak{F}\ ([b], [b]) \\
&\mathit{flush}' = \mathit{mkMM}\ \underline{\mathit{nil} \vartriangle \mathit{reverse}}
\end{aligned}
$$

flushes a stack contents to its output (reversed), where

$$
\begin{aligned}
&\mathit{mkMM} :: \mathit{Monad}\ \mathfrak{F} \Rightarrow (a \rightarrow c \rightarrow b) \rightarrow (c, a) \rightarrow \mathfrak{F}\ b \\
&\mathit{mkMM}\ f = \eta \cdot \widehat{f} \cdot \mathit{swap}
\end{aligned}
$$

Conditionals

Also mind the need for a MMM-level McCarthy-styled **conditional** combinator,

$$
\begin{array}{l}
\cdot \rightarrow \cdot \ , \ \cdot :: (\mathit{Monad}\ \mathfrak{F}, \mathit{Functor}\ \mathfrak{F}) \Rightarrow \\
\quad \text{-- condition} \\
\quad ((a, i) \rightarrow \mathfrak{F}\ (a, \mathbb{B})) \rightarrow \\
\quad \text{-- 'then' branch} \\
\quad ((a, 1) \rightarrow \mathfrak{F}\ (a, o)) \rightarrow \\
\quad \text{-- 'else' branch} \\
\quad ((a, 1) \rightarrow \mathfrak{F}\ (a, o)) \rightarrow \\
\quad \text{-- output} \\
\quad (a, i) \rightarrow \mathfrak{F}\ (a, o) \\
\quad \text{-- definition} \\
p \rightarrow m_1 \ , \ m_2 = [m_1, m_2] \bullet (\mathfrak{F}\ \mathrm{dr} \cdot (p_{\{id \rightarrow outB\}}))
\end{array}
$$

where *outB* witnesses isomorphism $\mathbb{B} \cong 1 + 1$.

Changing effect of composition

Finally, term $rev\ (flush' \mathbin{;} write)$ involves

$$
\begin{array}{l}
rev :: Functor\ \mathfrak{F} \Rightarrow \\
\quad \text{-- original MM} \\
\quad (((b, a), i) \rightarrow \mathfrak{F}\ ((b, a), o)) \rightarrow \\
\quad \text{-- changed MM} \\
\quad ((a, b), i) \rightarrow \mathfrak{F}\ ((a, b), o) \\
rev\ \ m = \mathfrak{F}\ (swap \times id) \cdot m \cdot (swap \times id)
\end{array}
$$

where

$$swap\ (b, a) = (a, b)$$

which changes which component is affected first in a composition.

Simulation (Haskell)

Enqueueing:

```
 > coalg queue ("ab","cd") (i1 'z')
Just (("ab","zcd"),Left ())
```

Dequeueing:

```
 > coalg queue ("ab","cd") (i2 ())
Just (("b","cd"),Right 'a')

 > coalg queue ("","cd") (i2 ())
Just (("c",""),Right 'd')
```

NB: `coalg m` converts MMM `m` into the corresponding coalgebra (currying).

Faulty components

Risk of pop' behaving like top' with **probability** $1 - p$

$$pop'' :: \mathbb{P} \to ([a], 1) \to \mathfrak{D}\ (\mathfrak{M}\ ([a], a))$$
$$pop''\ p = pop'\ {}_{p}\!\diamond\ top'$$

and risk of $push'$ not pushing anything, with probability $1 - q$

$$push'' :: \mathbb{P} \to ([a], a) \to \mathfrak{D}\ (\mathfrak{M}\ ([a], 1))$$
$$push''\ q = push'\ {}_{q}\!\diamond\ skip'$$

where $\mathbb{P} = [0, 1]$, $\mathfrak{D}$ is the (finite) **distribution** monad and

$$\cdot\ .\!\diamond \cdot :: \mathbb{P} \to (t \to a) \to (t \to a) \to t \to \mathfrak{D}\ a$$
$$(f\ {}_{p}\!\diamond\ g)\ x = choose\ p\ (f\ x)\ (g\ x)$$

choses between f and g .

Simulation

Example (no faults) — popping from one stack and pushing onto another,

$$m_1 = pop' \,;\, push'$$

should produce the intended behaviour, eg.

```
 > coalg m1 ([1],[2]) ()
Just (([],[1,2]),())
 > coalg m1 ([],[2]) ()
Nothing
```

Example (faults) — now suppose the stacks are faulty,

$$m_2 = pop''\ 0.95 \,;_D\, push''\ 0.8$$

over the same (global) state `([1],[2])`.

Simulation

Running the same simulation, now for machine m_2,

```
 > coalg m2 ([1],[2]) ()
 Just (([],[1,2]),())  76.0%
   Just (([],[2]),())  19.0%
Just (([1],[1,2]),())   4.0%
  Just (([1],[2]),())   1.0%
```

the risk of faulty behaviour is 24% $(1 - 0.76)$, structured as: (a) 1% — both components misbehave; (b) 19% — left stack misbehaves; (c) 4% — right stack misbehaves.

As expected,

```
 > coalg m2 ([],[2]) ()
Nothing 100.0%
```

is **catastrophic** (popping from an empty stack).

Faulty components

Simulation:

Using the **PFP library** *written by Erwig and Kollmansberger (2006).*

Important:

Our MMMs have become probabilistic, leading to coalgebras of general shape

$$S \to (\mathfrak{D}(\mathfrak{F}(S \times O)))^I$$

Challenge:

Need for probabilistic extension of the MMM combinators of Barbosa (2001), for instance (next slide):

Combining faulty components

Sequencing:

$$
\begin{array}{l}
\cdot \,;_D\, \cdot :: \\
\quad \text{-- input probabilistic MMM} \\
\quad ((u,i) \to \mathfrak{D}\ (\mathfrak{M}\ (u,k))) \to \\
\quad \text{-- input probabilistic MMM} \\
\quad ((v,k) \to \mathfrak{D}\ (\mathfrak{M}\ (v,o))) \to \\
\quad \text{-- output probabilistic MMM} \\
\quad (((u,v),i) \to \mathfrak{D}\ (\mathfrak{M}\ ((u,v),o))) \\
m_1 \,;_D\, m_2 = \\
\quad ((\mathfrak{D}\mathfrak{M}\ \mathrm{a}^\circ) \cdot \tau_l^D \cdot (id \times m_2) \cdot \mathrm{xl}) \bullet_D (\tau_r^D \cdot (m_1 \times id) \cdot \mathrm{xr})
\end{array}
$$

where

$$
\begin{array}{l}
\tau_l^D :: Strong\ \mathfrak{F} \Rightarrow (b, \mathfrak{D}\ (\mathfrak{F}\ a)) \to \mathfrak{D}\ (\mathfrak{F}\ (b,a)) \\
\tau_r^D :: Strong\ \mathfrak{F} \Rightarrow (\mathfrak{D}\ (\mathfrak{F}\ a), b) \to \mathfrak{D}\ (\mathfrak{F}\ (a,b))
\end{array}
$$

are "$\mathfrak{D}$"-extended strengths and... (next slide)

Combining faulty components

Combinator $(\bullet_D)$ implements $\mathfrak{M}$-Kleisli composition "wrapped by" $\mathfrak{D}$(istributions)

$$(\bullet_D) :: (c \rightarrow \mathfrak{D}\ (\mathfrak{M}\ b)) \rightarrow (a \rightarrow \mathfrak{D}\ (\mathfrak{M}\ c)) \rightarrow a \rightarrow \mathfrak{D}\ (\mathfrak{M}\ b)$$
$$g \bullet_D f = (\mathfrak{D}\ \mu) \cdot ((\mathfrak{M}_F\ g) \bullet f)$$

where

$$\mathfrak{M}_F :: (Functor\ \mathfrak{F}, Monad\ \mathfrak{F}) \Rightarrow (b \rightarrow \mathfrak{F}\ a) \rightarrow \mathfrak{M}\ b \rightarrow \mathfrak{F}\ (\mathfrak{M}\ a)$$

runs $\mathfrak{M}\ f$ "inside" $\mathfrak{F}$.

Back to research question

Recall coalgebraic type

$$S \to (\mathfrak{D}(\mathfrak{F}(S \times O)))^I$$

How tractable (mathematically) is this doubly-monadic framework? Can $\mathfrak{F}$ *be any monad?*

Relatives of this have been studied elsewhere, eg.

- **reactive probabilistic automata** — $S \to (\mathfrak{M}\ (\mathfrak{D}\ S))^I$
- **generative prob. automata** — $S \to \mathfrak{M}\ (\mathfrak{D}\ (O \times S))$
- **bundle systems** — $S \to \mathfrak{D}\ (\mathfrak{P}\ (O \times S))$

Cf. (Sokolova, 2011)

Back to research question

Related (and inspirational) work:

Bonchi et al. (2012) study coalgebras of functor

$$S \to \mathbb{K} \times (\mathbb{K}^S_\omega)^I$$

for $\mathbb{K}$ a field in their coalgebraic approach to weighted automata.

These coalgebras rely on the so-called **field valuation** (exponential) functor $\mathbb{K}^-_\omega$ calling for **vector spaces**.

Inspired by this approach, a similar framework was studied directly in suitable **categories of matrices** (Oliveira, 2012).

We will do the same in this talk concerning probabilistic MMMs.

Our strategy

Instead of working in Set,

$$
\begin{tikzcd}
 & \mathfrak{D}\,(\mathfrak{F}\,B) \arrow[d, dash, dotted] & A \arrow[l, "g"'] \\
\mathfrak{D}\,(\mathfrak{F}\,C) & B \arrow[l, "f"'] &
\end{tikzcd}
$$

we seek to work **directly** on the Kleisli category of $\mathfrak{D}$, that is

$$
\begin{tikzcd}
 & \mathfrak{F}\,B \arrow[d, dash, dotted] & A \arrow[l, "g"'] \arrow[dll, bend right, "f \bullet g"'] \\
\mathfrak{F}\,C & B \arrow[l, "f"'] &
\end{tikzcd}
$$

thus "abstracting from" monad $\mathfrak{D}$. But — how tractable is such a category?

Our strategy

It turns out to be the (monoidal) category of left (column) stochastic **matrices**,

$$
\begin{tikzcd}
A \to_{Set} \mathfrak{D} B \arrow[rr, bend left] \arrow[rr, phantom, "\cong"] & & A \to_{CS} B \arrow[ll, bend left]
\end{tikzcd}
$$

cf. the adjunction

$$
\begin{tikzcd}
A \arrow[d, "M"'] & \mathfrak{D}\ A & A \arrow[l, "\eta"'] \arrow[dl, "f = \underbrace{\mathfrak{L}\ M \cdot \eta}_{\lfloor M \rfloor}"] \\
B & \mathfrak{D}\ B \arrow[from=u, "\mathfrak{L}\ M"'] &
\end{tikzcd}
$$

where $\mathfrak{L}\ M$ corresponds to the **linear transformation** captured by matrix M :

$$(\mathfrak{L}\ M)\ v\ b \quad = \quad \langle \sum\ a \in A\ ::\ M\ (b, a) \times (v\ a) \rangle$$

Our strategy

Another way to put it is

$$M = \lceil f \rceil \quad \Leftrightarrow \quad \langle \forall\ b, a\ ::\ M(b,a) = (f\ a)\ b \rangle$$

where $\lceil f \rceil$ is the isomorphism converting probabilistic function f into the corresponding CS-matrix.

For instance, the probabilistic **negation** function

$$f \quad = \quad id\ {}_{0.1}\!\diamond (\neg)$$

corresponds to matrix

$$\lceil f \rceil \quad = \quad \begin{array}{c c} & \begin{array}{cc} \textbf{True} & \textbf{False} \end{array} \\ \begin{array}{c} \textbf{True} \\ \textbf{False} \end{array} & \begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix} \end{array}$$

Our strategy

Probabilistic choice is immediate on the matrix side,

$$\lceil f \ {}_{p}\!\diamond\, g \rceil \quad = \quad p \lceil f \rceil + (1 - p) \lceil g \rceil$$

where $(+)$ denotes addition of matrices of the same **type**.

Typed linear algebra: homset $A \rightarrow_{CS} B$ contains all matrices indexed by input (**column**) type A and indexed by output (**row**) type B, whose cardinality is bound to ensuring that matrix composition (**multiplication**),

$$(M \cdot N)(r, c) \quad = \quad \langle \sum \ x \ :: \ M(r, x) \times N(x, c) \rangle$$

is well-defined.

Matrix transform

As we wanted, $\mathfrak{D}$ -Kleisli composition becomes matrix multiplication,

$$\lceil f \bullet g \rceil \quad = \quad \lceil f \rceil \cdot \lceil g \rceil$$

and the unit of $\mathfrak{D}$ (Dirac function) becomes the identity matrix, $\lceil \eta \rceil = id$.

This is a special case of the embedding:

> *Any normal ("sharp") function f in Set is representable by CS matrix $\lceil \eta \cdot f \rceil$, as expected.*

For instance, the embedding of Set isomorphisms leads to **permutation** matrices, that is, CS matrices with exactly one 1 per column and per row.

Column stochasticity

Any CS (column) vector $A \xleftarrow{v} 1$ represents a **distribution**.

CS (row) vector $1 \xleftarrow{!} A$ — wholly filled with 1s — is known as the "**bang** vector" — a constant function.

Clearly, $!$ and id coincide on (scalar) type $1 \to 1$:

$$1 \xleftarrow{id} 1 \quad = \quad 1 \xleftarrow{!} 1$$

Bang is very useful, cf:

Definition: A matrix M is CS iff

$$! \cdot M \quad = \quad !$$

□

Column stochasticity

Not every matrix combinator preserves column stochasticity; composition (of course) does,

$$
\begin{aligned}
& \; ! \cdot (\lceil f \rceil \cdot \lceil g \rceil) \\
= & \qquad \{ \; ! \cdot \lceil f \rceil = \, ! \; \} \\
& \; ! \cdot \lceil g \rceil \\
= & \qquad \{ \; ! \cdot \lceil g \rceil = \, ! \; \} \\
& \; !
\end{aligned}
$$

and thus CS is a subcategory of $Mat_{\boldsymbol{R}}$, but eg. sum does not — $\lceil f \rceil + \lceil g \rceil > \, !$.

Weighted sum (choice) preserves column stochasticity, $! \cdot (\lceil f \rceil \; {}_p\!\diamond \; \lceil g \rceil) = \, !$ as several other useful combinators do (see below).

Column stochasticity

CS has coproducts,

$$(A+B) \to_{CS} C \quad \cong \quad (A \to_{CS} C) \times (B \to_{CS} C)$$

(where $A+B$ is disjoint union) as does $Mat_{\boldsymbol{R}}$, offering universal property

$$X = [M|N] \quad \Leftrightarrow \quad X.i_1 = M \wedge X.i_2 = N$$

where $[i_1|i_2] = id$.

$[M|N]$ is one of the basic matrix **block** combinators — it puts M and N side by side ("'junc").

Its dual, $X = \left[\frac{P}{Q}\right]$, puts P on top of Q ("'split").

Back to the main problem

Rewritten in CS, composition of probabilistic MMM reduces to the non-probabilistic case,

$$\lceil m_1 ;_D m_2 \rceil \quad = \quad \lceil m_1 \rceil ; \lceil m_2 \rceil$$

provided one unfolds the definition of $\lceil m_1 \rceil ; \lceil m_2 \rceil$ in CS rather than in Set.

Our strategy:

> *Instead of staying in the original category and adapting the definition to the probabilistic case*

we

> *keep the original definition by changing category (CS replaces Set)*

Back to the main problem

The advantage is that all *probabilistic accounting* is smoothly carried out by composition in CS, and we don't need to be concerned with that.

However, there is a price to pay, since the definition of $\lceil m_1 \rceil \,;\, \lceil m_2 \rceil$ is **strongly** monadic:

> *Which* **strong** *monads in Set are still strong monads in CS?*

Recall the types of the two strengths:

$$\tau_l : (B \times \mathfrak{F}\, A) \rightarrow \mathfrak{F}\, (B \times A)$$
$$\tau_r : (\mathfrak{F}\, A \times B) \rightarrow \mathfrak{F}\, (A \times B)$$

What do we know about **products** (pairing) in CS?

Probabilistic pairing

Pairing the outputs of probabilistic f and g is captured by their **Khatri-Rao** matrix product (dropping the $\lceil \cdot \rceil$'s for notation economy):

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

However — this is a **weak** categorial product:

$$
k = f \triangle g \quad \Rightarrow \quad \left\{ \begin{array}{l} fst \cdot k = f \\ snd \cdot k = g \end{array} \right. \tag{1}
$$

cf. the $\Rightarrow$ in (1). Unlike pairing in Set, Khatri-Rao is injective but not surjective.

Probabilistic pairing

Weak product (1) still grants the **cancellation** rule,

$$fst \cdot (f \vartriangle g) = f \;\;\wedge\;\; snd \cdot (f \vartriangle g) = g \tag{2}$$

$$\begin{tikzcd}
2 & & 2 \times 3 \arrow[ll, "fst=\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}"'] \arrow[rr, "snd=\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}"] & & 3 \\
& & 4 \arrow[ull, "f=\begin{bmatrix} 0.5 & 0.3 & 0 & 0.75 \\ 0.5 & 0.7 & 1 & 0.25 \end{bmatrix}"] \arrow[u, "f \vartriangle g=\begin{bmatrix} 0.15 & 0.12 & 0 & 0 \\ 0.35 & 0.06 & 0 & 0.75 \\ 0 & 0.12 & 0 & 0 \\ 0.15 & 0.28 & 0.1 & 0 \\ 0.35 & 0.14 & 0.2 & 0.25 \\ 0 & 0.28 & 0.7 & 0 \end{bmatrix}"'] \arrow[urr, "g=\begin{bmatrix} 0.3 & 0.4 & 0.1 & 0 \\ 0.7 & 0.2 & 0.2 & 1 \\ 0 & 0.4 & 0.7 & 0 \end{bmatrix}"'] & &
\end{tikzcd}$$

Probabilistic pairing

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

$$(f \vartriangle g) \cdot h = (f \cdot h) \vartriangle (g \cdot h) \quad \Leftarrow \quad h \text{ is “sharp” } (100\%) \tag{3}$$

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

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

cf. eg.

$$k : 2 \rightarrow 2 \times 3$$

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

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

Probabilistic pairing

In general, Khatri-Rao gives rise to the well-known **Kronecker** (tensor) product

$$M \otimes N \quad = \quad (M \cdot fst) \vartriangle (N \cdot snd)$$

(which is a **bifunctor**) and **absorption** holds:

$$(M \otimes N) \cdot (P \vartriangle Q) \quad = \quad (M.P) \vartriangle (N.Q)$$

Therefore:

$$M \vartriangle N \quad = \quad (M \otimes N) \cdot \underbrace{(id \vartriangle id)}_{\delta}$$

However, δ is not a "uniform copying operation" in the sense of Coecke (2011), for it lacks **naturality** (next slide).

Probabilistic pairing

For probabilistic f

f	a	b
F	0.3	1
T	0.7	0

evaluate $\delta \cdot f$

f	a	b
F	0.3	1
T	0.7	0

delta	F	T
(F,F)	1	0
(F,T)	0	0
(T,F)	0	0
(T,T)	0	1

delta * f	a	b
(F,F)	0.3	1
(F,T)	0	0
(T,F)	0	0
(T,T)	0.7	0

where $\delta : \mathbb{B} \to \mathbb{B} \times \mathbb{B}$

Then evaluate $(f \otimes f) \cdot \delta$

delta	a	b
(a,a)	1	0
(a,b)	0	0
(b,a)	0	0
(b,b)	0	1

(f x f)	(a,a)	(a,b)	(b,a)	(b,b)
(F,F)	0.09	0.3	0.3	1
(F,T)	0.21	0	0.7	0
(T,F)	0.21	0.7	0	0
(T,T)	0.49	0	0	0

(f x f) * delta	a	b
(F,F)	0.09	1
(F,T)	0.21	0
(T,F)	0.21	0
(T,T)	0.49	0

where $\delta : \{a, b\} \to \{a, b\} \times \{a, b\}$

Back to probabilistic MMMs

To extend the **bicategorial** construction of (Barbosa, 2001) to the **probabilistic** case we need to prove that $h ; k = h \otimes k$ (Kronecker product) is a (sharp) **morphism** between compound Mealy machines $m ; n$ and $m' ; n'$,

$$m' ; n' \xleftarrow{h;k} m ; n \tag{4}$$

that is,

$$\begin{tikzcd}
\mathfrak{F}\left((U \times V) \times O\right) \arrow[d, "\mathfrak{F}\left((h \otimes k) \otimes id\right)"'] & (U \times V) \times I \arrow[l, "m;n"'] \arrow[d, "(h \otimes k) \otimes id"] \\
\mathfrak{F}\left((U' \times V') \times O'\right) & (U' \times V') \times I' \arrow[l, "m';n'"]
\end{tikzcd}$$

wherever $m' \xleftarrow{h} m$ and $n' \xleftarrow{k} n$.

MMM morphisms in CS

The proof of (4) will be (in CS) **exactly the same** as that given by Barbosa (2001) (in Set) for "sharp" Mealy machines.

However, such a proof requires:

- τ_l and τ_r natural
- μ natural

Which $\mathfrak{F}$ ensure these properties? Probably many but not all — studying this at the moment.

Below we check the 'maybe' functor $\mathfrak{M}\ A = 1 + A$ in this respect.

This functor is relevant because it captures the **abrupt termination** catastrophic scenario so important in **PRA**.

To be or not to be (natural)

In general

$$\tau_l : B \times \mathfrak{F}\, A \rightarrow \mathfrak{F}\, (B \times A)$$

Maybe instance:

$$\tau_l : 1 + B \times A \leftarrow B \times (1 + A)$$
$$\tau_l = (! \oplus id) \cdot \mathrm{dr}$$

Naturality of τ_l easy to derive from the naturality of dr and that of $! \oplus id$.

The former is an isomorphism, therefore natural in the whole $Mat_{\boldsymbol{R}}$. Below we check the naturality of $1 + B \xleftarrow{! \oplus id} A + B$:

To be or not to be (natural)

Checking:

$$
\begin{array}{ll}
 & (id \oplus N) \cdot (! \oplus id) = (! \oplus id) \cdot (M \oplus N) \\
\Leftrightarrow & \qquad \{ \text{ bifunctor } \cdot \oplus \cdot \ \} \\
 & ! \oplus N = (! \cdot M) \oplus N \\
\Leftrightarrow & \qquad \{ \text{ assumption: } M \text{ is CS } \} \\
 & ! \oplus N = ! \oplus N
\end{array}
$$

Thus, in $Mat_{\boldsymbol{R}}$ the property is constrained by one matrix being CS (represented by f below):

$$(id \oplus N) \cdot (! \oplus id) = (! \oplus id) \cdot (f \oplus N) \tag{5}$$

In CS, (5) always holds.

To be or not to be (natural)

Impact on the naturality of τ_l (f is CS):

$$
\begin{array}{cl}
 & \tau_l \cdot (f \otimes (id \oplus M)) \\
\Leftrightarrow & \quad \{ \text{ definition of } \tau_l \ \} \\
 & (! \oplus id) \cdot \mathsf{dr} \cdot (f \otimes (id \oplus M)) \\
\Leftrightarrow & \quad \{ \text{ isomorphism dr is natural } \} \\
 & (! \oplus id) \cdot ((f \otimes id) \oplus (f \otimes M)) \cdot \mathsf{dr} \\
\Leftrightarrow & \quad \{ \text{ naturality of } ! \oplus id \text{ (5) } \} \\
 & (id \oplus (f \otimes M)) \cdot (! \oplus id) \cdot \mathsf{dr} \\
\Leftrightarrow & \quad \{ \text{ definition of } \tau_l \ \} \\
 & (id \oplus (f \otimes M)) \cdot \tau_l
\end{array}
$$

To be or not to be (natural)

Naturality of

$$\mu : 1 + (1 + A) \rightarrow 1 + A$$
$$\mu = [i_1|id]$$

is granted since any category of matrices has coproducts, upon which μ is defined. This is a special case of

$$\mu : \mathfrak{F}_{\mathfrak{B}}\ (\mathfrak{F}_{\mathfrak{B}}\ A) \rightarrow \mathfrak{F}_{\mathfrak{B}}\ A$$
$$\mu = (\![id|(\mathbf{in} \cdot i_2)]\!)$$

where $\begin{tikzcd} \mathfrak{F}_{\mathfrak{B}}\ A \arrow[r, bend left] \arrow[r, phantom, "\cong"] & A + \mathfrak{B}\ (\mathfrak{F}_{\mathfrak{B}}\ A) \arrow[l, bend left, "\mathbf{in}"] \end{tikzcd}$ is the free monad on polynomial $\mathfrak{B}$. This works because catamorphisms (vulg. folds) have solutions in CS for such functors.

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"...

Wrapping up

Back to main issue:

> *How does one* **compare** *CBS architectures with respect to fault propagation?*

Relationally, the worst (= most unsafe, **most risky**) system of its type corresponds to the topmost relation ($\top$) — anything can happen (**chaos**).

For a given type in CS, this corresponds to the probabilistic function which yields **normal distributions** (of outputs) for **every input** (cf. *'completely mixed states'*).

This tallies with the *partial order on classical and quantum states* discussed in the homonym chapter of (Coecke, 2011), whose maximal elements are the sharp functions.

Wrapping up

Summary:

- **Risky** software systems captured by probabilistic, monadic Mealy machines
- **Kleisli** categories reduce monadic complexity
- **LAoP** — exploiting the Kleisli category of the (countable support) distribution monad.

Wrapping up

Current work:

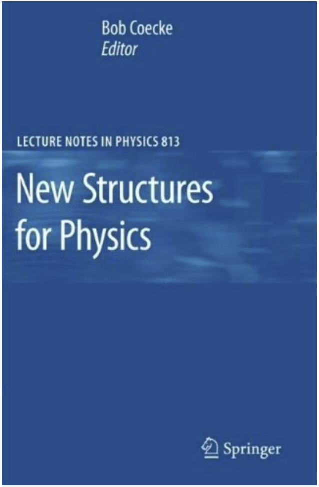
- Predict which machines are less likely to embark on **catastrophic** behaviour
- **Final** (behavioural) **semantics** of pMMM calls for infinite support distributions
- **Measure** theory — Kerstan and König (2012) provide an excellent starting point
- Are there "better" (less risky) software **architectures**?

Wrapping up

All in all...

On the right hand-side: what I think one should read before attempting doing PRA (probabilistic risk analysis) of software systems.

Enjoy your reading!



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

F. Bonchi, M. Bonsangue, M. Boreale, J. Rutten, and A. Silva. A coalgebraic perspective on linear weighted automata. *Inf. & Comp.*, 211:77–105, 2012.

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

V. Cortellessa and V. Grassi. A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In *Component-Based Software Engineering*, volume 4608 of *LNCS*, pages 140–156. 2007.

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

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.

H. Kerstan and B. König. Coalgebraic trace semantics for probabilistic transition systems based on measure theory. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR 2012*, LNCS, pages 410–424. Springer, 2012.

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

J.N. Oliveira. Typed linear algebra for weighted (probabilistic) automata. In *CIAA'12*, volume 7381 of *LNCS*, pages 52–65, 2012. doi: $10.1007/978\text{-}3\text{-}642\text{-}31606\text{-}7\_5$. .

Ana Sokolova. Probabilistic systems coalgebraically: A survey. *Theor. Comput. Sci.*, 412(38):5095–5110, 2011.

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