Relational Algebra for "Just Good Enough" Hardware

J.N. Oliveira

HASLab INESC TEC

INESC TEC & UNIVERSITY OF MINHO

RAMiCS 2014
Marienstatt im Westerwald, Germany
28 April - 1 May 2014

Motivation

Motivation

Software **V&V**

compared with...

Motivation

MITnews

engineering science management architecture + planning humanities, arts, and social sciences campus

The surprising usefulness of sloppy arithmetic

A computer chip that performs imprecise calculations could process some types of data thousands of times more efficiently than existing chips.

Larry Hardesty, MIT News Office

Sloppy arithmetic useful?

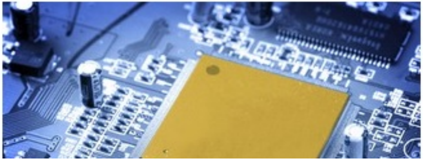
Horror!

But there is more. . .

"Just good enough" h/w

. . . coming from the land of the Swiss watch:

"We should stop designing perfect circuits"



02.10.13 - **Are integrated circuits "too good" for current technological applications? Christian Enz, the new Director of the Institute of Microengineering, backs the idea that perfection is overrated.**

Message:

> *Why* **perfection** *if (some)* **imperfection** *still meets the standards?*

S/w for "just good enough" h/w

What about **software** running over "just good enough" hardware?

Ready to **take the risk**?

Nonsense to run **safety critical** software on **defective** hardware?

Uups! — it seems "it already runs":

medical design *"IEC 60601-1 [brings]* ***risk management*** *into the very first stages of [product development]"*

Risk is everywhere — an inevitable (desired?) part of life.

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*

Interestingly,

medical design *"IEC 60601-1 [...]* **very first** *stages of [development]"*

From the very first stage in development

Think of things that **can go wrong**:

$$\mathit{bad} \cup \mathit{good}$$

How **likely**?

$$\mathit{bad}\ {}_{p}\!\diamond\ \mathit{good} \tag{1}$$

where

$$\mathit{bad}\ {}_{p}\!\diamond\ \mathit{good} \;=\; p \times \mathit{bad} + (1-p) \times \mathit{good}$$

for some **probability** p of *bad behaviour*, eg. the **imperfect** action

$$\mathit{top}\ {}_{(10^{-7})}\!\diamond\ \mathit{pop}$$

leaving a stack unchanged with 10^{-7} probability.

Imperfect truth tables

Imperfect **negation** $\mathit{id}\ {}_{0.01}\!\diamond\ \mathit{neg}$:

$$
\begin{aligned}
& \mathit{id}\ {}_{0.01}\!\diamond\ \mathit{neg} \\
= {} & 0.01 \times \left(\begin{array}{l|cc} & \textbf{False} & \textbf{True} \\ \hline \textbf{False} & 1 & 0 \\ \textbf{True} & 0 & 1 \end{array} \right) + 0.99 \times \left(\begin{array}{l|cc} & \textbf{False} & \textbf{True} \\ \hline \textbf{False} & 0 & 1 \\ \textbf{True} & 1 & 0 \end{array} \right) \\
= {} & \left(\begin{array}{l|cc} & \textbf{False} & \textbf{True} \\ \hline \textbf{False} & 0.01 & 0 \\ \textbf{True} & 0 & 0.01 \end{array} \right) + \left(\begin{array}{l|cc} & \textbf{False} & \textbf{True} \\ \hline \textbf{False} & 0 & 0.99 \\ \textbf{True} & 0.99 & 0 \end{array} \right) \\
= {} & \begin{array}{l|cc} & \textbf{False} & \textbf{True} \\ \hline \textbf{False} & 0.01 & 0.99 \\ \textbf{True} & 0.99 & 0.01 \end{array}
\end{aligned}
$$

Functions? Relations? Yes: **matrices**!

Better than the “anything can happen” **relation** $id \cup neg$, **matrix** $id \;{}_{p}\!\diamond\; neg$ carries useful **quantitative** information.

Aside: fragment of **function** $pres : President \rightarrow Country$ displayed as a **matrix** in the **Relational Mathematics** book (Schmidt, 2010).

Relational and **linear algebra** (LA) share a lot in common.

LA required when calculating **risk** of failure of **safety critical** s/w.

$$
\begin{array}{r c}
 & \begin{array}{ccccc} \text{US} & \text{French} & \text{German} & \text{British} & \text{Spanish} \end{array} \\
\begin{array}{r} \text{Clinton} \\ \text{Bush} \\ \text{Mitterand} \\ \text{Chirac} \\ \text{Schmidt} \\ \text{Kohl} \\ \text{Schröder} \\ \text{Thatcher} \\ \text{Major} \\ \text{Blair} \\ \text{Zapatero} \end{array} &
\begin{pmatrix}
1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{pmatrix}
\end{array}
$$

Linear algebra of programming

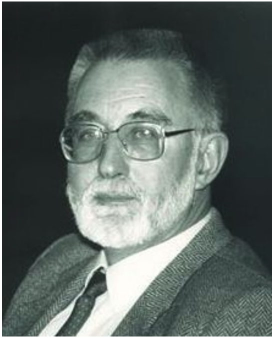
Relational / **KAT** algebra — a success story.

Linear algebra of programming (**LAoP**) — research track aiming at a quantitative extension of **heterogeneous** relational/KAT algebra.

Keeping the **pointfree** style!

Strategy: mild and pragmatic use of **categorial** techniques.

Main point — **Kleisli** categories matter!



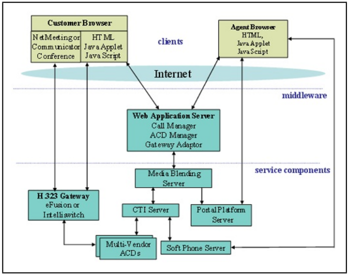
Heinrich Kleisli (1930-2011)

Context

Faults in CBS systems

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, e.g. component **call-graphs** (Cortellessa and Grassi, 2007).



Our starting point is a **coalgebraic** semantics for s/w components modeled as **monadic Mealy machines** (Barbosa and Oliveira, 2006).

Main ideas

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

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

where $\mathbb{F}$ is a **monad**.

Method = Elementary (single action) MMM.

CBS design = Algebra of MMM combinators.

Semantics = Coalgebraic, calculational.

To this framework we want to add analysis of

Risk = Probability of **faulty** (catastrophic) behaviour

Mealy machines in various guises

$\mathbb{F}$-transition structure:

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

Coalgebra:

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

State-monadic:

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

All versions useful in component algebra.

Abstracting from internal state S and branching effect $\mathbb{F}$, machine

$$m : S \times I \to \mathbb{F}(S \times O)$$

can be depicted as

$$\begin{tikzcd} I \arrow[d] \\ \boxed{\quad m \quad} \arrow[d] \\ O \end{tikzcd}$$

or as the **arrow** $I \xrightarrow{m} O$.

Mealy machines in various guises

$\mathbb{F}$-transition structure:

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

Coalgebra:

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

State-monadic:

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

All versions useful in component algebra.

Abstracting from internal state S and branching effect $\mathbb{F}$, machine

$$m : S \times I \to \mathbb{F}(S \times O)$$

can be depicted as

$$\begin{tikzcd} I \arrow[d] \\ \boxed{\quad m \quad} \arrow[d] \\ O \end{tikzcd}$$

or as the **arrow** $I \xrightarrow{m} O$.

Example — stack component

From a (**partial**) algebra of finite lists (Haskell syntax)

(partial) function	type
$push\ (s, a) = a : s$	$push :: ([a], a) \rightarrow [a]$
$pop = tail$	$pop :: [a] \rightarrow [a]$
$top = head$	$top :: [a] \rightarrow a$
$empty\ s = (length\ s = 0)$	$empty :: [a] \rightarrow \mathbb{B}$

to a collection of (**total**) methods (MMMs):

method	type
$push' = \eta \cdot (push \vartriangle !)$	$push' :: ([a], a) \rightarrow \mathbb{M}\ ([a], 1)$
$pop' = (pop \vartriangle top \Leftarrow (\neg \cdot empty)\ ) \cdot fst$	$pop' :: ([a], 1) \rightarrow \mathbb{M}\ ([a], a)$
$top' = (id \vartriangle top \Leftarrow (\neg \cdot empty)\ ) \cdot fst$	$top' :: ([a], 1) \rightarrow \mathbb{M}\ ([a], a)$
$empty' = \eta \cdot (id \vartriangle empty) \cdot fst$	$empty' :: ([a], 1) \rightarrow \mathbb{M}\ ([a], \mathbb{B})$

where...

Example — stack component

From a (**partial**) algebra of finite lists (Haskell syntax)

(partial) function	type
$push\ (s, a) = a : s$	$push :: ([a], a) \to [a]$
$pop = tail$	$pop :: [a] \to [a]$
$top = head$	$top :: [a] \to a$
$empty\ s = (length\ s = 0)$	$empty :: [a] \to \mathbb{B}$

to a collection of (**total**) methods (MMMs):

method	type
$push' = \eta \cdot (push \vartriangle !)$	$push' :: ([a], a) \to \mathbb{M}\ ([a], 1)$
$pop' = (pop \vartriangle top \Leftarrow (\neg \cdot empty)\ ) \cdot fst$	$pop' :: ([a], 1) \to \mathbb{M}\ ([a], a)$
$top' = (id \vartriangle top \Leftarrow (\neg \cdot empty)\ ) \cdot fst$	$top' :: ([a], 1) \to \mathbb{M}\ ([a], a)$
$empty' = \eta \cdot (id \vartriangle empty) \cdot fst$	$empty' :: ([a], 1) \to \mathbb{M}\ ([a], \mathbb{B})$

where. . .

Explanation

Pairing:

$$
\begin{tikzcd}
(a,b) \\
c \arrow[u, "(f \vartriangle g)\ c = (f\ c, b\ c)"']
\end{tikzcd}
\qquad
\begin{tikzcd}
a & (a,b) \arrow[l, "fst"'] \arrow[r, "snd"] & b \\
& c \arrow[ul, "f"] \arrow[u, "f \vartriangle g"'] \arrow[ur, "g"'] &
\end{tikzcd}
$$

"Sink" ("bang") function $A \xrightarrow{!} 1$ onto singleton type 1

$\mathbb{M}$: Monad with unit η and zero $\bot$ (typically **Maybe**)

$\mathbb{M}$ -totalizer on given pre-condition:

$$
\begin{array}{l}
\cdot \Leftarrow \cdot :: (a \rightarrow b) \rightarrow (a \rightarrow \mathbb{B}) \rightarrow a \rightarrow \mathbb{M}\ b \\
(f \Leftarrow p)\ a = \mathbf{if}\ p\ a\ \mathbf{then}\ (\eta \cdot f)\ a\ \mathbf{else}\ \bot
\end{array}
$$

Component $= \sum$ methods

Define

$$
\begin{aligned}
stack &:: ([a], 1 + 1 + a + 1) \to \mathbb{M}\ ([a], a + a + 1 + \mathbb{B}) \\
stack &= pop' \oplus top' \oplus push' \oplus empty'
\end{aligned}
$$

to obtain a **compound** $\mathbb{M}$ -MM (stack **component**) with 4 methods, where

- **input** 1 means "DO IT!"
- **output** 1 means "DONE!"

$$
\begin{tikzcd}
1 + 1 + a + 1 \arrow[d] \\
\boxed{stack} \arrow[d] \\
a + a + 1 + \mathbb{B}
\end{tikzcd}
$$

Notation $m \oplus n$ expresses the "coalesced" **sum** of two state-compatible MMMs (next slide).

Machine sums

Note the **pointfree** definition

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

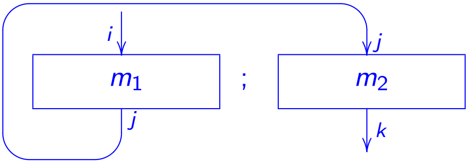
where dr° is the converse of isomorphism

$$\mathrm{dr} :: (s, i + j) \to (s, i) + (s, j)$$

and $\Delta :: \mathbb{F}\ a + \mathbb{F}\ b \to \mathbb{F}\ (a + b)$ is a kind of "cozip" operator.

S/w system = component **composition**

Forward **composition**



is central to component **communication**.

Abstracting from state, it means **composition** in a categorial sense:

$$\begin{tikzcd} i \arrow[r, "m_1"'] \arrow[rr, bend left, "m_2 \cdot m_1"] & j \arrow[r, "m_2"'] & k \end{tikzcd}$$

Exchange law

Formal definition of $m\, ;\, n$ to be discussed shortly.

For suitably typed MMM m_1 , m_2 , n_1 and n_2 , mind the useful **exchange law**

$$(m_1 \oplus m_2)\, ;\, (n_1 \oplus n_2) = (m_1\, ;\, n_1) \oplus (m_2\, ;\, n_2) \tag{2}$$

expressing two alternative approaches to s/w system construction:

- $\cdot \oplus \cdot$ -first — “component-oriented”
- $\cdot\, ;\, \cdot$ -first — “method-oriented”

For several other combinators in the algebra see (Barbosa and Oliveira, 2006).

Simulation (Haskell)

Let $\mathbb{M}$ instantiate to Haskell's `Maybe` monad:

- Running a **perfect** and **successful** composition:

  $$> (\mathit{pop}' \mathbin{;} \mathit{push}')\ (([1],[2]),())$$
  $$\mathit{Just}\ (([\,],[1,2]),())$$

- Running a **perfect** but **catastrophic** composition:

  $$> (\mathit{pop}' \mathbin{;} \mathit{push}')\ (([\,],[2]),())$$
  $$\mathit{Nothing}$$

  (source stack empty)

What about **imperfect** machine communication?

Imperfect components

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

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

Risk of $push'$ not pushing anything, with probability $1 - q$:

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

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

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

chooses between f and g according to p .

Faulty components

Define

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

where $\cdot \,;_D\, \cdot$ is a **probabilistic** enrichment of **composition** and run the same simulations for m_2 over the same state $([1],[2])$:

$$> m_2\ (([1],[2]),())$$
$$\begin{array}{r} Just\ (([\,],[1,2]),())\ 76.0\ \% \\ Just\ (([\,],[2]),())\ 19.0\ \% \\ Just\ (([1],[1,2]),())\ 4.0\ \% \\ Just\ (([1],[2]),())\ 1.0\ \% \end{array}$$

Total risk of faulty behaviour is 24% $(1 - 0.76\ )$ structured as:
(a) 1% — both stacks misbehave; (b) 19% — target stack misbehaves;
(c) 4% — source stack misbehaves.

Faulty components

As expected, the behaviour of

$$> m_2\ (([\,],[2]),())$$
$$\mathit{Nothing}\ 100.0\ \%$$

is 100% **catastrophic** (popping from an empty stack).

Simulation details:

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

Central topic

Our MMMs have become **probabilistic**, acquiring the general shape

$$S \times I \to \mathbb{D}\,(\mathbb{F}\,(S \times O))$$

where the additional $\mathbb{D}$ — (finite support) **distribution** monad — captures **imperfect** behaviour (fault propagation).

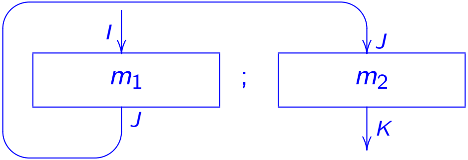
Questions:

- Shall we compose $\mathbb{D} \cdot \mathbb{F}$ and work over the **composite** monad?
- Or shall we try and find a way of working "as if $\mathbb{D}$ wasn't there"?

Let us first see how MMM compose.

MMM forward composition

Combinator



is defined by **Kleisli** composition

$$m_1 \,;\, m_2 \quad = \quad (\psi\ m_2) \bullet (\phi\ m_1)$$

of two steps:

- $\phi\ m_1$ — run m_1 "wrapped" with the state of m_2
- $\psi\ m_2$ — run m_2 "wrapped" with that of m_1 for the output it delivers

Kleisli composition

Let $X \xrightarrow{\eta} \mathbb{F}X \xleftarrow{\mu} \mathbb{F}^2 X$ be a **monad** in diagram

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

$f \bullet g$ denotes the so-called **Kleisli composition** of $\mathbb{F}$ -resultric arrows, forming a monoid with η as identity:

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

MMM composition — part I

Given $I \xrightarrow{m_1} J$ build $\phi\ m_1$:

$$\begin{tikzcd}
\mathbb{F}\,(S \times J) \times Q \arrow[d, "\tau_r"'] & (S \times I) \times Q \arrow[l, "m_1 \times id"'] & (S \times Q) \times I \arrow[l, "xr"'] \arrow[dll, "\phi\ m_1"] \\
\mathbb{F}\,((S \times J) \times Q) & &
\end{tikzcd}$$

where

- $\mathsf{xr} : (S \times Q) \times I \rightarrow (S \times I) \times Q$ is the obvious **isomorphism** ensuring the compound state and input I
- $\tau_r : (\mathbb{F}\ A) \times B \rightarrow \mathbb{F}\ (A \times B)$ is the right **strength** of monad $\mathbb{F}$, which therefore has to be a **strong** monad.

MMM composition — part II

Given $J \xrightarrow{m_2} K$ build $\psi\ m_2$:

$$
\begin{tikzcd}
S \times \mathbb{F}\,(Q \times K) \arrow[d, "\tau_l"'] & S \times (Q \times J) \arrow[l, "id \times m_2"'] & (S \times J) \times Q \arrow[l, "\mathsf{xl}"'] \arrow[lldd, "\psi\ m_2"] \\
\mathbb{F}\,(S \times (Q \times K)) \arrow[d, "\mathbb{F}\ \mathsf{a}^\circ"'] & & \\
\mathbb{F}\,((S \times Q) \times K) & &
\end{tikzcd}
$$

where

- a° is the converse of **isomorphism** $\mathsf{a} : (A \times B) \times C \to A \times (B \times C)$
- xl is a variant of xr
- $\tau_l : (B \times \mathbb{F}\ A) \to \mathbb{F}\,(B \times A)$ is the **left** strength of $\mathbb{F}$.

MMM composition — part III

Finally build $m_1 \,;\, m_2 = (\psi\ m_2) \bullet (\phi\ m_1)$:

$$\begin{tikzcd}
\mathbb{F}\,(\mathbb{F}\,((S \times Q) \times K)) \arrow[d, "\mu"'] & \mathbb{F}\,((S \times J) \times Q) \arrow[l, "\mathbb{F}\,(\psi\ m_2)"'] \arrow[d, dotted, no head] & \mathbb{F}\,((S \times Q) \times I) \arrow[l, "\phi\ m_1"'] \arrow[dll, bend left, "m_1 ; m_2"] \\
\mathbb{F}\,((S \times Q) \times K) & (S \times J) \times Q \arrow[l, "\psi\ m_2"'] &
\end{tikzcd}$$

This for **perfect** $\mathbb{F}$ -monadic machines. What about the **imperfect** ones?

What is the impact of adding probability-of-fault to the above construction? Does one need to rebuild the definition?

MMM composition — part III

Finally build $m_1\,;m_2 = (\psi\ m_2) \bullet (\phi\ m_1)$:

$$\begin{tikzcd}
\mathbb{F}\,(\mathbb{F}\,((S \times Q) \times K)) \arrow[d, "\mu"'] & \mathbb{F}\,((S \times J) \times Q) \arrow[l, "\mathbb{F}\,(\psi\ m_2)"'] \arrow[d, dash, dotted] & \mathbb{F}\,((S \times Q) \times I) \arrow[l, "\phi\ m_1"'] \arrow[dll, bend left, "m_1;m_2"] \\
\mathbb{F}\,((S \times Q) \times K) & (S \times J) \times Q \arrow[l, "\psi\ m_2"'] &
\end{tikzcd}$$

This for **perfect** $\mathbb{F}$ -monadic machines. What about the **imperfect** ones?

> *What is the impact of adding probability-of-fault to the above construction? Does one need to rebuild the definition?*

Doubly-monadic machines

Recall Haskell simulations running combinator $m_1 ;_D m_2$ for doubly-monadic machines of type

$$(S \times I) \to \mathbb{D}\,(\mathbb{M}\,(S \times O))$$

involving the Maybe ($\mathbb{M}$) and (finite support) **distribution** ($\mathbb{D}$) monads which generalize to

$$(S \times I) \to \mathbb{G}\,(\mathbb{F}\,(S \times O))$$

where, following the terminology of Hasuo et al. (2007):

- monad $X \xrightarrow{\eta_{\mathbb{F}}} \mathbb{F}\,X \xleftarrow{\mu_{\mathbb{F}}} \mathbb{F}^2 X$ caters for **transitional** effects (how the machine evolves)
- monad $X \xrightarrow{\eta_{\mathbb{G}}} \mathbb{G}\,X \xleftarrow{\mu_{\mathbb{G}}} \mathbb{G}^2 X$ specifies the **branching type** of the system.

Going relational

Doubly-monadic machines

Typical instance:

$\mathbb{G} = \mathbb{P}$ (powerset) and $\mathbb{F} = \mathbb{M} = (1+)$ ('maybe'), that is,

$$m : Q \times I \rightarrow \mathbb{P}\,(1 + Q \times J)$$

is a reactive, **non-deterministic** *finite state automaton with explicit termination.*

Such machines can be regarded as **binary** relations of (relational) type

$$(Q \times I) \rightarrow (1 + Q \times J)$$

and handled directly in **relational algebra**. (Details in the next slide)

Nondeterministic Maybe machines

The **power** transpose adjunction

$$R = \lceil m \rceil \quad \Leftrightarrow \quad \langle \forall\ b, a\ ::\ b\ R\ a = b\ \in\ m\ a \rangle$$

for trading between $\mathbb{P}$ -**functions** and **binary relations**, in a way such that

$$\begin{tikzcd} A \to \mathbb{P}\ B \arrow[r, bend left, "\lceil \cdot \rceil"] \arrow[r, phantom, "\cong"] & A \to B \arrow[l, bend left, "\lfloor \cdot \rfloor"] \end{tikzcd}$$

$$\lceil m \bullet n \rceil = \lceil m \rceil \cdot \lceil n \rceil$$

where

- $m \bullet n$ — Kleisli composition of $\mathbb{P}$ -functions
- $\lceil m \rceil \cdot \lceil n \rceil$ — **relational** composition

$$b\ (R \cdot S)\ a \quad \Leftrightarrow \quad \langle \exists\ c\ ::\ b\ R\ c \wedge c\ S\ a \rangle$$

of the corresponding *binary relations*.

Composing relational $\mathbb{M}$ -machines

Transition monad on duty is $\mathbb{M} = (1+)$, ie.

$$X \xrightarrow{i_2} 1 + X \xleftarrow{[i_1, id]} 1 + (1 + X)$$

(i_1 , i_2 = binary sum injections).

Lifting: in the original definition

$$m_1 \,;\, m_2 = (\psi \; m_2) \bullet (\phi \; m_1)$$

run **Kleisli** composition **relationally**:

$$\begin{tikzcd}
1 + (1 + C) \arrow[d, "{[i_1, id]}"'] & 1 + B \arrow[l, "id+R"'] \arrow[d, dotted, no head] & A \arrow[l, "S"'] \arrow[dll, bend left, "R \bullet S"] \\
1 + C & B \arrow[l, "R"'] &
\end{tikzcd}$$

$$\begin{aligned} R \bullet S &= [i_1, id] \cdot (id + R) \cdot S = [i_1, R] \cdot S \\ &= i_1 \cdot i_1^\circ \cdot S \cup R \cdot i_2^\circ \cdot S \end{aligned}$$

Composing relational $\mathbb{M}$ -machines

Pointwise: $y\ (R \bullet S)\ a$ holds iff

$$(y = *) \wedge (*\ S\ a) \vee \langle \exists\ c\ ::\ (y\ R\ c) \wedge ((i_2\ c)\ S\ a) \rangle$$

where $* = i_1\ \bot$

In words:

$R \bullet S$ doomed to fail if S fails;
Otherwise, $R \bullet S$ will fail where R fails.
For the same input, $R \bullet S$ may both succeed or fail.

Summary: Nondeterministic $\mathbb{M}$ -machines are $\mathbb{M}$ **-relations** and original (deterministic) definition is "reused" in the relational setting:

$$R_1 \mathbin{;} R_2 = (\psi\ R_2) \bullet (\phi\ R_1) = [i_1, \psi\ R_2] \cdot (\phi\ R_1)$$

Going linear

Probabilistic branching ($\mathbb{D}$ instead of $\mathbb{P}$)

Again, instead of working in Set,

$$\begin{tikzcd} & \mathbb{D}\,(\mathbb{F}\,B) \arrow[d, dotted, no head] & A \arrow[l, "g"'] \\ \mathbb{D}\,(\mathbb{F}\,C) & B \arrow[l, "f"'] & \end{tikzcd}$$

we seek to implement $\mathbb{F}$ -Kleisli-composition in the Kleisli category of $\mathbb{D}$, that is

$$\begin{tikzcd} & \mathbb{F}\,B \arrow[d, dotted, no head] & A \arrow[l, "\lceil g \rceil"] \arrow[dll, bend right, "\lceil f \rceil \bullet \lceil g \rceil"'] \\ \mathbb{F}\,C & B \arrow[l, "\lceil f \rceil"'] & \end{tikzcd}$$

thus "abstracting from" monad $\mathbb{D}$.

Question: Kleisli($\mathbb{D}$) = **??**

Probabilistic monadic machines

It turns out to be the (monoidal) category of column-stochastic (CS) **matrices**, cf. adjunction

$$\begin{tikzcd} A \to_{Set} \mathbb{D}B \arrow[rr, bend left, "\lceil \cdot \rceil"] \arrow[rr, phantom, "\cong"] & & A \to_{CS} B \arrow[ll, bend left, "\lfloor \cdot \rfloor"] \end{tikzcd}$$

such that

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

where $A \to_{CS} B$ is the **matrix type** of all matrices with B-indexed rows and A-indexed columns all adding up to 1 (100%).

Important:

CS *represents the* **Kleisli** *category of* $\mathbb{D}$

Probabilism versus matrix algebra

Recall probabilistic **negation** function

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

which corresponds to matrix

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

where **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

In general, category of matrices over a semi-ring $(\mathbb{S}; +, \times, 0, 1)$:

- **Objects** are types (A , B , ...) and **morphisms** ($M : A \to B$) are matrices whose columns have finite support.

- **Composition**:

$$\begin{tikzcd} B & A \arrow[l, "M"'] & C \arrow[l, "N"'] \arrow[ll, bend left, "C=M\cdot N"] \end{tikzcd}$$

  that is:

$$b\,(M \cdot N)\,c \;=\; \langle \sum\ a\ ::\ (r\,M\,a) \times (a\,N\,c) \rangle$$

- **Identity**: the diagonal Boolean matrix $id : A \to A$.

Typed linear algebra

Matrix **coproducts**

$$(A + B) \to C \quad \cong \quad (A \to C) \times (B \to C)$$

where $A + B$ is disjoint union, cf. **universal property**

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

where $[i_1|i_2] = id$.

$[M|N]$ is one of the basic matrix **block** combinators — it puts M and N side by side and is such that

$$[M|N] = M \cdot i_1^\circ + N \cdot i_2^\circ$$

as in **relation algebra**.

Typed linear algebra

Matrix **direct sum**

$$M \oplus N = \left[\begin{array}{c|c} M & 0 \\ \hline 0 & N \end{array}\right]$$

is an (endo,bi)functor, cf.

$$\begin{array}{rcl} (id \oplus id) & = & id \\ (M \oplus N) \cdot (P \oplus Q) & = & (M \cdot P) \oplus (N \cdot Q) \\ [M|N] \cdot (P \oplus Q) & = & [M \cdot P | N \cdot Q] \end{array}$$

as in **relation algebra** — etc, etc.

The Maybe monad in the category is therefore given by $\mathbb{M} = (id \oplus \cdot)$

Another "Kleisli shift"

As we did for relations representing Kleisli($\mathbb{P}$), let us encode $\mathbb{M}$ -Kleisli composition in matrix form:

$$M \bullet N = [i_1 | M] \cdot N$$

$$\begin{tikzcd}
1 + (1 + C) \arrow[d, "{[i_1|id]}"'] & 1 + B \arrow[l, "id \oplus M"'] \arrow[d, dotted, no head] & A \arrow[l, "N"'] \arrow[dll, bend left, "M \bullet N"] \\
1 + C & B \arrow[l, "M"'] &
\end{tikzcd}$$

Thus $M \bullet N = i_1 \cdot i_1^{\circ} \cdot N + M \cdot i_2^{\circ} \cdot N$ leading into the pointwise

$$y\ (M \bullet N)\ a = \\ (y = *) \times (*\ N\ a) + \langle \sum\ b\ ::\ (y\ M\ b) \times ((i_2\ b)\ N\ a) \rangle$$

— compare with the relational version and example (next slide).

Another “Kleisli shift”

Example:

Probabilistic $\mathbb{M}$ -Kleisli composition $M \bullet N$ of matrices $N : \{ a_1, a_2, a_3 \} \rightarrow 1 + \{ c_1, c_2 \}$ and $M : \{ c_1, c_2 \} \rightarrow 1 + \{ b_1, b_2 \}$.

Injection $i_1 : 1 \rightarrow 1 + \{ b_1, b_2 \}$ is the leftmost column vector.

				a1	a2	a3
			*	0.5	0	0
			c1	0.5	1	0.7
	*	c1	c2	0	0	0.3
*	1	0.2	0	0.6	0.2	0.14
b1	0	0	0.6	0	0	0.18
b2	0	0.8	0.4	0.4	0.8	0.68

Example: for input a_1 there is 60% probability of $M \bullet N$ failing = either N fails (50%) or passes c_1 to M (50%) which fails with 20% probability.

Probabilistic MMM (=pMMM) as matrices

Similarly to relations before, we can think of **probabilistic** $\mathbb{M}$ -monadic Mealy machines as CS matrices which communicate (as matrices) as follows

$$N\,;M = [i_1|(id \oplus \mathsf{a}^\circ) \cdot \tau_l \cdot (id \otimes M) \cdot \mathsf{xl}] \cdot \tau_r \cdot (N \otimes id) \cdot \mathsf{xr} \quad (3)$$

where

- **functions** are represented matricially by **Dirac** distributions;
- relational product becomes matrix **Kronecker product**

$$(y, x)(M \otimes N)(b, a) \quad = \quad (yMb) \times (xNa)$$

NB: Haskell implementation of pMMM composition follows (3).

Kleisli shift

Monad-monad lifting

For the above to make sense for machines of **generic** type $Q \times I \rightarrow \mathbb{G}\,(\mathbb{F}\,(Q \times J))$ make sure that

---

The lifting of monad $\mathbb{F}$ by monad $\mathbb{G}$ still is a monad in the Kleisli category of $\mathbb{G}$.

Recall:

- $\mathbb{F}$ — **transition** monad
- $\mathbb{G}$ — **branching** monad

Mind their different roles:

Branching *monad "hosts"* **transition** *monad.*

Monad-monad lifting

In general, given two monads

$$X \xrightarrow{\eta_{\mathbb{G}}} \mathbb{G}X \xleftarrow{\mu_{\mathbb{G}}} \mathbb{G}^2X \quad (\textit{the } \textbf{host})$$

$$X \xrightarrow{\eta_{\mathbb{F}}} \mathbb{F}X \xleftarrow{\mu_{\mathbb{F}}} \mathbb{F}^2X \quad (\textit{the } \textbf{guest})$$

in a category $\mathbf{C}$:

- let $\mathbf{C}^\flat$ denote the Kleisli category induced by host $\mathbb{G}$;
- let $B \xleftarrow{f^\flat} A$ be the morphism in $\mathbf{C}^\flat$ corresponding to $\mathbb{G}B \xleftarrow{f} A$ in $\mathbf{C}$;
- define

$$f^\flat \cdot g^\flat = (f \bullet g)^\flat = (\mu_{\mathbb{G}} \cdot \mathbb{G}\, f \cdot g)^\flat$$

Monad-monad lifting

For *any* morphism $B \xleftarrow{f} A$ in $\mathbf{C}$ define its lifting to $\mathbf{C}^\flat$ by

$$\overline{f} = (\eta_{\mathbb{G}} \cdot f)^\flat \tag{4}$$

As in (Hasuo et al., 2007), assume **distributive law**

$$\lambda : \mathbb{F}\mathbb{G} \to \mathbb{G}\mathbb{F}$$

Lift the **guest** endofunctor $\mathbb{F}$ from $\mathbf{C}$ to $\mathbf{C}^\flat$ by defining $\overline{\mathbb{F}}$ as follows, for $\mathbb{G}\ B \xleftarrow{f} A$:

$$\overline{\mathbb{F}}(f^\flat) \ = \ (\lambda \cdot \mathbb{F}\, f)^\flat$$

cf. diagram

$$\mathbb{G}\mathbb{F}B \xleftarrow{\lambda} \mathbb{F}\mathbb{G}B \xleftarrow{\mathbb{F}\, f} \mathbb{F}\, A$$

Monad-monad lifting

For $\overline{\mathbb{F}}$ to be a **functor** in $\mathbf{C}^\flat$ two conditions must hold (Hasuo et al., 2007):

$$\begin{aligned} \lambda \cdot \mathbb{F}\,\eta_{\mathbb{G}} &= \eta_{\mathbb{G}} \\ \lambda \cdot \mathbb{F}\,\mu_{\mathbb{G}} &= \mu_{\mathbb{G}} \cdot \mathbb{G}\lambda \cdot \lambda \end{aligned}$$

We need to find extra conditions for *guest* $\mathbb{F}$ to lift to a **monad** in $\mathbf{C}^\flat$; that is,

$$X \xrightarrow{\overline{\eta_{\mathbb{F}}}=(\eta_{\mathbb{G}}\cdot\eta_{\mathbb{F}})^\flat} \overline{\mathbb{F}}X \xleftarrow{\overline{\mu_{\mathbb{F}}}=(\eta_{\mathbb{G}}\cdot\mu_{\mathbb{F}})^\flat} \overline{\mathbb{F}}^2 X$$

should be a monad in $\mathbf{C}^\flat$.

The standard monadic laws, e.g. $\overline{\mu_{\mathbb{F}}} \cdot \overline{\eta_{\mathbb{F}}} = id$, hold via lifting (4) and Kleisli composition laws.

Monad-monad lifting

The remaining **natural** laws,

$$(\overline{\mathbb{F}}\ f^\flat)\cdot\overline{\eta_{\mathbb{F}}} = \overline{\eta_{\mathbb{F}}}\cdot f^\flat$$
$$(\overline{\mathbb{F}}\ f^\flat)\cdot\overline{\mu_{\mathbb{F}}} = \overline{\mu_{\mathbb{F}}}\cdot(\overline{\mathbb{F}}^2\ f^\flat)$$

are ensured by two “monad-monad” compatibility conditions:

$$\begin{aligned}\lambda\cdot\eta_{\mathbb{F}} &= \mathbb{G}\eta_{\mathbb{F}}\\ \lambda\cdot\mu_{\mathbb{F}} &= \mathbb{G}\mu_{\mathbb{F}}\cdot\lambda\cdot\mathbb{F}\lambda\end{aligned}$$

that is:

$$\begin{tikzcd}
\mathbb{G}X \arrow[r, "\eta_{\mathbb{F}}"] \arrow[dr, "\mathbb{G}\eta_{\mathbb{F}}"'] & \mathbb{F}\mathbb{G}X \arrow[d, "\lambda"'] & & \mathbb{F}^2(\mathbb{G}X) \arrow[ll, "\mu_{\mathbb{F}}"'] \arrow[d, "\mathbb{F}\lambda"] \\
& \mathbb{G}\mathbb{F}X & \mathbb{G}(\mathbb{F}^2X) \arrow[l, "\mathbb{G}\mu_{\mathbb{F}}"] & \mathbb{F}\mathbb{G}\mathbb{F}X \arrow[l, "\lambda"]
\end{tikzcd}$$

(Details in the paper.)

Pairing!

Not yet done!

There is a price to pay for the "hosting" process.

Definition of $\lceil m_1 \rceil\ \mathbf{;}\ \lceil m_2 \rceil$ is **strongly** monadic.

Question:

Do **strong** *monads* **lift** *to strong monads?*

Recall the types of the two **strengths**:

$$\tau_l : (B \times \mathbb{F}\ A) \to \mathbb{F}\ (B \times A)$$
$$\tau_r : (\mathbb{F}\ A \times B) \to \mathbb{F}\ (A \times B)$$

The basic properties, e.g. $\mathbb{F}\ \mathit{lft} \cdot \tau_r = \mathit{lft}$ and $\mathbb{F}\ \mathrm{a}^\circ \cdot \tau_r = \tau_r \cdot (\tau_r \times id) \cdot \mathrm{a}^\circ$ are preserved by their liftings (e.g. $\overline{\tau_r}$) by construction.

Naturality vs lifting

So, what may fail is their **naturality**, e.g.

$$\overline{\tau_l} \cdot (N \otimes \overline{\mathbb{F}}\ M) = \overline{\mathbb{F}}\ (N \otimes M) \cdot \overline{\tau_l}$$

where M and N are arbitrary CS matrices and $\cdot \otimes \cdot$ is Kronecker product.

Naturality *is essential to pointfree proofs!*

Example: for $\mathbb{F} = \mathbb{M} = (1+)$ we have e.g. $\overline{\tau_l} = (\overline{!} \oplus id) \cdot \overline{\mathrm{dr}}$, that is

$$1 + A \times B \xleftarrow{! \oplus id} (1 \times B) + (A \times B) \xleftarrow{\mathrm{dr}} (1 + A) \times B$$

dropping the $\overline{f}$ bars over functions for easier reading.

Naturality which lifts

Is $\overline{!} \oplus id$ natural? We check:

$$
\begin{array}{ll}
 & (id \oplus N) \cdot (! \oplus id) = (! \oplus id) \cdot (M \oplus N) \\
\Leftrightarrow & \quad \{ \text{ bifunctor } \cdot \oplus \cdot \ \} \\
 & ! \oplus N = (! \cdot M) \oplus N \\
\Leftrightarrow & \quad \{ \ ! \cdot M = ! \text{ because } M \text{ is a CS matrix } \} \\
 & \mathit{true}
\end{array}
$$

Note: matrix M is CS iff $! \cdot M = !$ holds. (Thus composition is closed over CS-matrices.)

Naturality which does not lift

Is the diagonal function $\delta = id \vartriangle id$ — that is

$$\delta\ x = (x, x)$$

still natural once lifted to matrices?

No! Diagram

$$\begin{tikzcd} A \times A \arrow[d, "M \otimes M"'] & A \arrow[l, "\delta"'] \arrow[d, "M"] \\ B \times B & B \arrow[l, "\delta"'] \end{tikzcd}$$

does not commute for every CS matrix $M : A \to B$ — counter-example in the next slide.

Naturality which does not lift

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

Probabilistic pairing

This happens because the Kleisli-lifting of **pairing**

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

is a **weak**-product for column stochastic matrices:

$$X = M \vartriangle N \quad \Rightarrow \quad \left\{ \begin{array}{r} fst \cdot X = M \\ snd \cdot X = N \end{array} \right. \tag{5}$$

ie. ($\Leftarrow$) is not guaranteed

So $(fst \cdot X) \vartriangle (snd \cdot X)$ differs from X in general.

In LA, $M \vartriangle N$ is known as the **Khatri-Rao** matrix product.
In RA, $R \vartriangle S$ is known as the **fork** operator.

Probabilistic pairing

In summary: weak product (5) still grants the **cancellation** rule,

$$fst \cdot (M \vartriangle N) = M \ \wedge\ snd \cdot (M \vartriangle N) = N$$

cf. e.g.

$$\begin{tikzcd}[column sep=huge, row sep=huge]
2 & 2 \times 3 \arrow[l, "{fst=\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}}"'] \arrow[r, "{snd=\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}}"] & 3 \\
 & 4 \arrow[ul, "{M=\begin{bmatrix} 0.5 & 0.3 & 0 & 0.75 \\ 0.5 & 0.7 & 1 & 0.25 \end{bmatrix}}"] \arrow[u, "{M \vartriangle N=\begin{bmatrix} 0.15 & 0.12 & 0 & 0 \\ 0.35 & 0.06 & 0 & 0.75 \\ 0 & 0.12 & 0 & 0 \\ 0.15 & 0.28 & 0.1 & 0 \\ 0.35 & 0.14 & 0.2 & 0.25 \\ 0 & 0.28 & 0.7 & 0 \end{bmatrix}}"] \arrow[ur, "{N=\begin{bmatrix} 0.3 & 0.4 & 0.1 & 0 \\ 0.7 & 0.2 & 0.2 & 1 \\ 0 & 0.4 & 0.7 & 0 \end{bmatrix}}"'] &
\end{tikzcd}$$

Probabilistic pairing

... but **reconstruction**

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

doesn't hold in general, cf. e.g.

$$X : 2 \rightarrow 2 \times 3$$

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

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

This is not surprising (cf. RA) but creates difficulties and needs attention.

Closing

Research proposal

Need to quantify software (un)reliability in presence of faults.

Need for **weighted** nondeterminism, e.g. **probabilism**.

Relation algebra $\rightarrow$ Matrix algebra

Usual strategy:

"Keep category (sets), change definition"

Proposed strategy:

"Keep definition, change category"

Change category

Possible wherever semantic models are structured around a pair $(\mathbb{F}, \mathbb{G})$ of monads:

Monad	$\mathbb{F}$	$\mathbb{G}$
Effect	Transition	Branching
Role	Guest	Host
Strategy	Lifted	"Kleislified"

Works nicely for those $\mathbb{G}$ for which well-established Kleisli categories are known, for instance (aside):

$\mathbb{G}$	Kleisli
$\mathbb{P}$	Relation algebra
Vec	Matrix algebra
$\mathbb{D}$	Stochastic matrices
Giry	Stochastic relations

cf. (Panangaden, 2009) etc.

Future work

- **LAoP** in its infancy — really a lot to do!
- Relation to **quantum** physics — cf. 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.*

- **Final** (behavioural) **semantics** of pMMM calls for infinite support distributions.
- **Measure** theory — Kerstan and König (2012) provide an excellent starting point.
- Case studies!

Verification of IBM 4765

Marić and Sprenger (2014) rely on MMM of type

$$(Q \times A) \to \mathbb{P}\,((2 + V) \times Q)$$

for verifying a persistent memory manager (in IBMs 4765 secure coprocessor) in face of **restarts** and **hardware failures**, where

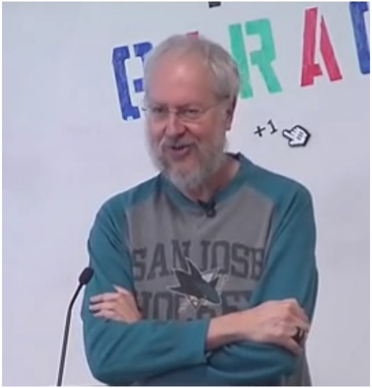
- V - (normal) return values
- 2 - exceptions (either "regular" or "restarts")

Interested in scaling up $\mathbb{P}$ to $\mathbb{D}$ and do the proofs using (pointfree!) matrix algebra where they use explicit monad **transformers** etc, etc (Isabelle).

The monadic "curse"

"Monads [...] come with a curse. The monadic curse is that once someone learns what monads are and how to use them, they lose the ability to explain it to other people"

(Douglas Crockford: Google Tech Talk on how to express monads in JavaScript, 2013)



Douglas Crockford (2013)

References

L.S. Barbosa and J.N. Oliveira. Transposing Partial Components — an Exercise on Coalgebraic Refinement. *Theor. Comp. Sci.*, 365(1):2–22, 2006.

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.

I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4):1–36, 2007. doi: 10.2168/LMCS-3(4:11)2007.

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.

O. Marić and C. Sprenger. Verification of a transactional memory manager under hardware failures and restarts, 2014. To appear in FM'14.

J.N. Oliveira. A relation-algebraic approach to the "Hoare logic" of functional dependencies. *JLAP*, 2014a. .

J.N. Oliveira. Relational algebra for "just good enough" hardware. In *RAMiCS*, volume 8428 of *LNCS*, pages 119–138. Springer Berlin / Heidelberg, 2014b. .

P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

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

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