Relational Algebra for “Just Good Enough” Hardware

J.N. Oliveira

INESC Tec & University of Minho

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

Software V&V

compared with...
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”:

“IEC 60601-1 [brings] risk management into the very first stages of [product development]”

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

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,

"IEC 60601-1 [...] very first stages of [development]"
From the very first stage in development

Think of things that can go wrong:

\[ \text{bad} \cup \text{good} \]

How likely?

\[ \text{bad} \ p \text{ } \Diamond \text{ } \text{good} \]

where

\[ \text{bad} \ p \text{ } \Diamond \text{ } \text{good} = p \times \text{bad} + (1 - p) \times \text{good} \]

for some probability \( p \) of bad behaviour, eg. the imperfect action

\[ \text{top} \ (10^{-7}) \Diamond \text{pop} \]

leaving a stack unchanged with \( 10^{-7} \) probability.
Imperfect truth tables

Imperfect **negation** \(id \ 0.01 \diamond neg\):

\[
id \ 0.01 \diamond neg = 0.01 \times \begin{pmatrix} \text{False} & \text{True} \\ \text{False} & 1 & 0 \\ \text{True} & 0 & 1 \end{pmatrix} + 0.99 \times \begin{pmatrix} \text{False} & \text{True} \\ \text{False} & 0 & 1 \\ \text{True} & 1 & 0 \end{pmatrix}
\]

\[
= \begin{pmatrix} \text{False} & \text{True} \\ \text{False} & 0.01 & 0 \\ \text{True} & 0 & 0.01 \end{pmatrix} + \begin{pmatrix} \text{False} & \text{True} \\ \text{False} & 0 & 0.99 \\ \text{True} & 0.99 & 0 \end{pmatrix}
\]

\[
= \begin{pmatrix} \text{False} & \text{True} \\ \text{False} & 0.01 & 0.99 \\ \text{True} & 0.99 & 0.01 \end{pmatrix}
\]
Functions? Relations? Yes: matrices!

Better than the “anything can happen” relation $id \cup neg$, matrix $id \triangleright neg$ 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.
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 categorical 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 \( F \)-evolving transition structure of type:

\[
S \times I \to F(S \times O)
\]

where \( F \) 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

\[ F\text{-transition structure:} \]

\[ S \times I \rightarrow F(S \times O) \]

Coalgebra:

\[ S \rightarrow (F(S \times O))^I \]

State-monadic:

\[ I \rightarrow (F(S \times O))^S \]

All versions useful in component algebra.

Abstracting from internal state \( S \) and branching effect \( F \), machine

\[ m : S \times I \rightarrow F(S \times O) \]

can be depicted as

\[
\begin{array}{c}
I \\
\downarrow \\
m \\
\downarrow \\
O
\end{array}
\]

or as the arrow \( I \xrightarrow{m} O \).
Mealy machines in various guises

\[ \text{\textbf{F}-transition structure:} \]
\[ S \times I \to \text{\textbf{F}}(S \times O) \]

Coalgebra:
\[ S \to (\text{\textbf{F}}(S \times O))^I \]

State-monadic:
\[ I \to (\text{\textbf{F}}(S \times O))^S \]

Abstracting from internal state \( S \) and branching effect \( \text{\textbf{F}} \), machine
\[ m : S \times I \to \text{\textbf{F}}(S \times O) \]
can be depicted as
\[ I \]
\[ \downarrow \]
\[ m \]
\[ \downarrow \]
\[ O \]

or as the arrow \( I \xrightarrow{m} O \).

All versions useful in component algebra.
Example — stack component

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

<table>
<thead>
<tr>
<th>(partial) function</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>push ((s, a) = a : s)</td>
<td>push :: ([a], a) \to [a]</td>
</tr>
<tr>
<td>pop = tail</td>
<td>push :: [a] \to [a]</td>
</tr>
<tr>
<td>top = head</td>
<td>top :: [a] \to a</td>
</tr>
<tr>
<td>empty (s \equiv (\text{length } s = 0))</td>
<td>empty :: [a] \to \bb</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>method</th>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>push' = (\eta \cdot (\text{push} \triangle !))</td>
<td>push' :: ([a], a) \to \mathbb{M} ([a], 1)</td>
</tr>
<tr>
<td>pop' = (pop \triangle top \leftarrow (\neg \cdot \text{empty}) \cdot \text{fst}</td>
<td>pop' :: ([a], 1) \to \mathbb{M} ([a], a)</td>
</tr>
<tr>
<td>top' = (id \triangle top \leftarrow (\neg \cdot \text{empty}) \cdot \text{fst}</td>
<td>top' :: ([a], 1) \to \mathbb{M} ([a], a)</td>
</tr>
<tr>
<td>empty' = (\eta \cdot (id \triangle \text{empty}) \cdot \text{fst}</td>
<td>empty' :: ([a], 1) \to \mathbb{M} ([a], \bb)</td>
</tr>
</tbody>
</table>

where...
Example — stack component

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

\[
\begin{array}{ll}
\text{(partial) function} & \text{type} \\
\text{push } (s, a) = a : s & \text{push} :: ([a], a) \rightarrow [a] \\
pop = \text{tail} & \text{pop} :: [a] \rightarrow [a] \\
top = \text{head} & \text{top} :: [a] \rightarrow a \\
\text{empty } s = (\text{length } s = 0) & \text{empty} :: [a] \rightarrow \mathbb{B}
\end{array}
\]

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

\[
\begin{array}{ll}
\text{method} & \text{type} \\
push' = \eta \cdot (\text{push} \uplus !) & push' :: ([a], a) \rightarrow \mathbb{M} ([a], 1) \\
pop' = (\text{pop} \uplus top \leftarrow (\neg \cdot \text{empty}) ) \cdot \text{fst} & pop' :: ([a], 1) \rightarrow \mathbb{M} ([a], a) \\
top' = (\text{id} \uplus top \leftarrow (\neg \cdot \text{empty}) ) \cdot \text{fst} & top' :: ([a], 1) \rightarrow \mathbb{M} ([a], a) \\
\text{empty'} = \eta \cdot (\text{id} \uplus \text{empty}) \cdot \text{fst} & \text{empty'} :: ([a], 1) \rightarrow \mathbb{M} ([a], \mathbb{B})
\end{array}
\]

where...
Pairing:

\[(a, b) \quad a \xla{\text{fst}} (a, b) \xla{\text{snd}} b\]

\[(f \triangle g) \quad c = (f \circ c, b \circ c)\]

“Sink” (“bang”) function \[A \xrightarrow{!} 1\] onto singleton type \[1\]

\[\mathcal{M}: \text{Monad with unit } \eta \text{ and zero } \bot \text{ (typically } \text{Maybe})\]

\[\mathcal{M} - \text{totalizer on given pre-condition}:
\begin{align*}
\cdot & \iff \cdot \circ (a \rightarrow b) \rightarrow (a \rightarrow \mathcal{B}) \rightarrow a \rightarrow \mathcal{M} b \\
(f & \iff p) a = \text{if } p a \text{ then } (\eta \cdot f) a \text{ else } \bot
\end{align*}\]
Component = \sum \text{ methods}

Define

\[ stack :: ([a], 1 + 1 + a + 1) \rightarrow M ([a], a + a + 1 + B) \]
\[ stack = \text{pop'} \oplus \text{top'} \oplus \text{push'} \oplus \text{empty'} \]

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

- \text{input 1} means “DO IT!”
- \text{output 1} means “DONE!”

Notation \( m \oplus n \) expresses the “coalesced” \text{ sum} of two state-compatible MMMs (next slide).
Machine sums

Note the **pointfree** definition

\[
\bullet \oplus \bullet :: (Functor F) \Rightarrow
\quad -- \text{input machines}
((s, i) \rightarrow F (s, o)) \rightarrow
((s,j) \rightarrow F (s, p)) \rightarrow
\quad -- \text{output machine}
(s, i + j) \rightarrow F (s, o + p)
\quad -- \text{definition}
m_1 \oplus m_2 = (F dr^\circ) \cdot \Delta \cdot (m_1 + m_2) \cdot dr
\]

where \(dr^\circ\) is the converse of isomorphism

\[
dr :: (s, i + j) \rightarrow (s, i) + (s,j)
\]

and \(\Delta :: F a + F b \rightarrow F (a + b)\) is a kind of “cozip” operator.
Forward composition

is central to component communication.

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

$$ (m_1 \oplus m_2) ; (n_1 \oplus n_2) = (m_1 ; n_1) \oplus (m_2 ; n_2) $$

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 $M$ instantiate to Haskell’s $\texttt{Maybe}$ monad:

- Running a **perfect** and **successful** composition:
  
  $$> (\texttt{pop'} ; \texttt{push'}) ((([1], [2]), ()))$$
  $$\texttt{Just} ((([]), [1, 2]), ())$$

- Running a **perfect** but **catastrophic** composition:
  
  $$> (\texttt{pop'} ; \texttt{push'}) ((([]), [2]), ())$$
  $$\texttt{Nothing}$$

(source stack empty)

What about **imperfect** machine communication?
Imperfect components

Risk of \( \text{pop'} \) behaving like \( \text{top'} \) with probability \( 1 - p \):

\[
\text{pop}'' :: \mathbb{P} \to ([a], 1) \to \mathbb{D} (\text{M} ([a], a)) \\
\text{pop}'' p = \text{pop'} \ p \circledast \text{top}'
\]

Risk of \( \text{push'} \) not pushing anything, with probability \( 1 - q \):

\[
\text{push}'' :: \mathbb{P} \to ([a], a) \to \mathbb{D} (\text{M} ([a], 1)) \\
\text{push}'' q = \text{push'} \ q \circledast !
\]

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

\[
\cdot \circledast \cdot :: \mathbb{P} \to (t \to a) \to (t \to a) \to t \to \mathbb{D} a \\
(f \ p \circledast g) \ x = \text{choose} \ p \ (f \ x) \ (g \ x)
\]

chooses between \( f \) and \( g \) according to \( p \).
Faulty components

Define

\[ m_2 = \text{pop}'' 0.95 ;D \cdot \text{push}'' 0.8 \]

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

\[
\begin{align*}
> m_2 & \left( ([1], [2]), () \right) \\
\text{Just} & \left( ([], [1, 2]), () \right) 76.0 \% \\
\text{Just} & \left( ([], [2]), () \right) 19.0 \% \\
\text{Just} & \left( ([1], [1, 2]), () \right) 4.0 \% \\
\text{Just} & \left( ([1], [2]), () \right) 1.0 \%
\end{align*}
\]

**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]], ()) \) \]

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

\[ S \times I \rightarrow \mathcal{D} (F (S \times O)) \]

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

**Questions:**

- Shall we compose \( \mathcal{D} \cdot F \) and work over the **composite** monad?
- Or shall we try and find a way of working “as if \( \mathcal{D} \) wasn’t there”?

Let us first see how MMM compose.
MMM forward composition

Combinator

\[
\begin{array}{c}
\downarrow I \\
\downarrow J \\
\downarrow K \\

m_1 \\
\; \\
\uparrow J \\
\; \\
\downarrow m_2
\end{array}
\]

is defined by **Kleisli** composition

\[
m_1 ; m_2 = (\psi m_2) \bullet (\phi m_1)
\]

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 \( X \xrightarrow{\eta} FX \xleftarrow{\mu} F^2X \) be a monad in diagram

\[
\begin{align*}
F (F C) &\xleftarrow{f} F B \xrightarrow{g} A \\
F C &\xleftarrow{f} B
\end{align*}
\]

\( f \circ g \) denotes the so-called Kleisli composition of \( F \) -resultic arrows, forming a monoid with \( \eta \) as identity:

\[
\begin{align*}
 f \circ (g \circ h) &= (f \circ g) \circ h \\
 f \circ \eta &= f = \eta \circ f
\end{align*}
\]
Given \( I \xrightarrow{m_1} J \) build \( \phi m_1 : \)

\[
\begin{array}{c}
\mathbb{F} (S \times J) \times Q & \xleftarrow{m_1 \times id} & (S \times I) \times Q & \xleftarrow{xr} & (S \times Q) \times I \\
\downarrow^{\tau_r} & & & & \\
\mathbb{F} ((S \times J) \times Q) & & & & \\
\end{array}
\]

where

- \( xr : (S \times Q) \times I \rightarrow (S \times I) \times Q \) 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 $\psi m_2$:

\[
\begin{align*}
S \times F(Q \times K) & \xleftarrow{id \times m_2} S \times (Q \times J) \xleftarrow{x_l} (S \times J) \times Q \\
F(S \times (Q \times K)) & \xrightarrow{\tau_l} F((S \times Q) \times K)
\end{align*}
\]

where

- $a^\circ$ is the converse of isomorphism $a : (A \times B) \times C \to A \times (B \times C)$
- $x_l$ is a variant of $x_r$
- $\tau_l : (B \times F A) \to F(B \times A)$ is the left strength of $F$. 
Finally build $m_1 ; m_2 = (\psi m_2) \bullet (\phi m_1)$:

\[
\begin{align*}
F \left( F \left( (S \times Q) \times K \right) \right) & \xleftarrow{\mu} \quad F ((S \times J) \times Q) & \xleftarrow{\phi m_1} \quad F ((S \times Q) \times I) \\
\downarrow \mu & & & & \downarrow \phi m_1 \\
F ((S \times Q) \times K) & \xleftarrow{\psi m_2} \quad (S \times J) \times Q & & & & \xleftarrow{m_1 ; m_2}
\end{align*}
\]

This for perfect $F$-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?
Finally build \( m_1 ; m_2 = (\psi m_2) \bullet (\phi m_1) \):

\[
\begin{align*}
\mathbb{F}(\mathbb{F}((S \times Q) \times K)) & \xrightarrow{\mu} \mathbb{F}((S \times J) \times Q) \xrightarrow{\phi m_1} \mathbb{F}((S \times Q) \times I) \\
\mathbb{F}((S \times Q) \times K) & \xrightarrow{\psi m_2} (S \times J) \times Q
\end{align*}
\]

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 $X \xrightarrow{\eta_F} \mathbb{F} X \xleftarrow{\mu_F} \mathbb{F}^2 X$ caters for transitional effects (how the machine evolves)

- monad $X \xrightarrow{\eta_G} \mathbb{G} X \xleftarrow{\mu_G} \mathbb{G}^2 X$ specifies the branching type of the system.
Going relational
Doubly-monadic machines

Typical instance:

\[ G = \mathcal{P} \text{ (powerset) and } F = M = (1+) \text{ (‘maybe’), that is,} \]
\[ m : Q \times I \rightarrow \mathcal{P} (1 + Q \times J) \]

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

\[ R = [m] \iff \langle \forall b, a :: b R a = b \in m \cdot a \rangle \]

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

\[ [m \cdot n] = [m] \cdot [n] \]

where

- \( m \cdot n \) — Kleisli composition of \( \mathbb{P} \)-functions
- \( [m] \cdot [n] \) — **relational** composition

\[ b (R \cdot S) a \iff \langle \exists c :: b R c \land c S a \rangle \]

of the corresponding **binary relations**.
Composing relational $\mathcal{M}$-machines

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

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

($i_1, i_2 = \text{binary sum injections}$).

**Lifting**: in the original definition

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

run **Kleisli** composition relationally:

$$R \bullet S = [i_1, id] \cdot (id + R) \cdot S = [i_1, R] \cdot S = i_1 \cdot i_1^\circ \cdot S \cup R \cdot i_2^\circ \cdot S$$
Composing relational $\mathcal{M}$-machines

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

$$(y = *) \land (* S a) \lor \exists c :: (y R c) \land ((i_2 c) S a)$$

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 $\mathcal{M}$-machines are $\mathcal{M}$-relations and original (deterministic) definition is “reused” in the relational setting:

$$R_1 ; R_2 = (\psi R_2) \bullet (\phi R_1) = [i_1, \psi R_2] \cdot (\phi R_1)$$
Going linear
Probabilistic branching (\(\mathcal{D}\) instead of \(\mathcal{P}\))

Again, instead of working in Set,

\[
\begin{align*}
\mathcal{D} \left( F \; B \right) & \xleftarrow{g} A \\
\mathcal{D} \left( F \; C \right) & \xleftarrow{f} B
\end{align*}
\]

we seek to implement \(F\)-Kleisli-composition in the Kleisli category of \(\mathcal{D}\), that is

\[
\begin{align*}
\left[ f \right] & \bullet \left[ g \right] \\
\mathcal{F} \; B & \xleftarrow{\left[ g \right]} A \\
\mathcal{F} \; C & \xleftarrow{\left[ f \right]} B
\end{align*}
\]

thus “abstracting from” monad \(\mathcal{D}\).

**Question:** Kleisli(\(\mathcal{D}\)) = ??
Probabilistic monadic machines

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

$$
\begin{aligned}
A \rightarrow_{Set} \mathbb{D}B & \cong \ A \rightarrow_{CS} B \\
\iff & \langle \forall b, a :: b M a = (f a) b \rangle
\end{aligned}
$$

such that

$$
M = \left[f\right] \iff \langle \forall b, a :: b M a = (f a) b \rangle
$$

where $A \rightarrow_{CS} B$ 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

\[ f = \text{id}_{0.1 \diamond (\neg)} \]

which corresponds to matrix

\[
\begin{bmatrix}
\text{True} & \text{False} \\
0.1 & 0.9 \\
0.9 & 0.1 \\
\end{bmatrix}
\]

where **probabilistic choice** is immediate on the matrix side,

\[
\begin{bmatrix}
\text{True} & \text{False} \\
0.1 & 0.9 \\
0.9 & 0.1 \\
\end{bmatrix}
\]

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 \((A, B, \ldots)\) and **morphisms** \((M : A \rightarrow B)\) are matrices whose columns have finite support.

- **Composition**:

  \[
  B \xleftarrow{M} A \xrightarrow{N} C
  \]

  that is:

  \[
  b(M \cdot N)c = \left\langle \sum a :: (rMa) \times (aNc) \rightangle
  \]

- **Identity**: the diagonal Boolean matrix \(id : A \rightarrow A\).
Typed linear algebra

Matrix coproducts

\[(A + B) \to C \cong (A \to C) \times (B \to C)\]

where \(A + B\) is disjoint union, cf. universal property

\[X = [M|N] \iff X \cdot i_1 = M \land X \cdot i_2 = N\]

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

\[ M \oplus N = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \]

is an (endo,bi)functor, cf.

\[
(id \oplus id) = id \\
(M \oplus N) \cdot (P \oplus Q) = (M \cdot P) \oplus (N \cdot Q) \\
[M|N] \cdot (P \oplus Q) = [M \cdot P|N \cdot Q]
\]

as in relation algebra — etc, etc.

The Maybe monad in the category is therefore given by

\[ M = (id \oplus \cdot) \]
As we did for relations representing Kleisli(\(\mathbb{P}\)), let us encode \(M\)-Kleisli composition in matrix form:

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

Thus \(M \bullet N = i_1 \cdot i_1^\circ \cdot N + M \cdot i_2^\circ \cdot N\) leading into the pointwise

\[
y (M \bullet N) a = (y = \ast) \times (\ast N a) + \langle \sum b :: (y M b) \times ((i_2 b) N a) \rangle
\]

— compare with the relational version and example (next slide).
Another “Kleisli shift”

Example:

Probabilistic \( \mathbf{M} \)-Kleisli composition \( \mathbf{M} \circ \mathbf{N} \) of matrices
\( \mathbf{N} : \{ a_1, a_2, a_3 \} \rightarrow 1 + \{ c_1, c_2 \} \)
and
\( \mathbf{M} : \{ c_1, c_2 \} \rightarrow 1 + \{ b_1, b_2 \} \).

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

Example: for input \( a_1 \) there is 60% probability of \( \mathbf{M} \circ \mathbf{N} \) failing = either \( \mathbf{N} \) fails (50%) or passes \( c_1 \) to \( \mathbf{M} \) (50%) which fails with 20% probability.
Similarly to relations before, we can think of probabilistic $M$-monadic Mealy machines as CS matrices which communicate (as matrices) as follows

$$N ; M = [i_1|(id \oplus a^\circ) \cdot \tau_l \cdot (id \otimes M) \cdot x_l] \cdot \tau_r \cdot (N \otimes id) \cdot x_r \quad (3)$$

where

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

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

**NB:** Haskell implementation of pMMM composition follows (3).
Kleisli shift
Monad-monad lifting

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

\begin{center}
\textbf{The lifting of monad $F$ by monad $G$ still is a monad in the Kleisli category of $G$.}
\end{center}

Recall:

- $F$ — transition monad
- $G$ — branching monad

Mind their different roles:

\begin{center}
\textbf{Branching monad “hosts” transition monad.}
\end{center}
Monad-monad lifting

In general, given two monads

\[
\begin{align*}
X &\xrightarrow{\eta_G} GX \xleftarrow{\mu_G} G^2 X \quad (\text{the host}) \\
X &\xrightarrow{\eta_F} FX \xleftarrow{\mu_F} F^2 X \quad (\text{the guest})
\end{align*}
\]

in a category \( \mathcal{C} \):

- let \( \mathcal{C}^b \) denote the Kleisli category induced by host \( \mathcal{G} \);
- let \( B \xleftarrow{f^b} A \) be the morphism in \( \mathcal{C}^b \) corresponding to \( \mathcal{G} B \xleftarrow{f} A \) in \( \mathcal{C} \);
- define

\[
f^b \cdot g^b = (f \cdot g)^b = (\mu_G \cdot G f \cdot g)^b
\]
Monad-monad lifting

For any morphism \( B \xleftarrow{f} A \) in \( \mathbf{C} \) define its lifting to \( \mathbf{C}^b \) by

\[
\bar{f} = (\eta_G \cdot f)^b
\]  

(4)

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

\[ \lambda : FG \to GF \]

Lift the guest endofunctor \( F \) from \( \mathbf{C} \) to \( \mathbf{C}^b \) by defining \( \bar{F} \) as follows, for \( \mathbf{G} \ B \xleftarrow{f} A \):

\[
\bar{F}(f^b) = (\lambda \cdot F f)^b
\]

cf. diagram

\[
\mathbf{G} \mathbf{F} B \xleftarrow{\lambda} \mathbf{F} \mathbf{G} B \xleftarrow{F f} \mathbf{F} A
\]
Monad-monad lifting

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

$$\lambda \cdot \overline{F} \eta_G = \eta_G$$
$$\lambda \cdot \overline{F} \mu_G = \mu_G \cdot G \lambda \cdot \lambda$$

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

$$X \xrightarrow{\eta_{\overline{F}}=(\eta_G \cdot \eta_{\overline{F}})^b} \overline{F}X \xleftarrow{\mu_{\overline{F}}=(\eta_G \cdot \mu_{\overline{F}})^b} \overline{F}^2X$$

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

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

\[(\mathcal{F} f^b) \cdot \eta_{\mathcal{F}} = \eta_{\mathcal{F}} \cdot f^b\]
\[(\mathcal{F} f^b) \cdot \mu_{\mathcal{F}} = \mu_{\mathcal{F}} \cdot (\mathcal{F}^2 f^b)\]

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

\[\lambda \cdot \eta_{\mathcal{F}} = \mathcal{G}\eta_{\mathcal{F}}\]
\[\lambda \cdot \mu_{\mathcal{F}} = \mathcal{G}\mu_{\mathcal{F}} \cdot \lambda \cdot \mathcal{F}\lambda\]

that is:

\[\begin{align*}
\mathcal{G}X & \xrightarrow{\eta_{\mathcal{F}}} \mathcal{F}\mathcal{G}X & \xleftarrow{\mu_{\mathcal{F}}} & \mathcal{F}^2(\mathcal{G}X) \\
\mathcal{G}\eta_{\mathcal{F}} & \xrightarrow{\lambda} \mathcal{G}\mathcal{F}X & \xleftarrow{\mu_{\mathcal{F}}} & \mathcal{G}(\mathcal{F}^2X) & \xleftarrow{\lambda} \mathcal{F}\mathcal{G}\mathcal{F}X
\end{align*}\]

(Details in the paper.)
Pairing!
Not yet done!

There is a price to pay for the “hosting” process.

Definition of $[m_1] ; [m_2]$ is strongly monadic.

Question:

---

Do strong monads lift to strong monads?

---

Recall the types of the two strengths:

\[
\begin{align*}
\tau_l &: (B \times \mathbb{F} A) \to \mathbb{F} (B \times A) \\
\tau_r &: (\mathbb{F} A \times B) \to \mathbb{F} (A \times B)
\end{align*}
\]

The basic properties, e.g. $\mathbb{F} lft \cdot \tau_r = lft$ and $\mathbb{F} a^\circ \cdot \tau_r = \tau_r \cdot (\tau_r \times id) \cdot a^\circ$ are preserved by their liftings (e.g. $\overline{\tau_r}$) by construction.
So, what may fail is their **naturality**, e.g.

\[
\overline{\tau}_l \cdot (N \otimes \overline{F} M) = \overline{F} (N \otimes M) \cdot \overline{\tau}_l
\]

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

---

**Naturality is essential to pointfree proofs!**

---

Example: for \( F = M = (1+) \) we have e.g. \( \overline{\tau}_l = (\bar{!} \oplus id) \cdot \overline{dr} \), that is

\[
1 + A \times B \xleftarrow{\bar{!} \oplus id} (1 \times B) + (A \times B) \xleftarrow{\bar{dr}} (1 + A) \times B
\]

dropping the \( \overline{f} \) bars over functions for easier reading.
Naturality which lifts

Is $\bar{1} \oplus id$ natural? We check:

$$(id \oplus N) \cdot (\bar{1} \oplus id) = (\bar{1} \oplus id) \cdot (M \oplus N)$$

$\iff \{ \text{bifunctor} \cdot \oplus \cdot \}$$

$\bar{1} \oplus N = (\bar{1} \cdot M) \oplus N$

$\iff \{ \bar{1} \cdot M = \bar{1} \text{ because } M \text{ is a CS matrix} \}$

true

---

**Note:** matrix $M$ is CS iff $\bar{1} \cdot M = \bar{1}$ holds. (Thus composition is closed over CS-matrices.)
Naturality which does not lift

Is the diagonal function $\delta = \text{id} \triangle \text{id}$ — that is

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

still natural once lifted to matrices?

No! Diagram

$$\begin{array}{ccc}
A \times A & \xleftarrow{\delta} & A \\
\downarrow_{M \otimes M} & & \downarrow_{M} \\
B \times B & \xleftarrow{\delta} & B
\end{array}$$

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

Given probabilistic $f$

\[
\begin{array}{ccc}
\text{f} & \text{a} & \text{b} \\
\text{F} & 0.3 & 1 \\
\text{T} & 0.7 & 0 \\
\end{array}
\]

evaluate $\delta \cdot f$

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

\[
\begin{array}{ccc}
\text{delta} & \text{a} & \text{b} \\
\text{(a,a)} & 1 & 0 \\
\text{(a,b)} & 0 & 0 \\
\text{(b,a)} & 0 & 0 \\
\text{(b,b)} & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{(f x f)} & \text{(a,a)} & \text{(a,b)} & \text{(b,a)} & \text{(b,b)} \\
\text{(F,F)} & 0.09 & 0.3 & 0.3 & 1 & 0.09 & 1 \\
\text{(F,T)} & 0.21 & 0 & 0.7 & 0 & 0.21 & 0 \\
\text{(T,F)} & 0.21 & 0 & 0 & 0 & 0.21 & 0 \\
\text{(T,T)} & 0.49 & 0 & 0 & 0 & 0.49 & 0 \\
\end{array}
\]

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

where $\delta : \mathcal{B} \rightarrow \mathcal{B} \times \mathcal{B}$
This happens because the Kleisli-lifting of \textbf{pairing}

\[(f \triangle g) \cdot x = (f \cdot x, g \cdot x)\]

is a \textbf{weak}-product for column stochastic matrices:

\[X = M \triangle N \Rightarrow \begin{cases} 
    \text{fst} \cdot X = M \\
    \text{snd} \cdot X = N
\end{cases}\]  

ie. \((\Leftarrow)\) is not guaranteed

So \((\text{fst} \cdot X) \triangle (\text{snd} \cdot X)\) differs from \(X\) in general.

In LA, \(M \triangle N\) is known as the \textbf{Khatri-Rao} matrix product.

In RA, \(R \triangle S\) is known as the \textbf{fork} operator.
Probabilistic pairing

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

\[ \text{fst} \cdot (M \triangle N) = M \land \text{snd} \cdot (M \triangle N) = N \]

cf. e.g.

\[
M = \begin{bmatrix}
0.5 & 0.3 & 0 & 0.75 \\
0.5 & 0.7 & 1 & 0.25 \\
\end{bmatrix}
\]

\[
M \triangle N = \begin{bmatrix}
0.15 & 0.12 & 0 & 0 \\
0.35 & 0.06 & 0 & 0.75 \\
0 & 0.12 & 0 & 0 \\
0.15 & 0.28 & 0.1 & 0 \\
0.35 & 0.14 & 0.2 & 0.25 \\
0 & 0.28 & 0.7 & 0 \\
\end{bmatrix}
\]

\[
N = \begin{bmatrix}
0.3 & 0.4 & 0.1 & 0 \\
0.7 & 0.2 & 0.2 & 1 \\
0 & 0.4 & 0.7 & 0 \\
\end{bmatrix}
\]
... but **reconstruction**

\[ X = (\text{fst} \cdot X) \triangle (\text{snd} \cdot X) \]

doesn’t hold in general, cf. e.g.

\[
X : 2 \to 2 \times 3
\begin{bmatrix}
0 & 0.4 \\
0.2 & 0 \\
0.2 & 0.1 \\
0.6 & 0.4 \\
0 & 0 \\
0 & 0.1
\end{bmatrix}
\]

\[
(fst \cdot X) \triangle (snd \cdot X) =
\begin{bmatrix}
0.24 & 0.4 \\
0.08 & 0 \\
0.08 & 0.1 \\
0.36 & 0.4 \\
0.12 & 0 \\
0.12 & 0.1
\end{bmatrix}
\]

\(X\) 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 → Matrix algebra

Usual strategy:

"Keep category (sets), change definition"

Proposed strategy:

"Keep definition, change category"
Possible wherever semantic models are structured around a pair \((F, G)\) of monads:

<table>
<thead>
<tr>
<th>Monad</th>
<th>Effect</th>
<th>Transition</th>
<th>Branching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role</td>
<td>Guest</td>
<td>Lifted</td>
<td>Host</td>
</tr>
<tr>
<td>Strategy</td>
<td>Lifted</td>
<td>“Kleisliﬁed”</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>(G)</th>
<th>Kleisli</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P)</td>
<td>Relation algebra</td>
</tr>
<tr>
<td>(Vec)</td>
<td>Matrix algebra</td>
</tr>
<tr>
<td>(D)</td>
<td>Stochastic matrices</td>
</tr>
<tr>
<td>(Giry)</td>
<td>Stochastic relations</td>
</tr>
</tbody>
</table>

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):

  \[ \text{Rel [the category of relations] possesses more 'quantum features' than the category Set of sets and functions [...]} \]

  \[ \text{The categories FdHilb and Rel moreover admit a categorical matrix calculus.} \]

- **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

\[(Q \times A) \rightarrow \mathbb{P} ((2 + V) \times Q)\]

for verifying a persistent memory manager (in IBM's 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)
References


H. Kerstan and B. König. Coalgebraic trace semantics for probabilistic transition systems based on measure theory. In
Motivation

Context

Going relational

Going linear

Kleisli shift

Pairing!

Closing

References


