Typed linear algebra for weighted (probabilistic) automata

J.N. Oliveira

High-Assurance-Software Lab (HASLab)
INESC TEC & U.Minho, Portugal

CIAA 2012 — 17th Int. Conf. on
Implementation and Application of Automata

Porto, July 17-20, 2012

Motivation

Formal methods are "going quantitative" — "**may it happen**"? people want to know "**how often it will happen**".

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

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

Propagation of software **faults** — can this be predicted (**calculated**) rather than simulated?

Motivation

Recent work: calculating fault **propagation** in **functional programs** (Oliveira, 2012).

Broadening scope: can this be extended to faulty **components**? Does software **architecture** matter in this respect?

Starting point: **coalgebraic** approach to software architecture — "Components as coalgebras" (Barbosa, 2001).

"*Components as coalgebras*" qualitative, not **quantitative**...

Automata as coalgebras

Generic approach to transition systems, described by functions of type

$$Q \to \mathbf{F} Q$$

where Q is a set of states and $\mathbf{F} Q$ captures the future behaviour of the system, according to evolution "pattern" $\mathbf{F}$ (functor).

Examples:

- Mealy machines — $\mathbf{F} Q = \mathbf{B}(Q \times O)^I$
- Moore machines — $\mathbf{F} Q = (\mathbf{B} Q)^I \times O$

for I, O input / output types, and $\mathbf{B}$ a behaviour **monad** — eg. **powerset** ($\mathcal{P}$), **distribution** ($\mathcal{D}$), etc.

Background

Vast literature:

- **Probabilistic program semantics** — eg. (Kozen, 1979)
- **Weighted automata** — eg. (Buchholz, 2008), (Droste and Gastin, 2009)
- **Probabilistic automata** — eg. (Larsen and Skou, 1991)
- **Coalgebraic approaches** — eg. (Sokolova, 2005)

  In particular, a recent paper

  Bonchi et al. (2012) — A coalgebraic perspective on linear weighted automata — Information and Computation, 211:77–105.

  combines coalgebraic reasoning with linear algebra.

Why **linear algebra**?

The function-relation-matrix hierarchy

- **Relations** — are everywhere, eg.

  $y \ likes \ x$

  $y \leq x$

- **Functions** — deterministic and total relations, eg.

  $y = ax + b$

- **Matrices** — quantified relations, cf.

  $y \ M \ x = k$

  further to

  $y \ M \ x = true$

  eg.

  $John \ loves \ Mary = 100$ *(very much!)*

The function-relation-matrix hierarchy

- **Functions** — functional programming, an advanced discipline strongly rooted on mathematics. Typing $f : A \to B$ well accepted.
- **Relations** — ubiquitous (eg. graphs) but still under the atavistic *set of pairs* interpretation. Thus $R \subseteq A \times B$ widespread, compared to $A \xrightarrow{R} B$.
- **Matrices** — key concept in mathematics as a whole, many tools (eg. MATLAB, MATHEMATICA) but still "untyped" — explicit **dimension** checking required.

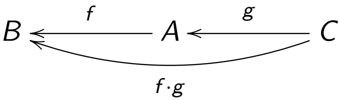
Arrow notation for functions

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

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

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

Compositionality — functions compose with each other:



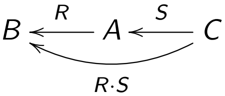
$$b = f(g\ c)$$

Arrow notation for (binary) relations

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

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

Compositionality — relations compose with each other:



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

Example: $uncle = brother \cdot parent$

Older than you probably think

Relational maths finds its roots in the pioneering work

> *On the syllogism: IV, and on the logic of relations*

read by the British mathematician Augustus de Morgan (1806-71), on the 23rd April 1860 to the Cambridge Philosophical Society.

Augustus de Morgan (1806-71)

Binary relations:

> [...] *Let $X..LY$ signify that X is some one of the objects of thought which stand to Y in the* **relation** *L, or is one of the Ls of Y.*

Relational composition:

> [...] *When the predicate is itself the subject of a relation, there may be a* **composition***: thus if $X..L(MY)$, if X be one of the Ls of one of the M s of Y, we may think of X as an 'L of M' of Y, expressed by $X..(LM)Y$, or simply by $X..LMY$.* [...][So] *brother of parent is identical with uncle, by mere definition.*

Relational converse:

> [...] *The* **converse** *relation of L, L^{-1}, is defined as usual: if $X..\ L\ Y$, $Y\ ..\ L^{-1}\ X$: if X be one of the Ls of Y, Y is one of the L^{-1} s of X.*

Later, in the 1940s

Alfred Tarski (1901-83) revives interest in relation algebra. Quoting Givant (2006):

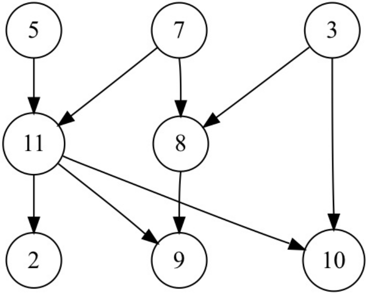
> *In describing this last result in a postcard to Willard van Orman Quine, dated March 27, 1942, Tarski concluded with the following play on a French saying: "Isn't [it] a nice thing 'pour épater les logiciens-bourgeois'?"*

This indicates how amused Tarski was in finding how effective the core of relational algebra is in laying foundations for mathematics as a whole.

From binary relations to matrices

As binary relations are Boolean matrices, eg.

Relation R:



Matrix M:

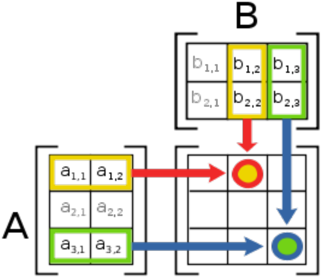
	1	2	3	4	5	6	7	8	9	10	11
1	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	1	0	0	1
10	0	0	1	0	0	0	0	0	0	0	1
11	0	0	0	0	1	0	1	0	0	0	0

why not represent **matrices** as **arrows** too, cf.

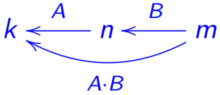
$$11 \xleftarrow{M} 11 \ ?$$

Compositionality — matrix-matrix multiplication

Picture (from the Wikipedia):



Given a **semiring** $(\mathbb{S}; +, \times, 0, 1)$ matrix **composition** $A \cdot B$ obeys to the **typing rule**



such that

$$r(A \cdot B)c \;=\; \langle \sum x \;::\; (rAx) \times (xBc) \rangle \qquad (1)$$

where $\sum$ is the finite iteration over n of the $+$ operation of $\mathbb{S}$.

Typed linear algebra

Notation:

We write rAc *for the* (r, c)*-th cell of matrix* A*, rather than* $A(r, c)$*, for compatibility with relational notation:*

- *you prefer* $4 \leq 5$ *to* $\leq (4, 5)$ *or even* $(4, 5) \in \leq$*, don't you?*

Type checking:

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

$$A + B \quad , \quad A \times B$$

and so on.

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

Typed linear algebra

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

For instance, the **identity matrix**

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

is polymorphic on type n.

(This view will help in equipping tools such as MATLAB and MATHEMATICA with a type system saving the burden of always checking for matrix dimensions.)

Converse

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

> **... or the "passive voice":** *"John* **eats** *the apple"* *converses into "The apple* **is eaten** *by John"* ,
> $(eats)^\circ = (is\ eaten\ by)$

M° is M changed by turning rows into columns and vice-versa.

The following unit, idempotence and contravariance laws hold:

$$id_n \cdot M = M = M \cdot id_m \tag{2}$$

$$(M^\circ)^\circ = M \tag{3}$$

$$(M \cdot N)^\circ = N^\circ \cdot M^\circ \tag{4}$$

Categories of matrices

Equipped with composition (1) and identity (2), matrices form a **category** whose

- **objects** are matrix dimensions and whose
- **morphisms** ($m \xleftarrow{M} n$, $n \xleftarrow{N} k$, etc) are the matrices themselves.

Strictly speaking, there is one such category per matrix cell-level algebra.

Notation $Mat_{\mathbb{S}}$ denotes such a category, parametric on **semiring** $\mathbb{S}$ or any other (richer) algebraic structure, typically a **field** (eg. $\mathbb{R}$).

Categories of matrices

Abelian structure

$$M + 0 = M = 0 + M \tag{5}$$
$$M \cdot 0 = 0 = 0 \cdot M \tag{6}$$

Bilinearity — composition is bilinear relative to $+$:

$$M \cdot (N + P) = M \cdot N + M \cdot C \tag{7}$$
$$(N + P) \cdot M = N \cdot M + P \cdot M \tag{8}$$

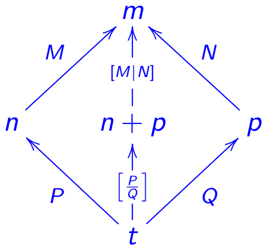
Biproducts — products and coproducts together enabling **block** algebra — the whole story in eg. (MacLane, 1971; MacLane and Birkhoff, 1999) and, more recently, (Macedo, 2012).

(Polymorphic) block combinators

Two ways of putting matrices together to build larger ones:

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

Mind the (polymorphic) types:



(A biproduct)

Blocked linear algebra

Rich set of laws, for instance **divide-and-conquer**,

$$[A|B] \cdot \left[\frac{C}{D}\right] = A \cdot C + B \cdot D \tag{9}$$

two **"fusion"-laws**,

$$C \cdot [A|B] = [C \cdot A|C \cdot B] \tag{10}$$

$$\left[\frac{A}{B}\right] \cdot C = \left[\frac{A \cdot C}{B \cdot C}\right] \tag{11}$$

structural equality,

$$\left[\frac{A}{B}\right] = \left[\frac{C}{D}\right] \Leftrightarrow A = C \wedge B = D \tag{12}$$

— all offered for free from **biproducts**.

Vectors

Vectors are special cases of matrices in which one of the types is 1, for instance

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \qquad \text{and} \qquad w = \begin{bmatrix} w_1 & \ldots & w_n \end{bmatrix}$$

Column vector v is of type $m \longleftarrow 1$ (m rows, one column) and **row** vector w is of type $1 \longleftarrow n$ (one row, n columns).

Our convention is that lowercase letters (eg. v, w) denote vectors and uppercase letters (eg. A, M) denote arbitrary matrices.

Special matrices

The following $(0,1)-$matrices (Boolean) are relevant:

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

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

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

Also note that, on type $1 \longleftarrow 1$:

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

Type generalization

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

In this setting, a **function** $B \xleftarrow{f} A$ will be represented in $Mat_{\mathbb{S}}$ by a $(0{,}1)$-matrix (Boolean) $B \xleftarrow{[\![f]\!]} A$ such that

$$b[\![f]\!]a \quad \triangleq \quad (b =_{\mathbb{S}} f\ a)$$

where, in general, $y =_{\mathbb{S}} x$ is 1 if $y = x$ and 0 otherwise. Thus

$$! \cdot [\![f]\!] \quad = \quad !$$

As $\mathbb{S}$ is always implicit and all diagrams are in $Mat_{\mathbb{S}}$, subscript $\mathbb{S}$ and the parentheses $[\![\ ]\!]$ can be safely dropped.

Weighted automata as $Mat_{\mathbb{S}}$ arrows

Following Droste and Gastin (2009), a **weighted finite automaton** $W = (A, Q; \lambda, \mu, \gamma)$ consists of

- input **alphabet** A
- finite set of **states** Q
- $\lambda, \gamma : Q \to \mathbb{S}$ — **weight** functions for entering and leaving a state
- $\mu : A \to \mathbb{S}^{Q \times Q}$ such that $\mu(a)(p, q)$ is the cost of **transition** $p \xrightarrow{a} q$ (0 if no such transition).

Thus μ can be regarded as a A-indexed family of weighted state transition structures — treated as **square matrices** by Buchholz (2008), Bonchi et al. (2012) and others.

Weighted automata as $Mat_{\mathbb{S}}$ arrows

Bonchi et al. (2012) instantiate $\mathbb{S}$ to a field $\mathbb{K}$ and only consider μ and γ, in a coalgebraic setting, by reshaping μ into the **isomorphic** type $Q \rightarrow (\mathbb{K}^Q)^A$ and putting this together with γ into a **coalgebra** of functor $\mathbf{F}X = \mathbb{K} \times (\mathbb{K}^X)^A$:

$$\langle \gamma, \mu \rangle \quad : \quad Q \rightarrow \mathbb{K} \times (\mathbb{K}^Q_\omega)^A$$

This is treated as a coalgebra in *Set* where the so-called **field valuation** (exponential) functor $\mathbb{K}^-_\omega$ calls for a **vector space**.

Inspired by this "hybrid" approach, ours will save ink in handling everything in $Mat_{\mathbb{S}}$.

Weighted automata as $Mat_{\mathbb{S}}$ arrows

Functions such as $\gamma : Q \rightarrow \mathbb{S}$, which evaluate in $\mathbb{S}$, can be encoded as $Mat_{\mathbb{S}}$ vectors of type $Q \longrightarrow 1$ under the rule

$$1\ \gamma\ q \quad \triangleq \quad \gamma(q) \tag{13}$$

Similarly, the matrix encoding of $\mu : A \rightarrow \mathbb{S}^{Q \times Q}$ can be regarded as either of type $Q \times A \longrightarrow Q$ or $Q \longrightarrow Q \times A$, as these types are isomorphic in $Mat_{\mathbb{S}}$.

We go for the second (coalgebraic) alternative and put μ and γ together into a $Mat_{\mathbb{S}}$ **coalgebra** using the **split** (biproduct) combinator,

$$Q \xrightarrow{W = \left[\frac{\mu}{\gamma}\right]} (Q \times A) + 1$$

Weighted automata as $Mat_{\mathbb{S}}$ arrows

This is a coalgebra of $Mat_{\mathbb{S}}$ endofunctor $\mathbf{F}X = (X \otimes id) \oplus id$, where $\otimes$ is **Kronecker** product and $\oplus$ is **direct sum**, two standard (bi)functors in $Mat_{\mathbb{S}}$.

Absorption

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

and fusion

$$\left[\frac{M}{N}\right] \otimes C = \left[\frac{M \otimes C}{N \otimes C}\right] \tag{15}$$

laws help in calculations. Concerning Kronecker product:

$$(y, x)(M \otimes N)(b, a) = (yMb) \times (xNa) \tag{16}$$

Weighted automata homomorphisms in $Mat_{\mathbb{S}}$

Let us now see how our **typed LA** encoding of **WA** regains the **simplicity** of the original, **qualitative** starting point.

A **homomorphism** *between weighted automata* W *and* W' *is a function* h *making the following* $Mat_{\mathbb{S}}$*-diagram commute,*

$$\begin{array}{ccc} \mathbf{F}Q & \xleftarrow{\ W\ } & Q \\ {\scriptstyle \mathbf{F}h}\downarrow & & \downarrow{\scriptstyle h} \\ \mathbf{F}Q' & \xleftarrow[\ W'\ ]{} & Q' \end{array} \tag{17}$$

for $\mathbf{F}h = (h \otimes id) \oplus id$.

Weighted automata homomorphisms in $Mat_{\mathbb{S}}$

In cross-checking that this indeed is the usual, quantified definition, we will resort to two **rules of thumb**,

$$y(f \cdot N)x = \langle \sum z \ : \ y = f(z) : \ zNx \rangle \qquad (18)$$

$$y(g^{\circ} \cdot N \cdot f)x = (g(y))N(f(x)) \qquad (19)$$

where N is an arbitrary matrix and f, g are functional matrices.

These rules generalize similar equalities in **relation algebra**.

They are expressed in the style of the **Eindhoven** quantifier calculus (Backhouse and Michaelis, 2006), as is

$$\langle \sum x \ : \ p(x) : \ e(x) \rangle = \langle \sum x \ :: \ p(x) \times e(x) \rangle \qquad (20)$$

for Boolean term $p(x)$, that is: $p(x) = 1$ iff $p(x)$ holds, 0 otherwise.

Weighted automata homomorphisms in $Mat_{\mathbb{S}}$

Let us calculate:

$$
\begin{array}{ll}
& (\mathbf{F}h) \cdot W = W' \cdot h \\
\Leftrightarrow & \quad \{ \text{ unfold } \mathbf{F}h \text{ , } W \text{ and } W' \ \} \\
& ((h \otimes id) \oplus id) \cdot \left[\frac{\mu}{\gamma}\right] = \left[\frac{\mu'}{\gamma'}\right] \cdot h \\
\Leftrightarrow & \quad \{ \text{ absorption (14), identity (2) and fusion (11) } \} \\
& \left[\frac{(h \otimes id) \cdot \mu}{\gamma}\right] = \left[\frac{\mu' \cdot h}{\gamma' \cdot h}\right] \\
\Leftrightarrow & \quad \{ \text{ equality (12) } \} \\
& \left\{ \begin{array}{l} (h \otimes id) \cdot \mu = \mu' \cdot h \\ \gamma = \gamma' \cdot h \end{array} \right.
\end{array} \tag{21}
$$

Weighted automata homomorphisms in $\mathit{Mat}_{\mathbb{S}}$

Next we unfold $(h \otimes id) \cdot \mu = \mu' \cdot h$ by extensional equality of matrices of type $Q' \times A \longleftarrow Q$:

$$(q', a)((h \otimes id) \cdot \mu)q = (q', a)(\mu' \cdot h)q$$

$\Leftrightarrow$ { (19) on the rhs, since h is a function }

$$(q', a)((h \otimes id) \cdot \mu)q = (q', a)\mu'(h(q))$$

$\Leftrightarrow$ { (18) on the lhs, since $h \otimes id$ is a function too }

$$\langle \sum (p, b) \ : \ (q', a) = (h \otimes id)(p, b) : \ (p, b)\mu q \rangle = (q', a)\mu'(h(q))$$

$\Leftrightarrow$ { since $(h \otimes id)(p, b) = (h(p), b)$; "one-point" rule over $a = b$ }

$$\langle \sum \ p \ : \ q' = h(p) : \ (p, a)\mu q \rangle = (q', a)\mu'(h(q))$$

Weighted automata homomorphisms in $Mat_{\mathbb{S}}$

Finally, liberally writing $p \xleftarrow{a} q$ for the weight of the corresponding transition:

$$\langle \sum p : q' = h(p) : p \xleftarrow{a} q \rangle \quad = \quad q' \xleftarrow{a} h(q)$$

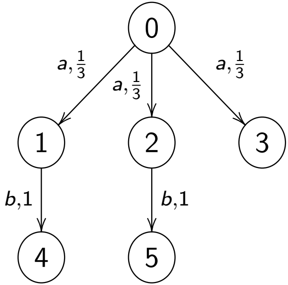
In words:

> *the weight associated to transition* $q' \xleftarrow{a} h(q)$ *in the target automaton accumulates the weights of all transitions* $p \xleftarrow{a} q$ *in the source automaton for all* p *which* h *maps to* q'.

Unfolding $\gamma = \gamma' \cdot h$ will yield the expected $\gamma(q) = \gamma'(h(q))$.

Weighted automata bisimulation in $Mat_{\mathbb{S}}$

We now treat **WA** bisimulation in the same way, illustrated with an example taken from (Buchholz, 2008):



		Q					
Q	A	0	1	2	3	4	5
0	a	0	0	0	0	0	0
0	b	0	0	0	0	0	0
1	a	0.3	0	0	0	0	0
1	b	0	0	0	0	0	0
2	a	0.3	0	0	0	0	0
2	b	0	0	0	0	0	0
3	a	0.3	0	0	0	0	0
3	b	0	0	0	0	0	0
4	a	0	0	0	0	0	0
4	b	0	1	0	0	0	0
5	a	0	0	0	0	0	0
5	b	0	0	1	0	0	0

Matrix μ is type $Q \times A \longleftarrow Q$, for $Q = \{0, ..., 5\}$ and $A = \{a, b\}$.

Probabilistic automata in $Mat_{\mathbb{S}}$

Already an example of a simple, **probabilistic** automaton (Markov chain), instantiating the general definition:

- $\mathbb{S}$ the interval $[0,1]$ in $\mathbb{R}$
- μ is such that $! \cdot \mu$ is a $(0,1)$-vector

($! \cdot M$ adds all columns of M).

Thus $! \cdot \mu \leq !$.

		Q					
Q	A	0	1	2	3	4	5
0	a	0	0	0	0	0	0
0	b	0	0	0	0	0	0
1	a	0.3	0	0	0	0	0
1	b	0	0	0	0	0	0
2	a	0.3	0	0	0	0	0
2	b	0	0	0	0	0	0
3	a	0.3	0	0	0	0	0
3	b	0	0	0	0	0	0
4	a	0	0	0	0	0	0
4	b	0	1	0	0	0	0
5	a	0	0	0	0	0	0
5	b	0	0	1	0	0	0

Wherever $! \cdot \mu = !$ the automaton is **total** and μ is a **column stochastic** matrix, or **probabilistic function** (Oliveira, 2012).

Weighted bisimulations in $Mat_{\mathbb{S}}$

Is equivalence relation

		Q					
		0	1	2	3	4	5
Q	0	1	0	0	0	0	0
	1	0	1	1	0	0	0
	2	0	1	1	0	0	0
	3	0	0	0	1	0	0
	4	0	0	0	0	1	1
	5	0	0	0	0	1	1

a bisimulation? It has four classes which can be represented by a quotient automaton using a suitable homomorphism h.

Weighted bisimulations in $Mat_{\mathbb{S}}$

Candidate **surjective** homomorphism $Q' \xleftarrow{h} Q$:

Q' \ Q	0	1	2	3	4	5
0	1	0	0	0	0	0
I	0	1	1	0	0	0
II	0	0	0	1	0	0
III	0	0	0	0	1	1

Its **kernel** $K = Q \xleftarrow{h^{\circ} \cdot h} Q$ is the given equivalence (kernels of functions are always equivalence relations):

Q \ Q	0	1	2	3	4	5
0	1	0	0	0	0	0
1	0	1	1	0	0	0
2	0	1	1	0	0	0
3	0	0	0	1	0	0
4	0	0	0	0	1	1
5	0	0	0	0	1	1

Weighted bisimulations in $Mat_{\mathbb{S}}$

Building $W' = W/K$ (below we focus on μ, μ' only).

First attempt:

$$W' = W/K = (\mathbf{F}h) \cdot W \cdot h^{\circ}$$

that is

$$\mu' = \mu/K = (h \otimes id) \cdot \mu \cdot h^{\circ}$$

Uups!

		Q'			
Q'	A	0	I	II	III
0	a	0	0	0	0
0	b	0	0	0	0
I	a	0.66	0	0	0
I	b	0	0	0	0
II	a	0.33	0	0	0
II	b	0	0	0	0
III	a	0	0	0	0
III	b	0	2	0	0

Weighted bisimulations in $Mat_{\mathbb{S}}$

It doesn't work because, in $Mat_{\mathbb{S}}$, h° is not a "true" converse of h: the **image** $h \cdot h^{\circ} \neq id$ is a **diagonal** counting "how much non-injective" h is, cf.

Q' \ Q'	0	I	II	III
0	1	0	0	0
I	0	2	0	0
II	0	0	1	0
III	0	0	0	2

However, **surjective** function h has inverses such as, eg. $h^{\bullet} = h^{\circ} \cdot (h \cdot h^{\circ})^{-1}$, obtained by straightforward **inversion** of diagonal $h \cdot h^{\circ}$:

Q \ Q'	0	I	II	III
0	1	0	0	0
1	0	0.5	0	0
2	0	0.5	0	0
3	0	0	1	0
4	0	0	0	0.5
5	0	0	0	0.5

Building $W' = W/K$

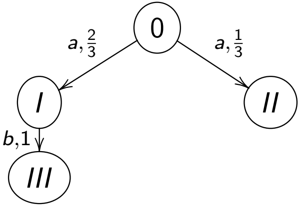
Second attempt:

$$W' = W/K = (\mathbf{F}h) \cdot W \cdot h^{\bullet}$$

that is (aside)

$$\mu' = \mu/K = (h \otimes id) \cdot \mu \cdot h^{\bullet}$$

which leads to automaton



		Q'			
Q'	A	0	I	II	III
0	a	0	0	0	0
0	b	0	0	0	0
I	a	0.66	0	0	0
I	b	0	0	0	0
II	a	0.33	0	0	0
II	b	0	0	0	0
III	a	0	0	0	0
III	b	0	1	0	0

Weighted bisimulations in $Mat_{\mathbb{S}}$

Definition. Equivalence relation K is a **bisimulation** for W iff any surjection h such that $K = h^\circ \cdot h$ is a homomorphism $W/K \xleftarrow{h} W$. That is, any of

$$\mathbf{F}h \cdot W = (W/K) \cdot h$$

$$\Leftrightarrow \qquad \{ \text{ definition of } W/K \ \}$$

$$\mathbf{F}h \cdot W = \mathbf{F}h \cdot W \cdot h^\bullet \cdot h$$

hold. ($h^\bullet \cdot h = K$ for injective h.) Composing both terms with $\mathbf{F}h^\circ$ we get

$$\mathbf{F}K \cdot W \quad = \quad \mathbf{F}K \cdot W \cdot K_\bullet$$

where $K_\bullet = h^\bullet \cdot h$; that is, $\mathbf{F}K \cdot W$ is invariant wrt the "weighted equivalence" $K_\bullet$.

Back to Larsen and Skou (1991)

Noting that $\mathbf{F}K$ is an equivalence relation (as K is so and $\mathbf{F}$ is a functor) and unfolding the invariant $\mathbf{F}K \cdot W$, for μ:

$$
\begin{array}{cl}
 & (q,a)((K \otimes id) \cdot \mu)p \\
= & \qquad \{ \text{ composition rule (1) } \} \\
 & \langle \sum \; q', a' \; :: \; (q,a)(K \otimes id)(q',a') \times ((q',a')\mu(p) \rangle \\
= & \qquad \{ \text{ Kronecker (1) ; term } K \otimes id \text{ is Boolean } \} \\
 & \langle \sum \; q', a' \; :: \; (qKq') \times (a = a') \times ((q',a')\mu(p) \rangle \\
= & \qquad \{ \text{ let } [q]_K \text{ denote the equivalence class of } q \; \} \\
 & \langle \sum \; q' \; : \; q' \in [q]_K : \; q' \xleftarrow{a} p \; \rangle
\end{array}
$$

Back to Larsen and Skou (1991)

In words:

$$\langle \sum q' \ : \ q' \in [q]_K : \quad q' \xleftarrow{a} p \ \rangle$$

is the accumulated cost (probability) of transitions within the same equivalence class, which is invariant for equivalent initial states (Larsen and Skou, 1991).

Equivalence of initial states is captured by "weighting" equivalence K,

$$K_\bullet \quad = \quad h^\circ \cdot (h \cdot h^\circ)^{-1} \cdot h$$

that is,

$$p_1 \ K_\bullet \ p_2 \quad = \quad (h(p_1))(h \cdot h^\circ)^{-1}(h(p_2))$$

Diagonal $(h \cdot h^\circ)^{-1}$ represents the *weight vector [which] is well known in stochastic modeling* (Buchholz, 2008).

Last but not least: behaviour

We finally consider the semantics of **WA** expressed in terms of **weighted languages**.

> *A weighted language over A is a function $\sigma : A^\star \to \mathbb{S}$ assigning a weight to each word in $A^\star$.*

The function $L_W : Q \to \mathbb{S}^{A^\star}$ which associates to each state in Q of W its recognized weighted language (Bonchi et al., 2012) can, as before, be encoded into a $Mat_{\mathbb{S}}$ matrix of type $Q \longrightarrow A^\star$, ie. the **F**-homomorphism (in $Mat_{\mathbb{S}}$)

$$\begin{array}{ccc} Q \times A + 1 & \xleftarrow{\;W\;} & Q \\ {\scriptstyle (L_W \otimes id) \oplus id} \downarrow & & \downarrow {\scriptstyle L_W} \\ A^\star \times A + 1 & \xleftarrow[\;out\;]{} & A^\star \end{array}$$

where

$$\begin{aligned} out &= [rcons|nil]^\circ \\ nil\ \_ &= \epsilon \\ rcons(x, a) &= a : x \end{aligned}$$

Last but not least: behaviour

What does this homomorphism,

$$out \cdot L_W = ((L_W \otimes id) \oplus id) \cdot W$$

mean? We calculate:

$$\begin{array}{ll} & out \cdot L_W = ((L_W \otimes id) \oplus id) \cdot W \\ \Leftrightarrow & \{ \text{ converses } \} \\ & \left[\frac{rcons^\circ}{nil^\circ}\right] \cdot L_W = ((L_W \otimes id) \oplus id) \cdot \left[\frac{\mu}{\gamma}\right] \\ \Leftrightarrow & \{ \text{ fusion (11) and absorption (14) } \} \\ & \left[\frac{rcons^\circ \cdot L_W}{nil^\circ \cdot L_W}\right] = \left[\frac{(L_W \otimes id) \cdot \mu}{\gamma}\right] \\ \Leftrightarrow & \{ \text{ equality (12) } \} \end{array}$$

Last but not least: behaviour

$$\begin{cases} rcons^{\circ} \cdot L_W = (L_W \otimes id) \cdot \mu \\ nil^{\circ} \cdot L_W = \gamma \end{cases}$$

$\Leftrightarrow$ { matrix extensional equality }

$$\begin{cases} (w,a)(rcons^{\circ} \cdot L_W)q = (w,a)((L_W \otimes id) \cdot \mu)q \\ 1(nil^{\circ} \cdot L_W)q = 1\gamma q \end{cases}$$

$\Leftrightarrow$ { thumb rule (19) }

$$\begin{cases} (a : w)\ L_W\ q = (a,w)((L_W \otimes id) \cdot \mu)q \\ \epsilon\ L_W\ q = \gamma(q) \end{cases}$$

Last but not least: behaviour

Finally, as before:

$$\begin{cases} (a:w)\ L_W\ q = \langle \sum\ a', q'\ ::\ (a,w)(L_W \otimes id)(a',q') \times (a',q')\mu\ q\rangle \\ \epsilon\ L_W\ q = \gamma(q) \end{cases}$$

$\Leftrightarrow$ { simplification }

$$\begin{cases} (a:w)\ L_W\ q = \langle \sum\ q'\ ::\ (w\, L_W\, q') \times (\ q' \xleftarrow{a} q\ )\rangle \\ \epsilon\ L_W\ q = \gamma(q) \end{cases}$$

In words:

every state q recognizes the empty language ϵ with weight $\gamma(q)$; and it recognizes sentence $a : w$ for all states which a leads to and which recognize w, accumulating the weights.

Last but not least: behaviour

Another way to look at matrix L_W:

$$out \cdot L_W = ((L_W \otimes id) \oplus id) \cdot W$$

$$\Leftrightarrow \quad \{ \; out \text{ is an isomorphism } \}$$

$$L_W = [rcons|nil] \cdot \left[\frac{(L_W \otimes id) \cdot \mu}{\gamma} \right]$$

$$\Leftrightarrow \quad \{ \; \text{divide and conquer (9)} \; \}$$

$$L_W = rcons \cdot (L_W \otimes id) \cdot \mu + nil \cdot \gamma$$

This shows how L_W is (recursively) filled up, adding to $nil \cdot \gamma$ (the matrix with γ as first row, 0s everywhere else) successive rows as dictated by *rcons*.

Last but not least: behaviour

Using this definition in MATLAB, for the given example automata, we obtain,

for L_W:

	Q					
	0	1	2	3	4	5
[]	0	0	0	1	1	1
[a]	0.3	0	0	0	0	0
[b]	0	1	1	0	0	0
[a,a]	0	0	0	0	0	0
[b,a]	0	0	0	0	0	0
[a,b]	0.7	0	0	0	0	0
[b,b]	0	0	0	0	0	0

for $L_{W/K}$:

	Q'			
	0	I	II	III
[]	0	0	1	1
[a]	0.3	0	0	0
[b]	0	1	0	0
[a,a]	0	0	0	0
[b,a]	0	0	0	0
[a,b]	0.7	0	0	0
[b,b]	0	0	0	0

Summing up

Much still to be done! — but time already to wrap up with the main points:

- Shift from **qualitative** to **quantitative** methods in CS
- Two approaches:
  - Reinvent (extend) original definitions in the **same** category or
  - Stay with original definitions but **change** category (better!)
- $Mat_{\mathbb{S}}$ appears to be a suitable choice for (simple) weighted (probabilistic) automata.

Related work

A lot of related work, the following deserving special reference:

- Trcka (2009) expresses *transition systems in matrix terms* of the form $Q \times Q \to \mathcal{P}A$.
  (Square) matrices of type $Q \to Q$ but not really "quantitative", as the additive operation of $\mathcal{P}A$ is idempotent.

- Bloom et al. (1996) offer the only **matrix-categorial** approach to transition systems I know of.

  Not coalgebraic, however — rather based on **iteration theories**.

  Currently comparing both approaches.

Future work

As in (Oliveira, 2012), rich interplay offered by adjunctions which offer a double perspective — one category is **"for calculating"**, the other **"for programming"** (with the **monad** offered by both):

- Monadic inspiration for more elaborate models coping with both measurable and **unmeasurable** non-determinism.
- Both the powerset functor $\mathcal{P}(-)$ and the distribution functor $\mathcal{D}(-)$ are **monads**.
- Characterize the adjoint categories required by the various forms in which both appear combined in the literature — see eg. the **taxonomy** given by Sokolova (2005).
- Asking for too much?

Linear algebra for software verification

Could not agree more on...

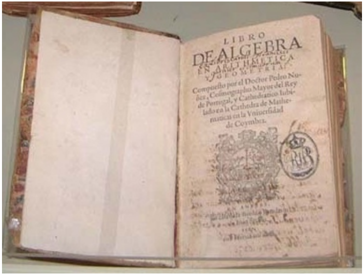
> *"(...) our key idea is to adopt linear algebra as the lingua franca of software verification"*

quoted from

> *LAP: Linear Algebra of bounded resources Programs*

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

Last slide



(...) "De manera, que quien sabe por Algebra, sabe scientificamente".

(...) In this way, who knows by Algebra knows scientifically

[Pedro Nunes (1502-1578) in **Libro de Algebra en Arithmetica y Geometria**, 1567, fol. 270.]

Annex

Index-wise definition of (weighted) bisimulation — recall that, from definition

$$\mathbf{F}K \cdot W \quad = \quad \mathbf{F}K \cdot W \cdot K_{\bullet}$$

we've already expanded, for $\mathbf{F}(X) = X \otimes id$

$$(q,a)(\mathbf{F}K \cdot \mu)p \quad = \quad \langle \sum \; q' \; : \; q' \in [q]_K : \quad q' \xleftarrow{a} p \; \rangle$$

In this annex we turn our attention to

$$(q,a)(\mathbf{F}K \cdot \mu \cdot K_{\bullet})p \quad = \quad \langle \sum \; p' \; :: \; (q,a)(\mathbf{F}K \cdot \mu)p' \times p' K_{\bullet} \; p \rangle$$

The weighted equivalence term is such that

$$p' K_{\bullet} \; p \quad = \quad \frac{1}{|p|_K} p' K \; p$$

where $|p|_K$ is the cardinal of equivalence class $[p]_K$.

Annex

Thus

$$(q,a)(\mathbf{F}K \cdot \mu \cdot K_\bullet)p \quad = \quad \frac{1}{|p|_K}\langle \sum \; p' \;:\; p' \in [p]_K : \; (q,a)(\mathbf{F}K \cdot \mu)p' \rangle$$

whose RHS unfolds into:

$$\frac{1}{|p|_K}\langle \sum \; p' \;:\; p' \in [p]_K : \; \langle \sum \; q'' \;:\; q'' \in [q]_K : \; q'' \xleftarrow{a} p' \rangle\rangle$$

In summary:

$$\langle \sum \; q' \;:\; q' \in [q]_K : \; q' \xleftarrow{a} p \rangle =$$
$$\frac{1}{|p|_K}\langle \sum \; p', q'' \;:\; p' \in [p]_K \wedge q'' \in [q]_K : \; q'' \xleftarrow{a} p' \rangle$$

Annex

The following notation abbreviation will help: for R, S subsets of Q,

$$S \xleftarrow{a} R \quad = \quad \langle \sum \ p, q \ : \ p \in R \wedge q \in S : \ \ q \xleftarrow{a} p \ \rangle$$

Then equivalence K is a bisimulation iff

$$[q]_K \xleftarrow{a} p \quad = \quad \frac{1}{|p|_K} \times (\ [q]_K \xleftarrow{a} [p]_K \ )$$

References

R. Backhouse and D. Michaelis. Exercises in quantifier manipulation. In T. Uustalu, editor, *MPC'06*, volume 4014 of *LNCS*, pages 70–81. Springer, 2006.

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

S.L. Bloom, N. Sabadini, and R.F.C. Walters. Matrices, machines and behaviors. *Applied Categorical Structures*, 4(4):343–360, 1996.

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

P. Buchholz. Bisimulation relations for weighted automata. *Theoretical Computer Science*, 393(1-3):109–123, 2008. ISSN 0304-3975. doi: 10.1016/j.tcs.2007.11.018.

M. Droste and P. Gastin. Weighted automata and weighted logics. In W. Kuich, H. Vogler, and M. Droste, editors, *Handbook of Weighted Automata*, EATCS Monographs in Theoretical Computer Science, chapter 5, pages 175–211. Springer, 2009.

S. Givant. The calculus of relations as a foundation for mathematics. *J. Autom. Reasoning*, 37(4):277–322, 2006. ISSN 0168-7433. doi: http://dx.doi.org/10.1007/s10817-006-9062-x.

Dexter Kozen. Semantics of probabilistic programs. In *FOCS*, pages 101–114, 1979.

K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

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

S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, New-York, 1971.

S. MacLane and G. Birkhoff. *Algebra*. AMS Chelsea, 1999.

José N. Oliveira. Towards a linear algebra of programming. *Formal Asp. Comput.*, 24(4-6):433–458, 2012.

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

A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Ph.D. dissertation, Tech. Univ. Eindhoven, Eindhoven, The Netherlands, 2005.

N. Trcka. Strong, weak and branching bisimulation for transition systems and Markov reward chains: A unifying matrix approach. In S. Andova and *et al*, editors, *Proceedings First Workshop on Quantitative Formal Methods: Theory and Applications*, volume 13 of *EPTCS*, pages 55–65, 2009.