Weighted automata as coalgebras in categories of matrices

J.N. Oliveira

Ref. [Ol13] — 2013

J.N. Oliveira. Weighted automata as coalgebras in categories of matrices. *International Journal of Foundations of Computer Science*, 2013. Accepted for publication

International Journal of Foundations of Computer Science

Weighted automata as coalgebras in categories of matrices

José N. Oliveira
High Assurance Software Laboratory
INESC TEC and University of Minho
Braga, Portugal
jno@di.uminho.pt
http://www.di.uminho.pt/~jno

Received (Day Month Year)
Accepted (Day Month Year)
Communicated by (xxxxxxxxxx)

The evolution from non-deterministic to weighted automata represents a shift from *qualitative* to *quantitative* methods in computer science. The trend calls for a language able to reconcile quantitative reasoning with formal logic and set theory, which have for so many years supported qualitative reasoning. Such a *lingua franca* should be typed, polymorphic, diagrammatic, calculational and easy to blend with conventional notation.

This paper puts forward *typed linear algebra* as a candidate notation for such a unifying role. This notation, which emerges from regarding matrices as morphisms of suitable categories, is put at work in describing weighted automata as coalgebras in such categories.

Some attention is paid to the interface between the index-free (categorial) language of matrix algebra and the corresponding index-wise, set-theoretic notation.

Keywords: Weighted automata; linear algebra; categories of matrices.

1. Introduction

There is a trend towards *quantitative methods* in computing. Further to predicting that something *may happen*, going quantitative should allow one to anticipate *how often or costly it will happen*. Or, looking from the negative side of things, if something bad can take place one wishes to know how likely it is.

As happened with other sciences in the past (eg. physics), computer science is in some sense becoming *quantitative* or *probabilistic*. Take *reliability* as an example. From a qualitative perspective, a software system is *reliable* if it can *successfully carry out its own task as specified* [8]. But our italicized text is a inexact quotation of [8], the exact one being: *reliability [is] defined as a probabilistic measure of the system ability to successfully carry out its own task as specified*.

From a functional perspective, this means moving from specifications (input/output relations) and implementations (functions) to something which lives in between, for instance *probabilistic functions* expressing the propensity, or likelihood of multiple, possibly erroneous outputs [19].

Quantitative software reliability analysis is more complex in practice because, as is well-known, software systems are nowadays built component-wise. Cortellessa and Grassi [8] quantify component-to-component error propagation in terms of a *matrix* whose entry (i, j) gives the probability of component i transferring control to component j — a kind of probabilistic *call-graph*.

Component-oriented design has been successfully formalized under the *components as coalgebras* motto (see eg. [4]), which builds on top of extensive work on automata using coalgebra theory [20]. This theory can be regarded as a generic approach to transition systems, described by functions of type

$$Q \to \mathsf{F}Q$$

where Q is a set of states and $\mathsf{F}Q$ captures the future behaviour of the system according to evolution "pattern" F which is, technically, a functor. Mealy machines, for instance, are captured by $\mathsf{F}Q = \mathsf{B}(Q \times O)^I$ for I, O input/output types, while Moore machines are captured by $\mathsf{F}Q = (\mathsf{B}Q)^I \times O$. In both definitions, B is a behaviour *monad* — eg. the *powerset* ($\mathcal{P}$) monad capturing non-determinism, the *distribution* ($\mathcal{D}$) monad capturing probabilistic behaviour, and so on.

Coalgebras have followed the trend and become quantitative too [24, 23]. Research in this area builds upon foundational work by Larsen and Skou [11] on probabilistic bisimulation. Broadening scope, recent work by Bonchi et al. [6] gives a coalgebraic perspective of so-called *linear weighted automata*. There is, however, a price to pay for such weighted approaches, as functors now need to handle quantities explicitly while states become vectors and coalgebras become linear maps.

In the current paper we wish to obtain the same quantitative effect while keeping the simplicity of the original (qualitative) coalgebra approach. The idea is to keep weighting and quantification *implicit* rather than explicit, the trick being a change of category: instead of the category of sets where traditional coalgebra theory finds its room, we change to suitable categories of matrices tuned to the specific weighting mechanism. That is, we replace set-theoretical operations and weight calculations by matrix operations which hide such calculations. This entails a *change of notation*, as linear algebra rather than set notation becomes the *lingua franca*. For this shift to be effective, special care has to be taken with notation issues, as shown in the current paper. The outcome is, we believe, worthwhile and reassuring.

The remainder of this paper is structured as follows: Section 2 introduces weighted automata (WA) and sections 3 and 4 introduce typed linear algebra. Sections 5 and 6 express WA as comorphisms in categories of matrices. This leads to a coalgebraic treatment of weighted bisimulation and weighted languages in sections 7 and 8. The remaining sections conclude and address related and future work.

2. Weighted automata

Weighted automata [7, 9, 6] are a generalisation of finite state, non-deterministic automata where each state transition, in addition to some input, involves a quantity

indicative of the *weight* (typically expressing cost or probability) of its execution. The minimal structure for managing weights is a *semiring* $(\mathbb{S}; +, \times, 0, 1)$ where $(\mathbb{S}; +, 0)$ is a commutative monoid, $(\mathbb{S}; \times, 1)$ is a monoid, multiplication distributes over addition and 0 annihilates multiplication $(0 \times s = s \times 0 = 0)$.

Following Droste and Gastin [9], a weighted finite automaton $W = (A, Q; \lambda, \mu, \gamma)$ consists of an input alphabet A, a finite set of states Q and three functions: $\lambda, \gamma : Q \to \mathbb{S}$ are weight functions for entering and leaving a state, respectively, and $\mu : A \to \mathbb{S}^{Q \times Q}$ is such that $\mu(a)(p, q)$ indicates the cost of transition $p \xrightarrow{a} q$. Cost 0 means that there is no transition from p to q labelled a.

For $\mathbb{S}$ the Boolean algebra $\mathbb{B}$ of truth values, weighted automaton W becomes a (non-deterministic) labelled transition system (LTS) or non-deterministic finite-state automaton (FSA): $\mu(a) \in \mathbb{B}^{Q \times Q}$ is the state-transition relation associated to input a, λ is the set of initial states and γ the set of terminal states. For $\mathbb{S}$ the interval $[0, 1]$ of the real numbers $\mathbb{R}$, W can be regarded as a *probabilistic automaton* under certain conditions [a]. Bonchi et al. [6] only consider μ and the output function γ. Their coalgebraic model twists the type of μ into $Q \to (\mathbb{S}^Q)^A$ and then amalgamates γ and μ into a coalgebra of functor $\mathsf{F}X = \mathbb{S} \times (\mathbb{S}^X)^A$.

Clearly, for each $a \in A$, $\mu(a) \in \mathbb{S}^{Q \times Q}$ can be regarded as a Q-indexed *matrix* expressing the cost of each state transition in which input a participates. In the same way, λ and γ can be regarded as Q-indexed vectors. It is therefore no wonder that the work on weighted automata often resorts to matrix terminology and operations such as matrix-matrix and matrix-vector multiplications. However, linear algebra (LA) is seldom assumed explicitly as the *central* notation and calculus, as LA reasoning takes place episodically, where convenient, conventional set theory doing the main job. The main advantage of LA — the conciseness of blocked, index-free notation and its powerful algebra — is thus partly lost. There are, however, approaches in which LA is the main notational device, see eg. references [7, 25] and, in particular, [5]. But in all these works LA notation is *untyped* and therefore hard to combine with that of the relations, predicates and functions which are around.

3. Typed versus untyped mathematics.

What does *(un)typed* mean in the previous sentence? It is a commonplace in mathematics to regard functions as special cases of relations (the deterministic, total ones) and relations as special cases of matrices (the Boolean ones, provided addition is trimmed to 1). Yet the three classes of object are treated in disparate ways, with incompatible (if not contradictory) notation.

For instance, one writes $y = f(x)$ to define a function and $(x, y) \in Graph(f)$ — note how x and y swap position — to express the input/output pairs of the graph of function f, which is a relation. As far as typing is concerned, most people accept notation $f : A \to B$ for defining the signature of a function but only reluctantly

[a]For a comprehensive analysis and taxonomy of probabilistic systems see eg. [24].

will accept the same notation $R : A \to B$ to define the *type* of relation R, writing $R \subseteq A \times B$ instead. As far as matrices are concerned, writing $M : m \to n$ to declare the type of a matrix with m columns and n rows will look surprising — textbooks simply tell that M is of order $m \times n$ (or is it $n \times m$?), with loose typing rules. As for type checking, results are stated as *"valid only for matrices of the same order"* [1] and the like. Polymorphic functions are well-accepted. But telling that the identity matrix is as polymorphic as the identity function will sound odd to many people.

Relational mathematics [21] is a step forward towards conceptual unification between relations and matrices. But it is first and foremost *category theory* [16] which provides for successful unification, by regarding functions, relations and matrices as morphisms (arrows) of suitable categories. The category of functions is well known, that of relations less known and those of matrices by and large ignored.

This paper shows how weighted automata can be handled in the *typed* LA which emerges from regarding matrices as morphisms (rather than objects) of suitable categories, as pioneered by MacLane [16] and MacLane and Birkhoff [17]. This is part of a research line which started in [12] and whose aim is to provide evidence of the usefulness of changing notation (and reasoning style) and adopting *typed* LA as the lingua franca of quantitative methods in computer science.

4. Typed linear algebra

Computer scientists tend to regard matrices as rectangular shaped data structures implemented as bidimensional arrays, lists of lists and the like. Mathematicians tend to regard them as linear transforms, i.e. vector-to-vector operations. Yet matrices are abstract entities independent of either such views: they can be regarded as arrows of particular categories, whereby they become *typed*.

By studying categories of matrices [16], the authors of [12] have identified type-rich constructors related to standard matrix operations. Backhouse [3] regards matrices as a way of compacting sets of equations into single equations which *is a tremendous improvement in concision that does not incur any loss of precision!* Reference [12] furthermore shows how the general concept of a *biproduct* [17] promotes individual values to blocks and value-level operations to block-level operations, in fact the great conceptual advantage offered by matrix notation and LA.

Matrices as arrows. A matrix M with n rows and m columns is a function which tells the value occupying each cell (r, c), for $1 \leq r \leq n$ and $1 \leq c \leq m$. As the type of such cell-values varies, the minimal algebraic structure of semirings is required for matrix operations to make sense. Standard linear algebra operates over the richer structure of a *field* (further offering additive and multiplicative inverses) and the field of real numbers ($\mathbb{R}$) is often taken by default.

Interestingly, what is meant by the *type* of a matrix in the sequel does not bear a direct relationship to such algebraic structures: it rather provides (as in programming) a way of interfacing matrices with each other. The type of a matrix

M with m columns and n rows will be denoted by the arrow $m \longrightarrow n$ between the number of columns and the number of rows. By writing $m \xrightarrow{M} n$ (or the equivalent $n \xleftarrow{M} m$) one declares matrix M and its type.

The most interesting matrix combinator is *composition*, commonly referred to as *matrix multiplication*. Denoting the (r, c)-th cell of a given matrix M by rMc [b], the (r, c)-th cell of composite matrix $M \cdot N$ is given by

$$r(M \cdot N)c = \langle \sum x :: (rMx) \times (xNc) \rangle \tag{1}$$

where $\times$ is the cell-level semiring multiplicative operation and $\sum$ is the finite iteration of its additive operation. What is x in (1) and what is its range? This is easy to answer by inspecting the types of both M and N:

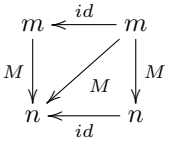
$$n \xleftarrow{M} m \xleftarrow{N} k \qquad (M \cdot N : n \leftarrow k) \tag{2}$$

Thus $1 \leq x \leq m$ and matrix multiplication can be abstracted by arrow composition.

For every n there is a matrix of type $n \longleftarrow n$ which is the unit of composition. This is nothing but the *identity matrix* of size n, indistinguishably denoted by $n \xleftarrow{id_n} n$ or $n \xleftarrow{1} n$. This is the diagonal of size n, that is [c], $r(id)c \triangleq r = c$ under the $\{0, 1\}$ encoding of the Booleans. Therefore, $id_n \cdot M = M = M \cdot id_m$ holds, where subscripts such as m and n can be omitted wherever the underlying type diagrams are assumed (as aside).

$$id_n = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Equipped with composition (2) and identity, matrices form a *category* whose *objects* are matrix dimensions and whose *morphisms* ($m \xleftarrow{M} n$ etc) are the matrices themselves [16, 17]. Strictly speaking, there is one such category per matrix cell-level algebra. Notation $Mat_{\mathbb{S}}$ will be used to denote such a category, parametric on semiring $\mathbb{S}$ or any other (richer) algebraic structure.



Vectors are special cases of matrices in which one of the dimensions is 1, for instance row vector $w = \begin{pmatrix} w_1 \ldots w_n \end{pmatrix}$ of type $1 \longleftarrow n$ (one row, n columns). Our convention is that lowercase letters (eg. v, w) denote vectors and uppercase letters (eg. M, N) denote arbitrary matrices.

Further to composition, another kernel operation of linear algebra is *transposition*, whereby a given matrix changes shape by turning its rows into columns and vice-versa. Given matrix $n \xleftarrow{M} m$, notation $m \xleftarrow{M^\circ} n$ will denote its trans-

[b] Rather than the more conventional $M(r, c)$ — we will explain later why we propose such an infix notation.

[c] Notation $x \triangleq y$ means $x = y$ by definition.

pose, or *converse*. The following idempotence and contravariance laws hold:

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

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

Bilinearity. Given two matrices of the same type $n \xleftarrow{M,N} m$ it makes sense to add them up index-wise, leading to matrix $M + N$ where symbol $+$ promotes the underlying semiring additive operator to matrix-level. Likewise, additive unit cell value 0 is promoted to matrix 0 wholly filled with 0s, the unit of matrix addition, $M+0 = M = 0+M$, and zero of matrix composition: $M \cdot 0 = 0 = 0 \cdot M$. Furthermore, composition is bilinear relative to $+$, that is, $M \cdot (N + P) = M \cdot N + M \cdot C$ and $(N + P) \cdot M = N \cdot M + P \cdot M$ hold.

In the same way $M + N$ denotes the promotion of addition of matrix cells to matrix addition, the same promotion can take place with respect to the whole semiring algebra. For instance, cell value multiplication leads to matrix multiplication, denoted $M \times N$ or simply MN (for M and N of the same type), also known as the *Hadamard product*, which is commutative, associative and distributive over addition (ie. bilinear). Clearly, $M \times \top = \top \times M = M$ where matrix $\top$ has the same type as M and is wholly filled with 1s.

Type generalization. Matrix types (the end points of arrows) can be generalized to arbitrary, denumerable sets since addition in $\mathbb{S}$ is commutative, that is, the summation of (1) can be evaluated in arbitrary order.

In fact, and as is standard in relational mathematics [21], objects in categories of matrices can be generalized from numeric dimensions (n, $m \in \mathbb{N}_0$) to arbitrary denumerable types (A, B), taking disjoint union $A+B$ for $m+n$, Cartesian product $A \times B$ for mn, unit (singleton) type 1 for number 1, the empty set $\emptyset$ for 0, etc. Conversely, dimension n corresponds to the type made of the initial segment of the natural numbers up to n. Our convention is that lowercase letters (eg. n, m) denote the traditional dimension types (natural numbers), letting uppercase letters denote arbitrary other types.

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

Following [6], we will consider a simpler notion of weighted automaton $W = (Q, A; \mu, \gamma)$ which deals without the input weight function λ. This facilitates the comparison between the coalgebraic approach of [6] and our own and helps in staying with the binary matrix block combinators of [12], to be presented shortly. For this purpose, we assign the type $Q \longrightarrow 1$ to output function γ, which is therefore regarded as a row vector in $Mat_{\mathbb{S}}$. Concerning μ, it can either be regarded as a matrix of type $Q \times A \longrightarrow Q$ or of type $Q \longrightarrow Q \times A$, as these types are isomorphic in $Mat_{\mathbb{S}}$ [d]. We prefer the second (coalgebraic) alternative and therefore regard

[d]This follows from a self-adjunction in $Mat_{\mathbb{S}}$ which is studied in detail in [15]. The isomorphism reshapes matrices by reducing the number of columns by the same factor the number of rows

the following diagram as representation of weighted automaton $W = (Q, A; \mu, \gamma)$:

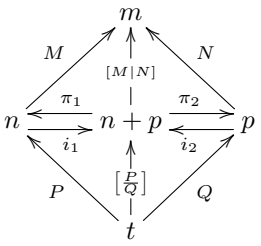
$$Q \times A \xleftarrow{\mu} Q \xrightarrow{\gamma} 1 \tag{5}$$

Clearly, both μ and γ can be packaged into a single coalgebra (matrix) of type $(Q \times A) + 1 \xleftarrow{W} Q$ and made of two blocks

$$W = \left[\frac{\mu}{\gamma}\right] \tag{6}$$

provided we explain what the meaning of combinator $\left[\text{-}\right]$ is. This leads into matrix block notation and its algebra.

Block notation. Matrices can be built block-wise. Two basic combinators are available for building matrices out of other matrices, say M and N: $[M|N]$, meaning M and N side by side (read $[M|N]$ as "M *junc* N"); or $\left[\frac{M}{N}\right]$, meaning M on top of N (read $\left[\frac{M}{N}\right]$ as "M *split* N"). That is, the matrices are stacked either vertically ($\left[\frac{M}{N}\right]$) or horizontally ($[M|N]$). Dimensions should agree, as shown in the diagram aside, where m, n, p and t are types.



The special matrices i_1, i_2, π_1 and π_2 are fragments of the identity matrix as given by the so-called *reflexion laws*, $[i_1|i_2] = id$ and $\left[\frac{\pi_1}{\pi_2}\right] = id$. In brief, *junc* and *split* form a so-called *biproduct* [16], whereby $[M|N] = M \cdot \pi_1 + N \cdot \pi_2$ and $\left[\frac{P}{Q}\right] = i_1 \cdot P + i_2 \cdot Q$ hold. The details of this, however, can be skipped for the purposes of this paper, sufficing to be aware of the rich algebra of such combinators, of which we single out *divide-and-conquer*,

$$[A|B] \cdot \left[\frac{C}{D}\right] = A \cdot C + B \cdot D \tag{7}$$

two "fusion"-laws,

$$R \cdot [M|N] = [R \cdot M|R \cdot N] \tag{8}$$

$$\left[\frac{M}{N}\right] \cdot R = \left[\frac{M \cdot R}{N \cdot R}\right] \tag{9}$$

two structural equality laws,

$$[A|B] = [C|D] \quad \equiv \quad A = C \wedge B = D \tag{10}$$

$$\left[\frac{A}{B}\right] = \left[\frac{C}{D}\right] \quad \equiv \quad A = C \wedge B = D \tag{11}$$

and two absorption laws,

$$[A|B] \cdot (C \oplus D) = [A \cdot C|B \cdot D] \tag{12}$$

$$(C \oplus D) \cdot \left[\frac{A}{B}\right] = \left[\frac{C \cdot A}{D \cdot B}\right] \tag{13}$$

increases, keeping the "rectangular area" and its information intact.

where $\oplus$ is the so-called *direct sum* bifunctor, $M \oplus N = [i_1 \cdot M | i_2 \cdot N]$, of type $k + j \xleftarrow{} n + m$ for $k \xleftarrow{M} n$ and $j \xleftarrow{} m$. Mind the types: the laws above are only valid for matrices which typecheck, types being obtained by unification as explained in [12].

Back to (5) and (6) above, we are in position to regard weighted automaton $Q \times A + 1 \xleftarrow{W} Q$ as a coalgebra for $Mat_{\mathbb{S}}$ endofunctor $\mathsf{F}X = (X \otimes id) \oplus id$ which, further to direct sum, resorts to the so-called *Kronecker* product $\otimes$, another $Mat_{\mathbb{S}}$ bifunctor: given matrices $n \xrightarrow{M} k$ and $m \xrightarrow{N} j$, $M \otimes N$ is of type $n \times m \longrightarrow k \times j$ and obeys fusion laws $[M|N] \otimes C = [M \otimes C | N \otimes C]$ and $\left[\frac{M}{N}\right] \otimes C = \left[\frac{M \otimes C}{N \otimes C}\right]$. These laws capture the meaning of Kronecker product blockwise. Index-wise, one has:

$$(y, x)(M \otimes N)(b, a) = (yMb) \times (xNa) \tag{14}$$

6. Weighted automata comorphisms

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

$$\begin{array}{ccc} \mathsf{F}Q & \xleftarrow{W} & Q \\ {\scriptstyle \mathsf{F}h}\downarrow & & \downarrow{\scriptstyle h} \\ \mathsf{F}Q' & \xleftarrow[W']{} & Q' \end{array} \tag{15}$$

for $\mathsf{F}X = (X \otimes id) \oplus id$. We say that h is an F-*coalgebra homomorphism* or, abbreviating text, an F-*comorphism*.

The reader may wonder about how does h (a function) fit into a diagram of matrices (15). The explanation is easy: every function $A \xrightarrow{f} B$ can be represented in $Mat_{\mathbb{S}}$ by a matrix $[\![f]\!]$ of the same type defined by $b[\![f]\!]a \triangleq (b =_{\mathbb{S}} f\ a)$ where, in general, $y =_{\mathbb{S}} x$ is the unit 1 of $\mathbb{S}$ if $y = x$ and 0 otherwise. Thus $[\![f]\!]$ is the matrix which represents the graph of f: there is a 1 in every entry of $[\![f]\!]$ addressed by $(f(a), a)$ and 0s everywhere else. As $\mathbb{S}$ is always implicit and all diagrams are drawn in $Mat_{\mathbb{S}}$ unless otherwise specified, subscript $\mathbb{S}$ in $=_{\mathbb{S}}$ and the parentheses in $[\![f]\!]$ can be safely dropped.

Below we show how diagram (15) unfolds into the usual definition of weighted automata homomorphism [6], which is termed *functional* simulation in [7]. For this we will rely on typed, blocked linear algebra:

$$\begin{array}{ll} & (\mathsf{F}h) \cdot W \ = \ W' \cdot h \\ \equiv & \quad \{ \text{ unfold } \mathsf{F}h \text{ ; } W \text{ and } W' \text{ are splits defined by (6) } \} \\ & ((h \otimes id) \oplus id) \cdot \left[\frac{\mu}{\gamma}\right] \ = \ \left[\frac{\mu'}{\gamma'}\right] \cdot h \end{array}$$

$$
\begin{array}{ll}
\equiv & \{ \text{ absorption (13), identity and fusion (9) } \} \\
& \left[\dfrac{(h \otimes id) \cdot \mu}{\gamma}\right] \;=\; \left[\dfrac{\mu' \cdot h}{\gamma' \cdot h}\right] \\
\equiv & \{ \text{ equality (11) } \} \\
& \begin{cases} (h \otimes id) \cdot \mu = \mu' \cdot h \\ \gamma = \gamma' \cdot h \end{cases}
\end{array} \tag{16}
$$

Converting the equalities of (16) into index-wise formulas for cross-checking with other sources is greatly helped by the following rules interfacing index-free and index-wise matrix notation, where N is an arbitrary matrix and f, g are functional matrices:

$$y(f \cdot N)x \;=\; \langle \sum z : y = f(z) : zNx \rangle \tag{17}$$

$$y(g^{\circ} \cdot N \cdot f)x \;=\; (g(y))N(f(x)) \tag{18}$$

These rules are expressed in the style of the Eindhoven quantifier calculus [2]. Their calculation [18] provides evidence of the safe mix among matrix, predicate and function notation in typed LA. We start by unfolding the first equality of (16):

$$
\begin{array}{ll}
& (h \otimes id) \cdot \mu = \mu' \cdot h \\
\equiv & \{ \text{ index-wise equality on matrices of type } \; Q' \times A \xleftarrow{\quad} Q \; \} \\
& (q', a)((h \otimes id) \cdot \mu)q = (q', a)(\mu' \cdot h)q \\
\equiv & \{ \text{ (18) on the right hand side, for } g, N, f := id, \mu', h \; \} \\
& (q', a)((h \otimes id) \cdot \mu)q = (q', a)\mu'(h(q)) \\
\equiv & \{ \text{ (17) on the left hand side, for } f, N := h \otimes id, \mu \; \} \\
& \langle \sum (p, b) : (q', a) = (h \otimes id)(p, b) : (p, b)\mu q \rangle = (q', a)\mu'(h(q)) \\
\equiv & \{ \text{ since } (h \otimes id)(p, b) = (h(p), b); \text{ one-point rule [2] over } a = b \; \} \\
& \langle \sum p : q' = h(p) : (p, a)\mu q \rangle = (q', a)\mu'(h(q)) \\
\equiv & \{ \text{ liberally writing } \; p \xleftarrow{a} q \; \text{ for the weight of the corresponding transition } \; \} \\
& \langle \sum p : q' = h(p) : \; p \xleftarrow{a} q \; \rangle = \; q' \xleftarrow{a} h(q)
\end{array}
$$

In words: the weight associated to transition $q' \xleftarrow{a} h(q)$ in the target automaton is the accumulation of the weights of all transitions $p \xleftarrow{a} q$ in the source automaton for all p which h maps to q'.

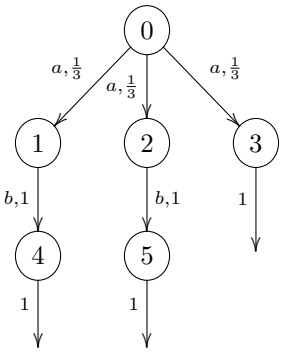
		Q					
Q	A	0	1	2	3	4	5
0	a	0	0	0	0	0	0
0	b	0	0	0	0	0	0
1	a	0.3	0	0	0	0	0
1	b	0	0	0	0	0	0
2	a	0.3	0	0	0	0	0
2	b	0	0	0	0	0	0
3	a	0.3	0	0	0	0	0
3	b	0	0	0	0	0	0
4	a	0	0	0	0	0	0
4	b	0	1	0	0	0	0
5	a	0	0	0	0	0	0
5	b	0	0	1	0	0	0

Fig. 1. Transition table μ of weighted automaton of type $Q \times A \leftarrow Q$. Columns are indexed by input type Q and rows by Cartesian product type $Q \times A$.

Unfolding the other matrix equality in (16) is simpler: as γ, γ' are vectors, we get, for all $q \in Q$, $1\gamma q = 1(\gamma' \cdot h)q$, since there is only one row. By (18) this becomes $1\gamma q = 1\gamma'(h(q))$, that is $\gamma(q) = \gamma'(h(q))$ once γ, γ' are regarded back as functions.

Summing up, both calculations show that weighted automata comorphisms defined in a category of matrices coincide with those defined by Bonchi et al. [6] in the category of sets. As in the standard (set-theoretic) setting, id is an F-comorphism and F-comorphisms compose; thus coalgebraic weighted automata are the objects of a category whose arrows are F-comorphisms in $Mat_{\mathbb{S}}$.

7. Weighted bisimulation



Having laid a basis for a coalgebraic, typed LA approach to weighted automata, we can experiment with further constructions. As a follow up example, this section aims at a purely quantitative definition of WA *bisimulation*, taking the WA aside from [7] as illustrative example. The matrix μ which describes this automaton, of type $Q \times A \xleftarrow{} Q$ for $Q = \{0, ..., 5\}$ and $A = \{a, b\}$, is shown in Figure 1 . Transitions leading nowhere and labelled with 1s correspond to $\gamma = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}$, a vector of type $1 \xleftarrow{} Q$. This is an example of a simple, probabilistic automaton in $Mat_{\mathbb{S}}$ (Markov chain), instantiating the general definition: $\mathbb{S}$ is the interval $[0, 1]$ in $\mathbb{R}$, and μ is such that $! \cdot \mu$ is a $(0, 1)$-vector, where row vector $!$ is wholly filled with 1s. (Thus $! \cdot M$ adds all columns of M) [19].

Is the equivalence relation depicted in Figure 2 (a) a bisimulation? It has four classes which can be represented by a quotient automaton using a suitable *surjective* homomorphism $Q' \xleftarrow{h} Q$ depicted in Figure 2(b) for $Q' = \{0, \text{I}, \text{II}, \text{III}\}$. It can

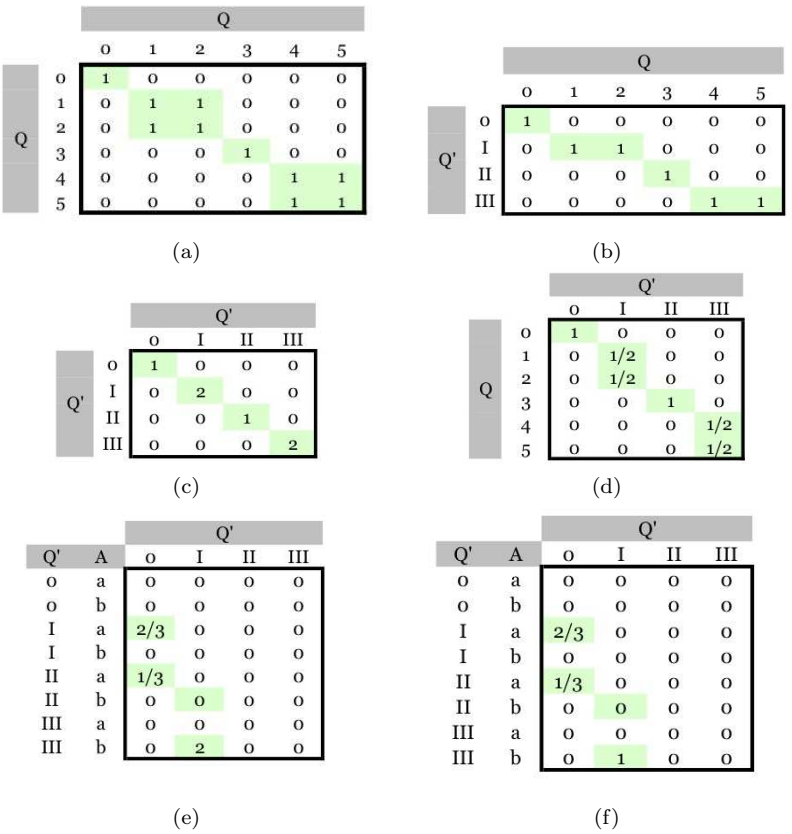


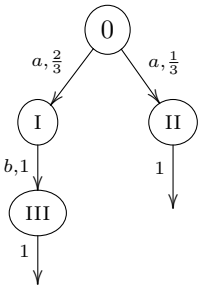
Fig. 2. Equivalence relation K (a) on state space Q is the kernel of surjection h (b) mapping Q to another state space $Q' = \{0, \mathrm{I}, \mathrm{II}, \mathrm{III}\}$. The image $h \cdot h^\circ$ of h is a diagonal (c) which tells how much non-injective h is. The inverse $h^\bullet$ (d) is used to compute reduced-state automaton (f), fixing (e) in which converse h° is used.

be observed that its *kernel* [10], $K = Q \xleftarrow{h^\circ \cdot h} Q$, is the given equivalence [e].

Diagram (15) suggests a first attempt to build the quotient automaton, $W' = W/K = (\mathsf{F}h) \cdot W \cdot h^\circ$, that is $\mu' = \mu/K = (h \otimes id) \cdot \mu \cdot h^\circ$ focusing on μ, μ' only. But this doesn't work because, in $Mat_{\mathbb{S}}$, h° is not a "true" converse of h, as the *image* of h ($h \cdot h^\circ$) is not below *id*. In fact, $h \cdot h^\circ > id$ is a *diagonal* counting "how much non-injective" h is, cf. Figure 2(c).

Fortunately, a *surjective* function h has inverses such as eg. $h^\bullet = h^\circ \cdot (h \cdot h^\circ)^{-1}$, obtained by standard *inversion* of diagonal $h \cdot h^\circ$, see Figure 2(d). Using $h^\bullet$ instead of h° one defines $W' = W/K = (\mathsf{F}h) \cdot W \cdot h^\bullet$, that is, $\mu' = \mu/K = (h \otimes id) \cdot \mu \cdot h^\bullet$ and $\gamma' = \gamma/K = \gamma \cdot h^\bullet$, leading to the automaton pictured below. Matrix μ' is shown in Figure 2(f).

[e]Kernels of functions are always equivalence relations, that is, $\{0, 1\}$-matrices.



This leads into the following pointfree definition of *weighted bisimulation equivalence*: equivalence relation $Q \xleftarrow{K} Q$ is a bisimulation for weighted automaton $\mathsf{F}Q \xleftarrow{W} Q$ provided any surjection $Q' \xleftarrow{h} Q$ that represents K, ie. such that $K = h^\circ \cdot h$, is such that

$$\mathsf{F}h \cdot W = \mathsf{F}h \cdot W \cdot K_\bullet \tag{19}$$

where $K_\bullet$ is the "weighted (kernel) equivalence" $K_\bullet = h^\bullet \cdot h$. (Clearly, $K_\bullet = K$ for injective h.) Composing both terms of (19) with $\mathsf{F}h^\circ$ we obtain the following consequence of the definition,

$$\mathsf{F}K \cdot W = \mathsf{F}K \cdot W \cdot K_\bullet \tag{20}$$

expressing the fact that $\mathsf{F}K \cdot W$ is an *invariant* of the weighted kernel $K_\bullet$.

Let us now relate the bisimulation construction above with the standard definition originating from [11]. Equalities

$$(K \otimes id) \cdot \mu = (K \otimes id) \cdot \mu \cdot K_\bullet \tag{21}$$

$$\gamma = \gamma \cdot K_\bullet \tag{22}$$

are obtained from (20) by unfolding F and simplifying. Noting that $\mathsf{F}K$ is an equivalence relation (as K is so and F is a functor), we start by unfolding the invariant component $(K \otimes id) \cdot \mu$:

$$\begin{array}{ll} & (q,a)((K \otimes id) \cdot \mu)p \\ = & \quad \{ \text{ composition rule (1) } \} \\ & \langle \sum q', a' :: (q,a)(K \otimes id)(q',a') \times ((q',a')\mu(p)) \rangle \\ = & \quad \{ \text{ Kronecker (14) } \} \\ & \langle \sum q', a' :: (qKq') \times (a = a') \times ((q',a')\mu(p)) \rangle \\ = & \quad \{ \text{ term } qKq' \text{ is Boolean ; one-point } a = a' \} \\ & \langle \sum q' : qKq' : (q',a')\mu(p) \rangle \\ = & \quad \{ \text{ let } [q]_K \text{ denote the equivalence class of } q \} \\ & \langle \sum q' : q' \in [q]_K : \ q' \xleftarrow{a} p \rangle \end{array}$$

Thus $\langle \sum q' : q' \in [q]_K : \ q' \xleftarrow{a} p \rangle$ is the accumulated cost (eg. probability) of transitions within the same equivalence class, which should be invariant for equivalent initial states according to the original definition [11]. To check this, first of all note that equivalence of initial states is captured by weight $q_1 \ K_\bullet \ q_2$ which un-

folds to $(h(q_1))(h \cdot h^\circ)^{-1}(h(q_2))$, recall (18) [f]. Assuming a finite number of states, standard diagonal inversion makes weighted equivalence be of the form

$$p' K_\bullet \; p = \frac{1}{|p|_K} p' K \; p \tag{23}$$

where $|p|_K$ is the cardinal of equivalence class $[p]_K$. This helps in the following expansion of the right hand side of (21):

$$\begin{array}{ll} & (q,a)((K \otimes id) \cdot \mu \cdot K_\bullet)p \\ \equiv & \quad \{ \text{ composition (1) ; (23) } \} \\ & \langle \sum p' :: (q,a)((K \otimes id) \cdot \mu)p' \times \frac{1}{|p|_K} p' K \; p \rangle \\ \equiv & \quad \{ \text{ linearity ; } p' K \; p \text{ is Boolean } \} \\ & \frac{1}{|p|_K} \langle \sum p' : p' K \; p : (q,a)((K \otimes id) \cdot \mu)p' \rangle \\ \equiv & \quad \{ \text{ recall the calculation of } (K \otimes id) \cdot \mu \text{ above } \} \\ & \frac{1}{|p|_K} \langle \sum p' : p' K \; p : \langle \sum q' : q' \in [q]_K : \; q' \xleftarrow{a} p \; \rangle \rangle \\ \equiv & \quad \{ \text{ rearranging the summations } \} \\ & \frac{1}{|p|_K} \langle \sum p', q' : p' \in [p]_K \wedge q' \in [q]_K : \; q' \xleftarrow{a} p \; \rangle \end{array}$$

The following notation abbreviation, for R, S subsets of Q,

$$S \xleftarrow{a} R \; = \langle \sum p, q : p \in R \wedge q \in S : \; q \xleftarrow{a} p \; \rangle \tag{24}$$

will help in putting everything together in an easy-to-read equality: equivalence K is a weighted bisimulation provided

$$[q]_K \xleftarrow{a} p \; = \frac{1}{|p|_K} \times (\; [q]_K \xleftarrow{a} [p]_K \;) \tag{25}$$

holds, which renders not only (21) but also (22) pointwise, as this amounts to $S = \emptyset$ in (24), ie. $\xleftarrow{} p \; = \frac{1}{|p|_K} \times (\; \xleftarrow{} [p]_K \;)$ where $\xleftarrow{} p$ is weight $\gamma(p)$. As the equality is universally quantified on p, it abbreviates a system of equations whose unique solution forces all weights the same within the same equivalence class. The same reasoning applied to (25) will lead to

$$\langle \forall q, p, p', a : p \; K \; p' : \; [q]_K \xleftarrow{a} p \; = \; [q]_K \xleftarrow{a} p' \; \rangle$$

which recovers the original definition by Larsen and Skou [11].

[f]Note in passing that diagonal $(h \cdot h^\circ)^{-1}$ represents the *weight vector [which] is well known in stochastic modeling* [7].

8. Weighted languages

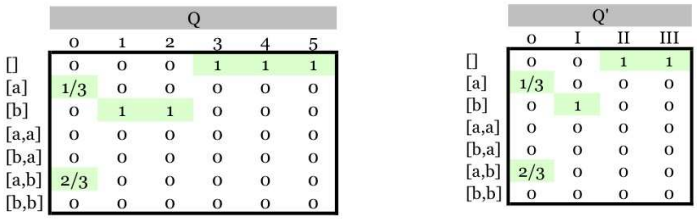
WA semantics is usually characterized by the weights they assign to finite action sequences [7]. This can be expressed in LA in terms of *weighted languages* [6]. A weighted language over A is a function $\sigma : A^\star \to \mathbb{S}$ assigning a weight to each word in $A^\star$. The function $L_W : Q \to \mathbb{S}^{A^\star}$ which associates to each state in Q of W its recognized weighted language [6] can, as before, be encoded into a $Mat_{\mathbb{S}}$ matrix of type $Q \longrightarrow\!\!\!\!\!\rightarrow A^\star$, ie. into the $Mat_{\mathbb{S}}$ F-comorphism

$$\begin{array}{ccc} Q \times A + 1 & \xleftarrow{W} & Q \\ {\scriptstyle \mathsf{F}L_W}\downarrow & & \downarrow{\scriptstyle L_W} \\ A^\star \times A + 1 & \xleftarrow[out]{} & A^\star \end{array} \qquad \text{where} \qquad \begin{array}{l} out = [cons|nil]^\circ \\ nil\ \_ = \epsilon \\ cons(x,a) = a : x \end{array} \tag{26}$$

Function nil is the constant function which outputs the empty language ϵ, and $a : x$ denotes the outcome of prefixing word x by a.

What does comorphism L_W mean? As earlier on, we start by unfolding the matrix equality described by diagram (26):

$$\begin{array}{ll} & out \cdot L_W = (\mathsf{F}L_W) \cdot W \\ \equiv & \quad \{ \text{ unfolding definitions ; converses } \} \\ & \left[\dfrac{cons^\circ}{nil^\circ}\right] \cdot L_W = ((L_W \otimes id) \oplus id) \cdot \left[\dfrac{\mu}{\gamma}\right] \\ \equiv & \quad \{ \text{ fusion (9) and absorption (13) } \} \\ & \left[\dfrac{cons^\circ \cdot L_W}{nil^\circ \cdot L_W}\right] = \left[\dfrac{(L_W \otimes id) \cdot \mu}{\gamma}\right] \\ \equiv & \quad \{ \text{ equality (11) } \} \\ & \begin{cases} cons^\circ \cdot L_W = (L_W \otimes id) \cdot \mu \\ nil^\circ \cdot L_W = \gamma \end{cases} \\ \equiv & \quad \{ \text{ matrix extensional equality } \} \\ & \begin{cases} (w,a)(cons^\circ \cdot L_W)q = (w,a)((L_W \otimes id) \cdot \mu)q \\ 1(nil^\circ \cdot L_W)q = 1\gamma q \end{cases} \\ \equiv & \quad \{ \text{ thumb rule (18) twice; definitions of } cons \text{ and } nil \ \} \\ & \begin{cases} (a : w)\ L_W\ q = (w,a)((L_W \otimes id) \cdot \mu)q \\ \epsilon\ L_W\ q = \gamma(q) \end{cases} \\ \equiv & \quad \{ \text{ composition (1) } \} \\ & \begin{cases} (a : w)\ L_W\ q = \langle \sum q', a' :: (w,a)(L_W \otimes id)(q',a') \times (q',a')\mu\ q \rangle \\ \epsilon\ L_W\ q = \gamma(q) \end{cases} \end{array}$$



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

(a) L_W

	Q'			
	0	I	II	III
[]	0	0	1	1
[a]	1/3	0	0	0
[b]	0	1	0	0
[a,a]	0	0	0	0
[b,a]	0	0	0	0
[a,b]	2/3	0	0	0
[b,b]	0	0	0	0

(b) $L_{W/K}$

Fig. 3. Weighted languages recognized by automata W and W/K in matrix format.

$$\equiv \quad \{ \text{ simplification (as before) } \}$$

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

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.* Another way to look at matrix L_W is:

$$\begin{aligned} & out \cdot L_W = ((L_W \otimes id) \oplus id) \cdot W \\ \equiv \quad & \{ \ out \text{ is an isomorphism ; } W \text{ (6) ; converses (3) } \} \\ & L_W = [cons|nil] \cdot \left[\frac{(L_W \otimes id) \cdot \mu}{\gamma}\right] \\ \equiv \quad & \{ \text{ divide and conquer (7) } \} \\ & L_W = cons \cdot (L_W \otimes id) \cdot \mu + nil \cdot \gamma \end{aligned}$$

This shows how L_W is (recursively) filled up over type $A^\star \xleftarrow{} Q$, adding to $nil \cdot \gamma$ (the matrix with γ as first row and 0s everywhere else) successive rows as dictated by *cons*. Using this definition for the example automaton of Figure 1 we obtain L_W as the fixpoint depicted in Figure 3 (a), and L'_W (b) for the reduced state automaton where, clearly, bisimilar states assign the same weights to the same words. This fact can be asserted in general: for all q, q', qKq' implies $L_W(q) = L_W(q')$ [6]. This is expressed quantitatively by $L_W \cdot K_\bullet = L_W$, as we show next:

$$\begin{aligned} & L_W \cdot K_\bullet = L_W \\ \equiv \quad & \{ \ out \text{ is an isomorphism } \} \\ & out \cdot L_W \cdot K_\bullet = out \cdot L_W \\ \equiv \quad & \{ \ out \cdot L_W = \mathsf{F}L_W \cdot W \text{ (26) } \} \\ & \mathsf{F}L_W \cdot W \cdot K_\bullet = out \cdot L_W \end{aligned}$$

$$
\begin{array}{ll}
\equiv & \{\ L'_W \cdot h = L_W \ (\mathit{out} \text{ is final}) \ ; \text{ functor } \mathsf{F}\ \} \\
& \mathsf{F}L'_W \cdot \mathsf{F}h \cdot W \cdot K_\bullet \ = \ \mathit{out} \cdot L_W \\
\equiv & \{\ \mathsf{F}h \cdot W \cdot K_\bullet = \mathsf{F}h \cdot W \ (19)\ \} \\
& \mathsf{F}L'_W \cdot \mathsf{F}h \cdot W \ = \ \mathit{out} \cdot L_W \\
\equiv & \{\ \text{functor } \mathsf{F} \ ; \ L'_W \cdot h = L_W \ \} \\
& \mathsf{F}L_W \cdot W \ = \ \mathit{out} \cdot L_W \\
\equiv & \{\ (26)\ \} \\
& \mathit{true}
\end{array}
$$

Finality of *out* is handled in [6] by deriving *linear weighted automaton* $W^{\#}$ from W, whose states are vectors of type $Q \leftarrow 1$. Thus $W^{\#}$, a coalgebra in the category of *vector spaces*, is expressed in typed LA by $\gamma^{\#}(v) = \gamma \cdot v$ and $\mu^{\#}(v)(a) = (id \otimes a) \cdot \mu \cdot v$, for state vector $Q \xleftarrow{v} 1$ and $1 \xleftarrow{a} A$ the membership of label a in A.

9. Conclusions

The evolution from non-deterministic to weighted automata witnesses a shift from *qualitative* to *quantitative* methods in computer science. In such a trend there seem to be two main approaches: either *reinvent* (extend) the original definitions in the *same* mathematical setting (category), or keep the *original* (categorial) definitions but *change* to an *enriched* category. Most approaches stay with the first alternative; in the current paper we argue in favour of the latter by proposing categories of matrices as the *natural evolution* of the category of sets towards quantification.

Category $Mat_{\mathbb{S}}$ of all matrices over semiring $\mathbb{S}$ offers a *quantifier-free*, *typed linear algebra* (LA) coalgebraic approach to weighted automata which retains the original simplicity of the corresponding qualitative approach. This adds to previous work [12, 15] in which the same techniques are applied to areas as diverse as eg. data mining [13] or probabilistic recursive program calculation [19].

In the case of weighted automata, LA is a natural choice already identified by other researchers. Buchholz [7], for instance, praises matrix notation because it *allows an elegant and compact formulation of the theory.* Trcka [25] writes that *matrices (...) increase clarity and compactness, simplify proofs, make known results from linear algebra directly applicable* and also mentions their *didactic advantage.*

Both [7, 25] have influenced the approach of the current paper. But the main inspiration comes from [6], in which coalgebras and linear algebra are used on a different plan: triggered by the need to extend the powerset functor quantitatively, *vector spaces* are introduced to weight multi-way state evolution. In a sense, powersets become "metric". Thus coalgebras in [6] involve functors $\mathsf{W} = \mathbb{K} \times (\mathbb{K}^{-}_{\omega})^{A}$ over a field $\mathbb{K}$, where $\mathbb{K}^{-}_{\omega}$ is the so-called field valuation (exponential) functor, and

$\mathsf{L} = \mathbb{K} \times (-)^A$ on vector spaces. Our approach flattens such exponentials by changing category: sets and functions (and linear transformations) give room to matrices built on top of $\mathbb{K}$. As $(\_)^A$ with sets becomes $(\_) \times A$ with matrices, by successively dropping $\mathbb{K}$s and flattening exponentials, weights no longer need to be taken explicitly into account, as the underlying matrix algebra circumspectly takes care of them.

In broad terms, this paper proposes that LA be *typed* on top of a categorial setting in which index-free matrix terms form the main notation, diagrammatic representations and proofs included. That is to say, rather than accepting LA arguments embedded in ordinary set-theoretical reasoning, we propose that typed LA be regarded as a *lingua franca* for computing, the other approaches coming as suitable instantiations.

We should say we are not the first proposing this kind of strategy. The acronym LAoP, for "linear algebra of programming" has been put forward already, albeit in a somewhat different setting, by Sernadas et al. [22], the key idea being *to adopt linear algebra as the lingua franca of software verification.*

Our main contribution is the emphasis on LA *polymorphic typing.* For this to work in practice, we believe that LA interfacing with standard logic, set theory and relation algebra should not be neglected [18]. Schmidt [21] already relies on matrix notation for doing relation algebra. Our experiments with eg. the Eindhoven quantifier notation show that the interface between functions, relations, predicates and matrices is (at least pedagogically) relevant. The infix notation we adopt for matrix entries — yMx rather than $M(y, x)$ — intends to bridge with that commonly used for binary relations, eg. $y \leq x$ preferred to $\leq (y, x)$.

10. Future work

Much remains to be done, in particular calling for the unification with related work. In particular, we would like to relate our ideas with those of Trcka [25], who presents a matrix approach to the notions of strong, weak and branching bisimulation ranging from labeled transition systems to Markov reward chains [g]. This already is the aim of Buchholz [7], who targets at a *universal definition of bisimulation which can be applied to a wide class of model types such that the different forms of bisimulation can all be seen as specific cases*, helping to unify system analysis.

We believe matrix types will improve the approaches of both [7] and [25] in a significant way. But, above all, in its use of matrix categories our strategy is close to the iteration theory $Mat_{\mathcal{L}(X^*)}$ of Bloom et al. [5] whose morphisms are matrices with entries in the semiring of languages. We intend to investigate the relationship between both approaches in a thorough way.

Last but not least, another target is the *linear algebra of components* which,

[g]Note that the matrices of [25] are of the form $Q \times Q \to \mathcal{P}A$ and therefore not truly "quantitative", in the sense that the additive operation over $\mathcal{P}A$ is idempotent.

anticipated in [14], promises a quantitative expansion of the coalgebraic approach of Barbosa [4] on software components. We believe that quantitative software reliability [8] can be approached coalgebraicly and hope this paper contributes with foundations for such follow-up research.

Acknowledgements

This paper is an expanded version of extended abstract [18]. The author is indebted to Nelma Moreira for comments on an earlier draft. This research was carried out in the QAIS (Quantitative analysis of interacting systems) project funded by the ERDF through the Programme COMPETE and by the Portuguese Government through FCT (Foundation for Science and Technology) contract PTDC/EIA-CCO/122240/2010.

References

[1] Abadir, K., Magnus, J.: Matrix algebra. Econometric exercises 1. C.U.P. (2005)
[2] Backhouse, R., Michaelis, D.: Exercises in quantifier manipulation. In: Uustalu, T. (ed.) MPC'06, LNCS, vol. 4014, pp. 70–81. Springer (2006)
[3] Backhouse, R.: Mathematics of Program Construction. Univ. of Nottingham (2004), draft of book in preparation. 608 pages
[4] Barbosa, L.: Towards a Calculus of State-based Software Components. JUCS 9(8), 891–909 (August 2003)
[5] Bloom, S., Sabadini, N., Walters, R.: Matrices, machines and behaviors. Applied Categorical Structures 4(4), 343–360 (1996)
[6] Bonchi, F., Bonsangue, M., Boreale, M., Rutten, J., Silva, A.: A coalgebraic perspective on linear weighted automata. Inf. & Comp. 211, 77–105 (2012)
[7] Buchholz, P.: Bisimulation relations for weighted automata. Theoretical Computer Science 393(1-3), 109–123 (2008)
[8] Cortellessa, V., Grassi, V.: A modeling approach to analyze the impact of error propagation on reliability of component-based systems. In: Component-Based Software Engineering, LNCS, vol. 4608, pp. 140–156 (2007)
[9] Droste, M., Gastin, P.: Weighted automata and weighted logics. In: Kuich, W., Vogler, H., Droste, M. (eds.) Handbook of Weighted Automata, chap. 5, pp. 175–211. EATCS Mon. in TCS, Springer (2009)
[10] Gumm, H., Schröder, T.: Products of coalgebras. Algebra Universalis 46, 163–185 (2001)
[11] Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. 94(1), 1–28 (1991)
[12] Macedo, H., Oliveira, J.: Matrices As Arrows! A Biproduct Approach to Typed Linear Algebra. In: MPC'10. LNCS, vol. 6120, pp. 271–287. Springer (2010)
[13] Macedo, H., Oliveira, J.: Do the middle letters of "OLAP" stand for linear algebra ("LA")? TR-HASLab:04:2011, INESC TEC & U.Minho (2011)

[14] Macedo, H., Oliveira, J.: Towards linear algebras of components. In: FACS 2010. LNCS, vol. 6921, pp. 300–303. Springer (2011)
[15] Macedo, H., Oliveira, J.: Typing linear algebra: A biproduct-oriented approach (2011), (Accepted for publication in SCP)
[16] MacLane, S.: Categories for the Working Mathematician. Springer (1971)
[17] MacLane, S., Birkhoff, G.: Algebra. AMS Chelsea (1999)
[18] Oliveira, J.: Typed linear algebra for weighted (probabilistic) automata. In: CIAA. LNCS, vol. 7381, pp. 52–65 (2012)
[19] Oliveira, J.N.: Towards a linear algebra of programming. Formal Asp. Comput. 24(4-6), 433–458 (2012)
[20] Rutten, J.: Universal coalgebra: A theory of systems. Theor. Comp. Sci. 249(1), 3–80 (2000), (Revised version of CWI Techn. Rep. CS-R9652, 1996)
[21] Schmidt, G.: Relational Mathematics. No. 132 in Encyclopedia of Mathematics and its Applications, Cambridge University Press (November 2010)
[22] Sernadas, A., Ramos, J., Mateus, P.: Linear algebra techniques for deciding the correctness of probabilistic programs with bounded resources. Tech. rep., TU Lisbon, 1049-001 Lisboa, Portugal (2008)
[23] Silva, A., Bonchi, F., Bonsangue, M., Rutten, J.: Quantitative Kleene coalgebras. Inf. Comput. 209(5), 822–849 (2011)
[24] Sokolova, A.: Coalgebraic Analysis of Probabilistic Systems. Ph.D. dissertation, Tech. Univ. Eindhoven, Eindhoven, The Netherlands (2005)
[25] Trcka, N.: Strong, weak and branching bisimulation for transition systems and Markov reward chains: A unifying matrix approach. EPTCS, vol. 13, pp. 55–65 (2009)