Under consideration for publication in Formal Aspects of Computing

Towards a Linear Algebra of Programming

José N. Oliveira[1]

[1] High Assurance Software Lab / INESC TEC & Univ. Minho, Portugal

Abstract. The Algebra of Programming (AoP) is a discipline for programming from specifications using relation algebra. Specification vagueness and nondeterminism are captured by relations. (Final) implementations are functions.

Probabilistic functions are half way between relations and functions: they express the propensity, or likelihood of ambiguous, multiple outputs. This paper puts forward a basis for a Linear Algebra of Programming (LAoP) extending standard AoP towards probabilistic functions. Because of the quantitative essence of these functions, the allegory of binary relations which supports the AoP has to be extended. We show that, if one restricts to discrete probability spaces, categories of matrices provide adequate support for the extension, while preserving the pointfree reasoning style typical of the AoP.

Keywords: Algebra of programming, quantitative formal methods, probabilistic reasoning

1. Introduction

The Algebra of Programming (AoP) [BdM97] is a discipline for *programming from specifications* [Mor90] in a pointfree, calculational manner. (Final) implementations are regarded as functions and specifications are regarded as relations. The calculus is *typed relation algebra* [FS90].

Functions are special cases of relations — the deterministic, totally defined ones. But relations, too, can be regarded as special cases of functions — the set-valued ones, as captured by universal property:

$$f = \Lambda R \quad \equiv \quad \langle \forall\, b, a \ :: \ bRa \equiv b \in f\ a \rangle \tag{1}$$

This tells that a binary relation R can be uniquely represented by a function ΛR which yields, for a given input a, the (possibly empty) set of all b which R relates to a. Conversely, any set-valued function f "is" a relation — precisely the one ($R = \in \cdot f$, where $\cdot$ denotes composition) which relates every input a of f to every *possible* outcome in set $f\ a$.

Note the word "possible" in the previous paragraph: it means that any such outcome may be output but

---

Correspondence and offprint requests to: José N. Oliveira, High Assurance Software Lab / INESC TEC & Univ. Minho, Gualtar Campus, 4710-057 Braga, Portugal. E-mail: `jno@di.uminho.pt`

nothing is said about which outputs are more *likely* than others. Even if one could predict such a likelihood, or propensity, how would it be expressed?

Written in terms of types, (1) is the isomorphism

$$A \to \mathcal{P}B \underset{\Lambda}{\overset{(\in\cdot)}{\cong}} A \to B \qquad (2)$$

whose left hand side $A \to \mathcal{P}B$ is the *functional type* which can also be written $(\mathcal{P}B)^A$, where $\mathcal{P}B$ denotes the powerset of B (set of all its subsets) and right hand side is the *relational type* $A \to B$ of all relations $R \subseteq B \times A$. The isomorphism is established, from right to left, by the operator Λ known as *power transpose* [1]. Since $\mathcal{P}B$ is in turn isomorphic to 2^B, the set of all predicates on B, one might write $A \to 2^B$ for the type of f in (1), where $2 = \{0,1\}$ is the set of truth values, 0 meaning false and 1 meaning true. So, for every input $a \in A$, $f\ a$ is the predicate which tells which $b \in B$ are likely as outputs.

Ranking output likelihood can be achieved by extending such predicates on B to *distributions* in $[0,1]^B$, where $[0,1]$ denotes the interval of real numbers between 0 and 1, which extends $\{0,1\}$ to the reals. Not every function $\mu \in [0,1]^B$ qualifies: only those such that $\sum_{b \in B} \mu\ b = 1$ holds. By defining [Sok05]

$$\mathcal{D}B = \{\mu \in [0,1]^B \mid \sum_{b \in B} \mu\ b = 1\} \qquad (3)$$

we will regard $A \to \mathcal{D}B$ as the type of all *probabilistic functions* from A to B. Writing $b =_p f\ a$ as alternative to $(f\ a)b = p$, both meaning that f outputs b for input a with probability p, one might regard the following probabilistic factorial function

$$\begin{aligned} fac\ 0 &=_{.95} 1 \\ fac\ 0 &=_{.05} 2 \\ fac(n+1) &=_1 (n+1) \times fac\ n \end{aligned}$$

as a model of a *faulty* factorial function which, with low probability (5%), can wrongly double the factorial of a number. Subscript 1 in the inductive case (100% probability) can be taken as default and omitted.

Probabilistic functions have been around in various guises. For $B = A$ they can be regarded as *Markov chains* [KS76] or as (elementary) *probabilistic coalgebras* [Sok05]. The latter reference deals with a hierarchy of probabilistic coalgebras built upon the distribution functor $\mathcal{D}_-$ which, on a different register, is implemented in [EK06] as a monad which supports the Haskell library PFP (Probabilistic Functional Programming). They illustrate the use of basic library functions with examples which demonstrate the high-level declarative style of probabilistic functional programming.

On a different front, functions such as the adulterated factorial above can be regarded as an instance of software *fault injection* [VM97], a more and more widespread technique for quality software assurance by measuring the propagation of a fault through paths that might otherwise rarely be followed in testing. The prospect of predicting the behaviour of such adulterated code rather than testing (running) it calls for an algebra of probabilistic functions similar to that developed by for "normal" functions and relations in [BdM97]. (Note that normal, "sharp" functions of type $A \to B$ are special cases of probabilistic functions — those such that every output distribution is 1 on a single $b \in B$ and 0 everywhere else — the Dirac δ function.)

The question arises: how much of the Algebra of Programming [BdM97] extends to probabilistic functions? This question may be addressed in several ways. One is to regard probabilistic functions just as normal, $\mathcal{D}$-resultic functions and stay within functions. But staying with functions is not the lesson learnt from the AoP, where functions are most of the time handled as relations, making rich operators such as converse and division universally applicable.

In the same vein, one should search for an isomorphism similar to (1), this time with $\mathcal{D}B$ instead of $\mathcal{P}B$. This is not hard to achieve: just write $(\mathcal{D}B)^A$ instead of $A \to \mathcal{D}B$ and expand $\mathcal{D}B$ into $[0,1]^B$, temporarily leaving aside the requirement captured by the summation in (3): by uncurrying, $([0,1]^B)^A$ is isomorphic

[1] See [FS90, BdM97].

to $[0,1]^{B\times A}$ which can be regarded as the mathematical space of all $[0,1]$-valued *matrices* with as many columns as elements in A and rows as elements in B. So, given probabilistic function $A \xrightarrow{f} \mathcal{D}B$, its *matrix transform* $[\![f]\!]$ is the unique matrix M such that

$$M = [\![f]\!] \;\equiv\; \langle \forall\ b,a\ ::\ M(b,a) = (f\ a)b \rangle \tag{4}$$

holds. Recalling (3), every such matrix will be such that all its columns sum up to 1 — a *left-stochastic* (LS) matrix [2].

How useful is such a matrix transpose? The main aim of this paper is to show that it is as productive with respect to probabilistic functions as relations are concerning standard, sharp functions. But, for this to happen, such matrices should be regarded not as mere *rectangles with numbers* (data structures) but rather as *typed* mathematical objects describing computations.

Under the slogan "matrices as arrows", the authors of [MO10] have campaigned in this direction, putting matrices side by side with relations in the style of Schmidt's *relational mathematics* [3]. In the same vein, the current paper will show how typed, index-free reasoning helps in keeping probability (convoluted) notation under control. For instance, we will show that expression

$$! \cdot [\![f]\!] = \,! \tag{5}$$

will be enough to capture the fact that the columns of $[\![f]\!]$ are distributions as in (3), putting summations off the way, where $M \cdot N$ denotes matrix-matrix multiplication and ! is a special vector wholly filled with 1s. (Details later on.)

Structure of the paper. We start by reviewing the essentials of "matrices as arrows" [MO10] in section 2. Section 3 shows how to encode elementary probability theory in such a typed linear algebra. Section 4 expresses a number of probabilistic extensions to standard AoP combinators using the matrix transform. Section 5 illustrates the emerging linear algebra of programming with the calculation of a result concerning probabilistic folds. Section 6 reviews related work. Finally, section 7 concludes and points directions for future work.

2. Typed linear algebra

Matrices as arrows. A matrix M with n rows and m columns is a function $M(r,c)$ which tells the value occupying each cell (r,c), for $1 \le r \le n$, $1 \le c \le m$. In this paper we will follow the arrow notation of [MO10] and write $n \xleftarrow{M} m$ to denote that matrix M is of *type* $n \longleftarrow m$ (m columns, n rows). Thus matrix-matrix multiplication can be expressed by arrow composition:

$$n \xleftarrow{M} m \xleftarrow{N} k \qquad (n \xleftarrow{P = M\cdot N} k) \tag{6}$$

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 $n \xleftarrow{1} n$, which is the diagonal of size n, that is, $id(r,c) \triangleq r = c$ ($x \triangleq y$ means $x = y$ by definition) under the $\{0,1\}$ encoding of the Booleans:

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

[2] Recall that a left (resp. right) stochastic matrix is a matrix of nonnegative reals each of whose columns (resp. row) sums up to 1 [LR99].

[3] See [Sch10] and references thereof to the pioneering works in the field by de Morgan, Schröder, Tarski, Maddux and others. An account of the origins of relation algebra can also be found in [Mad91] and [Pra92]. A categorial (typed) approach to relations (generally, to matrices over *locales*) can be found in [FS90].

Therefore:

$$id_n \cdot M \;=\; M \;=\; M \cdot id_m \qquad \begin{array}{ccc} m & \xleftarrow{id_m} & m \\ {\scriptstyle M}\downarrow & \swarrow{\scriptstyle M} & \downarrow{\scriptstyle M} \\ n & \xleftarrow[id_n]{} & n \end{array} \tag{7}$$

Subscripts m and n can be omitted wherever the underlying diagrams are assumed.

Under composition $M \cdot N$ (6) 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 [Mac98, MB99]. Strictly speaking, there is one such category per matrix cell value algebra, which is usually a *field* but can also be a *semiring*, a *Kleene algebra*, etc. In this paper we stick to $Mat_{\mathbb{R}}$, the category of matrices over the field $\mathbb{R}$ of the real numbers. The interval $[0,1]$ will be particularly at target in handling probabilistic functions. The particular class of *Boolean* matrices, ie. $\mathbb{R}$-valued matrices only holding the extreme points of such an interval will be handled using operations on $\mathbb{R}$, eg. $+$, $\times$, etc and not disjunction, conjunction etc.

Vectors as arrows. Vectors are special cases of matrices in which one of the dimensions is 1, for instance

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_m \end{pmatrix} \qquad \text{and} \qquad w = (w_1 \quad \ldots \quad w_n)$$

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. M, M) denote arbitrary matrices.

Converse of a matrix. One of the kernel operations of linear algebra (LA) is *transposition*, whereby a given matrix changes shape by turning its rows into columns and vice-versa. Given matrix $n \xleftarrow{M} m$, notation $m \xleftarrow{M^\circ} n$ denotes its transpose, or converse. The following idempotence and contravariance laws hold:

$$(M^\circ)^\circ = M \tag{8}$$

$$(M \cdot N)^\circ = N^\circ \cdot M^\circ \tag{9}$$

Bilinearity. Given two matrices of the same type $m \xleftarrow{M,N} n$ it makes sense to add them up index-wise, leading to matrix $M + N$ where symbol $+$ promotes the underlying cell-level 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 and zero of matrix composition:

$$M + 0 \;=\; M \;=\; 0 + M \tag{10}$$

$$M \cdot 0 \;=\; 0 \;=\; 0 \cdot M \tag{11}$$

Composition is bilinear relative to $+$:

$$M \cdot (N + P) = M \cdot N + M \cdot C \tag{12}$$

$$(N + P) \cdot M = N \cdot M + P \cdot M \tag{13}$$

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 cell-level 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,

$$M \times \top \;=\; \top \times M \;=\; M \tag{14}$$

where matrix $\top$ is of the same type as M and is wholly filled with 1s — a Boolean matrix.

Boolean matrices. A matrix M is Boolean iff

$$M \times M = M \tag{15}$$

that is, its entries are either 0 or 1 [4]. One such matrix is the identity (7). The following symbols will denote other Boolean matrices which will play a role in the sequel:

- The *top* matrix $n \xleftarrow{\top} m$ wholly filled with 1s and already seen above — the largest Boolean matrix of its type.
- The *bottom* matrix $n \xleftarrow{\bot} m$, wholly filled with 0s — the smallest Boolean matrix of its type, also denoted by 0, recall (10,11).
- The *bang* (row) vector $1 \xleftarrow{!} m$, wholly filled with 1s.

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

$$1 \xleftarrow{!} m \;=\; 1 \xleftarrow{\top} m \tag{16}$$

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

$$\top \;=\; ! \;=\; id \tag{17}$$

Boolean matrices allow for the encoding of binary relations as matrices. Relation intersection corresponds to the matrix (Hadamard) product, $M \cap N = M \times N$ for Boolean matrices M, N of the same type. We define their *union* as follows:

$$M \cup N = M + N - M \times N \tag{18}$$

Note the need to subtract the entries which are both in M and in N to stay within Boolean matrices [5].

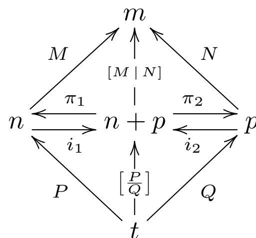
Boolean matrices at most the identity ($M \leq id$) will be referred to as *partial identities* or *coreflexive* matrices. We will tend to denote coreflexive matrices using capital Greek letters. Any coreflexive Φ is symmetric, that is, $\Phi^\circ = \Phi$. Two coreflexives Φ and Ψ are said to be *complementary* iff $\Phi + \Psi = id$.

The affinity between relations and matrices extends from Boolean matrices to arbitrary matrices. To see this we need to introduce the combinators which underlie matrix blocked notation.

Block notation. Two basic binary combinators are available for building matrices out of other matrices, say M and N:

- $[M|N]$ — M and N side by side (read $[M|N]$ as "M *junc* N")
- $\left[\frac{M}{N}\right]$ — M on top of N (read $\left[\frac{M}{N}\right]$ as "M *split* N").

That is, matrices are stacked either vertically ($\left[\frac{M}{N}\right]$) or horizontally ($[M|N]$). Dimensions should agree, as shown in the diagram below, taken from [MO10], where m, n, p and t are types:



$$[M|N] = M \cdot \pi_1 + N \cdot \pi_2 \tag{19}$$

$$\left[\frac{P}{Q}\right] = i_1 \cdot P + i_2 \cdot Q \tag{20}$$

4 Since $x = 0, 1$ are the two only solutions of equation $x^2 = x$ in $\mathbb{R}$.

5 Checking that $M \cup N$ (18) satisfies (15) in $Mat_{\mathbb{R}}$ is routine work once M and N are assumed to do so.

The special Boolean matrices i_1, i_2, π_1 and π_2 are fragments of the identity matrix as given by the so-called *reflexion laws* [6]:

$$[i_1 \,|\, i_2] = id \tag{21}$$

$$\left[\frac{\pi_1}{\pi_2}\right] = id \tag{22}$$

They play an important role in explaining the semantics of the two combinators. In brief, *junc* (19) and *split* (20) form a so-called *biproduct*. The details of this, however, can be skipped for the purposes of the current paper [7], sufficing to know the rich algebra of such combinators of which the most useful laws follow, where capital letters M, N, etc. denote suitably typed matrices (the types, i.e. dimensions, involved in each equality can be inferred by drawing the corresponding diagram):

- "Fusion"-laws:

$$P \cdot [M \,|\, N] = [P \cdot M \,|\, P \cdot N] \tag{23}$$

$$\left[\frac{M}{N}\right] \cdot P = \left[\frac{M \cdot P}{N \cdot P}\right] \tag{24}$$

- Divide-and-conquer:

$$[M \,|\, N] \cdot \left[\frac{P}{Q}\right] = M \cdot P + N \cdot Q \tag{25}$$

- Converse duality:

$$[R \,|\, S]^\circ = \left[\frac{R^\circ}{S^\circ}\right] \tag{26}$$

  Putting (26) and (25) together:

$$[M \,|\, N] \cdot [P \,|\, Q]^\circ = M \cdot P^\circ + N \cdot Q^\circ \tag{27}$$

- The *exchange law*:

$$\left[\frac{[M \,|\, N]}{[P \,|\, Q]}\right] \quad = \quad \left[\left[\frac{M}{P}\right] \,|\, \left[\frac{N}{Q}\right]\right] \quad = \quad \left[\begin{array}{c|c} M & N \\ \hline P & Q \end{array}\right] \tag{28}$$

The last property tells the equivalence between row-major and column-major construction of matrices by blocks — thus the four-block notation on the right.

Remarks concerning notation. Outfix notation such as that used in *splits* and *juncs* provides for unambiguous parsing of matrix algebra expressions. Concerning infix operators (such as eg. composition, $+$) and unary ones (eg. converse, and others to appear) the following conventions will be adopted for saving parentheses: (a) unary and prefix operators bind tighter than binary; (b) multiplicative binary operators bind tighter than additive ones; (c) matrix multiplication (composition) binds tighter than any other multiplicative operator (eg. Kronecker product, to appear later).

Type generalization. Matrix types (the end points of arrows) can be generalized to arbitrary, denumerable sets thanks to addition and multiplication being commutative and associative. This ensures unambiguous definition of matrix composition because the summation inside the inner product of any row/column vector pair can be calculated in any order.

In fact, and as is standard in relational mathematics [Sch10], 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 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

[6] Note however that neither of these matrices is a coreflexive, for both $i_1 + i_2$ and $\pi_1 + \pi_2$ are ill-typed.
[7] The interested reader is referred to [MO10] for the details omitted.

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.

In this paper we will focus on matrices taking elements from $\mathbb{R}_0^+$, the non-negative reals. The interval $[0,1]$ will be particularly at target in the case of probabilistic functions.

Type generalization enables us to establish the embedding of the allegory of binary relations [FS90] into categories of matrices in a more precise way: every relation $A \xrightarrow{R} B$ has a *matrix transform* $[\![R]\!]$ which is the unique (Boolean) matrix M of type $A \longrightarrow B$ such that:

$$M = [\![R]\!] \equiv \langle \forall\, y, x \,::\, M(y,x) = (y\; R\; x) \rangle \tag{29}$$

under the aforementioned $\{0,1\}$ encoding of the Booleans. Relation inclusion corresponds to matrix less-or-equal, $[\![R \subseteq S]\!] = [\![R]\!] \leq [\![S]\!]$. Intersection is the Hadamard product, $[\![R \cap S]\!] = [\![R]\!] \times [\![S]\!]$ and union is $[\![R \cup S]\!] = [\![R]\!] + [\![S]\!] - [\![R]\!] \times [\![S]\!]$, recall (18).

As functions are special cases of relations, transform (29) also applies to them. This explains why we use notation ! to denote any row vector of 1s: it is the transform of the homonym function $1 \xleftarrow{!} A$ which, in the category of sets, is the unique morphism which maps any type A to the singleton type 1, a terminal object in the category [BdM97].

Handling relations and functions as matrices enables a nice mix of qualitative and quantitative analysis, as the following way to classify Boolean matrices according to the terminology of [FS90, BdM97] shows: a Boolean matrix M is said to be

- simple — iff $! \cdot M \leq\, !$
- injective — iff M° is simple, that is, $M \cdot !^\circ \leq\, !^\circ$
- entire — iff $! \cdot M \geq\, !$
- surjective — iff M° is entire, that is, $M \cdot !^\circ \geq\, !^\circ$.

This is so because the *sum* of all entries of column vector $m \xleftarrow{v} 1$ is given by

$$1 \xleftarrow{!} m \xleftarrow{v} 1 \qquad (\text{composite: } 1 \xleftarrow{! \cdot v} 1)$$

and therefore, $! \cdot M$ yields the row vector which contains the sums of all columns of M. Since f being a *function* is being both *simple* and *entire*, one has that

$$! \cdot [\![f]\!] = \,! \tag{30}$$

always holds for functions — the matrix transpose of fact $! \cdot f = \,!$, the free theorem of polymorphic function !. It turns out that (30) extends to any probabilistic function $A \xrightarrow{f} \mathcal{D}B$, as already anticipated in (5). But there is a difference: $[\![f]\!]$ is not Boolean in the case of f probabilistic.

The interplay between matrices and relations enables a concise way to express statistical operations. For instance, the *normalization* of column vector $A \xleftarrow{v} 1 > \bot$ into a *distribution* μ is a vector of the same type as v,

$$\mu = \frac{v}{\top \cdot v}$$

where $\top$ is of type $A \longleftarrow A$ and $\frac{M}{N}$ denotes pointwise division on the same type.

Likewise, any matrix $B \xleftarrow{M} A > \bot$ filled with natural numbers counting *experiments* can be normalized into a function f of the same type such that $f(a)$ is the corresponding *probabilistic* (mass) function on B:

$$[\![f]\!] = \frac{M}{\top \cdot M}$$

That any probabilistic function is already normalized, that is, $\frac{[\![f]\!]}{\top \cdot [\![f]\!]} = [\![f]\!]$ is easy to show: $\top = \,!^\circ \cdot !$ and therefore $\top \cdot [\![f]\!] = \top$, thanks to (30); then $\frac{[\![f]\!]}{\top} = [\![f]\!]$ since $\frac{M}{\top} = M$, for any M. To complete this illustration,

suppose that $B \xleftarrow{\geq} B$ is a total ordering on B. Then $Q = \geq \cdot [\![f]\!]$ is a matrix which gives, for each input $a \in A$, the (discrete) *cumulative mass function* associated to (discrete) distribution $f(a)$. Note the nice mix among matrices and relations such as $\geq, \top$, used in matrix expressions as the Boolean matrices "they are".

3. Doing elementary probability theory with linear algebra

Clearly, any Boolean vector of type $1 \xleftarrow{!} A$ (16) represents a subset of A, corresponding to the *right-condition* encoding of subsets in relation algebra [Hoo97]. Let us use $[\![X]\!]$ to denote such a vectorial representation of a given $X \subseteq A$. Clearly, $[\![\emptyset]\!] = \bot$ and $[\![A]\!] = !$. This representation [8] makes set-theoretical definitions and reasoning surprisingly simple in many situations.

Take for instance the statement that $\{X, Y\}$ form a *partition* of A: $X \cup Y = A$ and $X \cap Y = \emptyset$. In the vectorial encoding above, writing

$$[\![X]\!] + [\![Y]\!] = ! \qquad (31)$$

(which could also be written $[\![X]\!] + [\![Y]\!] = \top$, recall (16)), is enough.

LA based set-theory reasoning is nicely calculational, as the following evidence that X and Y are necessarily disjoint shows, where $[\![X]\!]$ and $[\![Y]\!]$ are unambiguously abbreviated to X and Y, respectively, to spruce up the formulæ. The fact that the Hadamard product on Boolean matrices models intersection should be recalled:

$$\begin{array}{ll}
& X + Y = ! \\
\Longrightarrow & \quad \{ \text{ Leibniz } \} \\
& X \times (X + Y) = X \times ! \\
\equiv & \quad \{ \text{ Hadamard: linearity, } X \times X = X \cap X = X \text{ and unit } ! \ \} \\
& X + X \times Y = X \\
\equiv & \quad \{ \text{ subtract } X \text{ from both sides of the equation } \} \\
& X \times Y = X - X \\
\equiv & \quad \{ \text{ cancellation of inverses } \} \\
& X \times Y = \bot \\
\equiv & \quad \{ \ X \times Y = X \cap Y \text{ as Hadamard on Boolean matrices is intersection } \} \\
& X \text{ and } Y \text{ are disjoint}
\end{array}$$

Basic probability theory may also be handled in the same way. Let, for instance, S be some *sample space* [MM05] and function $S \xrightarrow{\mu} [0, 1]$ be a discrete probability distribution over the sample space, giving for each event $X \subseteq S$ the probability of its occurrence. As we have already seen, function μ can be encoded as a column vector of type $1 \xrightarrow{[\![\mu]\!]} S$ such that $! \cdot [\![\mu]\!] = !$. In the same vectorial notation, the probability $P(X)$ of event $X \subseteq S$ under μ will be given by [9]

$$P(X) = 1 \xleftarrow{[\![X]\!]} S \xleftarrow{[\![\mu]\!]} 1 \qquad \text{(with composite } 1 \xleftarrow{[\![X]\!] \cdot [\![\mu]\!]} 1\text{)} \qquad (32)$$

A *random variable* $S \xrightarrow{v} T$ on probability space (S, μ) induces a new probability distribution on T by

[8] Which extends from vectors to other Boolean matrices encoding subsets of Cartesian products.

[9] This corresponds to function `(??) :: Event a -> Dist a -> Probability` in the PFP library [EK06]. Note how the encoding of distributions as **column** vectors and sets as **row** vectors save us from the need to adopt Dirac's "bra-ket" notation, $P(X) = \langle [\![X]\!] | [\![\mu]\!] \rangle$, required in case both $[\![X]\!]$ and $[\![\mu]\!]$ were row vectors.

composition,

$$\mu' = [\![v]\!] \cdot [\![\mu]\!] \tag{33}$$

generating a new probability space (T, μ'). That μ' is indeed a distribution can be easily calculated:

$$\begin{array}{ll} & ! \cdot [\![\mu']\!] \\ = & \{ \ (33) \ \} \\ & ! \cdot ([\![v]\!] \cdot [\![\mu]\!]) \\ = & \{ \ v \text{ is a function (30)} \ \} \\ & ! \cdot [\![\mu]\!] \\ = & \{ \ \mu \text{ is a distribution} \ \} \\ & ! \end{array}$$

In the same setting, the independence of two events $A, B \subseteq S$ in probability space (S, μ),

$$A \ \mathbf{indep}. \ B \equiv P(A \cap B) = P(A)P(B)$$

takes the form of a distributive property:

$$A \ \mathbf{indep.} \ B \ \ \equiv \ \ ([\![A]\!] \times [\![B]\!]) \cdot [\![\mu]\!] \ = \ [\![A]\!] \cdot [\![\mu]\!] \times [\![B]\!] \cdot [\![\mu]\!] \tag{34}$$

Likewise, the *addition law* of probability

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

comes out as an easy-to-check consequence of matrix *union* (18) and linearity (13). The calculation of the law of *total probability*

$$P(A) = P(A \cap B_1) + P(A \cap B_2)$$

for $\{B_1, B_2\}$ a partition (31) of the sample space S, is another school exercise in LA (again sprucing up the layout by unambiguously dropping parentheses $[\![_]\!]$):

$$\begin{array}{ll} & P(A \cap B_1) + P(A \cap B_2) \\ = & \{ \ (32) \text{ twice} \ \} \\ & (A \times B_1) \cdot \mu + (A \times B_2) \cdot \mu \\ = & \{ \ \text{composition and Hadamard product are bilinear} \ \} \\ & (A \times (B_1 + B_2)) \cdot \mu \\ = & \{ \ \text{partition } B_1, \, B_2 \text{ (31)} \ \} \\ & (A \times !) \cdot \mu \\ = & \{ \ ! = \top \text{ is the unit of Hadamard } \times\text{; definition (32)} \ \} \\ & P(A) \end{array}$$

Finally, conditional probability is defined as expected,

$$P(A|B) = \frac{([\![A]\!] \times [\![B]\!]) \cdot [\![\mu]\!]}{[\![B]\!] \cdot [\![\mu]\!]}$$

whereby Bayes theorem is obtained by simple multiplicative inverse cancellation:

$$P(B|A) = P(A|B)\frac{P(B)}{P(A)}$$

4. Probabilistic functions in the pointfree style

As we have seen, the matrix transform of a probabilistic function $A \xrightarrow{f} \mathcal{D}B$ is a left stochastic (LS) matrix of type $A \longrightarrow B$ taking values in the interval $[0, 1]$. Isomorphism (4) unambiguously maps any such function f to its LS-matrix transpose $[\![f]\!]$ and vice versa. However, handling f directly as a function (in the category of sets) or its transform $[\![f]\!]$ in the category of LS-matrices makes a lot of difference, since f is *monadic* on its output [EK06], leading to unnecessarily complex reasoning.

That LS-matrices form a (sub)category of matrices is easy to show: the identity matrix *id* is LS-stochastic and the composition of two LS-matrices S, R is an LS-matrix:

$$
\begin{aligned}
& \ ! \cdot (S \cdot R) \\
= & \quad \{ \text{ composition is associative } \} \\
& \ (! \cdot S) \cdot R \\
= & \quad \{ \text{ (5) twice, as } S \text{ and } R \text{ are assumed LS } \} \\
& \ !
\end{aligned}
$$

We also observe that the LS-matrix (sub)category has coproducts, cf.

$$
\begin{aligned}
& \ ! \cdot [R|S] \\
= & \quad \{ \text{ fusion (23) } \} \\
& \ [! \cdot R|! \cdot S] \\
= & \quad \{ \ R \text{ and } S \text{ are assumed probabilistic } \} \\
& \ [!|!] \\
= & \quad \{ \ ! \text{ is unique, so } [!|!] = \ ! \ \} \\
& \ !
\end{aligned}
$$

LS-matrices offer support for pointfree reasoning about probabilistic functions much in the same way relations do for standard functions in the AoP, leading to what one may term a *linear algebra of programming* (LAoP). Let us see examples of how this works.

On probabilistic function application. As warming up exercise, let us see how to handle binary (n-ary in general) probabilistic function application, $f(a, b)$, in the LS-setting. We consider the unary case first, $f(a)$ denoting the application of an arbitrary function $A \xrightarrow{f} B$ to some point $a \in A$. Note that the same can be written by composing $f \cdot \underline{a}$, where $A \xleftarrow{\underline{a}} 1$ is a *point* in the categorial sense [LS97]:

$$B \xleftarrow{f} A \xleftarrow{\underline{a}} 1$$

Let us see what happens if we interpret the same diagram in the category of LS-matrices: f becomes probabilistic and, as LS-vectors are distributions, $\underline{a}$ becomes the "Dirac distribution" of point a [10]. So, in general, what makes sense is to apply probabilistic functions to distributions, writing $[\![f]\!] \cdot [\![\mu]\!]$ for what might be written, in monad-speak, $[\![\mu \gg\!= f]\!]$ in sets. One may even define probabilistic function (Kleisli) composition simply as

$$[\![f \bullet g]\!] = [\![f]\!] \cdot [\![g]\!] \tag{35}$$

since the $[\![_]\!]$ transpose is an isomorphism (4).

In order to save notation and add to readability, we will henceforth omit the $[\![_]\!]$ parentheses wherever the formulæ and diagrams we write are unambiguously interpreted in the LS-matrix category. That is to say,

[10] That is, *return* a in the $\mathcal{D}_$ monad [EK06].

writing $A \xrightarrow{f} B$ will mean the LS-matrix which represents probabilistic function $A \xrightarrow{f} \mathcal{D}B$. (Recall that the Boolean matrices which represent standard functions are LS.) Thus drawing $B \xleftarrow{f} A \xleftarrow{\mu} 1$ will mean $[\![f]\!] \cdot [\![\mu]\!]$.

Example: Monty-Hall. We provide a very simple illustration of matrix-transposed probabilistic functional application by taking the well-known *Monty-Hall* probabilistic puzzle as example. For an account of this brain teaser and the controversy it caused see eg. [Ros09]. For its mathematical treatment, references [MM05, EK06, Heh11, GH11] are recommended. The problem has been handled in a myriad of ways, ours following those which explore the puzzle symmetry: the contestant having just chosen one of the three doors and the host opened another one, the former has $\frac{1}{3}$ probability of having won the game, since there are three doors. The question is: is it worthwhile keeping the choice or betting on the one door left?

The starting state is a column vector of type $\{\mathbf{W}, \mathbf{L}\} \xleftarrow{\mu} 1$, where $\mathbf{W}$ (resp. $\mathbf{L}$) stands for *win* (resp. *lose*) — the distribution shown on the right where, in the style of [Sch10], labels are added to enhance type perception. Keeping the bet means opting for the identity function on states (nothing changes); by contrast, changing bet means changing state according to the function

$$[\![\mu]\!] = \begin{array}{c} \\ \mathbf{W} \\ \mathbf{L} \end{array} \begin{array}{c} 1 \\ \begin{pmatrix} \frac{1}{3} \\ \frac{2}{3} \end{pmatrix} \end{array}$$

$$\begin{aligned} &swap\ \mathbf{W} = \mathbf{L} \\ &swap\ \mathbf{L} = \mathbf{W} \end{aligned}$$

since if a closed door is the winning one the other is the losing one.

The matrix transpose of this function is the matrix on the right, the "twisted identity" of its type. The change of state is captured by the application $[\![swap]\!] \cdot [\![\mu]\!]$ and the probability of winning after such a change of state is obtained by the first projection ($P(\mathbf{W}) = \pi_1 = (1 \quad 0)$):

$$[\![swap]\!] = \begin{array}{c} \\ \mathbf{W} \\ \mathbf{L} \end{array} \begin{array}{c} \mathbf{W} \quad \mathbf{L} \\ \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array}$$

$$\pi_1 \cdot [\![swap]\!] \cdot [\![\mu]\!] = \frac{2}{3} > \frac{1}{3} = \pi_1 \cdot id \cdot [\![\mu]\!]$$

So, swapping is worthwhile.

Probabilistic binary function application. What is the probabilistic extension of a binary function f? In the AoP, writing $f(a, b)$ is the same as writing $f \cdot \underline{(a, b)}$, itself the same as writing $f \cdot (\underline{a} \vartriangle \underline{b})$, where $\vartriangle$ denotes the pairing operator

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

Can we write, as extension of $f \cdot (\underline{a} \vartriangle \underline{b})$ in the LS-matrix category, "$f \cdot (\mu \vartriangle \mu')$" for two input distributions μ and μ'? It turns out the the answer is affirmative, provided $\vartriangle$ is interpreted as the Khatri-Rao matrix product [RR98]. In general, given matrices $n \xleftarrow{A} m$ and $p \xleftarrow{B} m$, the Khatri-Rao product of A and B, denoted $n \times p \xleftarrow{A \vartriangle B} m$ is a column-wise Kronecker product. At this point we should recall that the Kronecker product is a bifunctor in a category of matrices, cf. diagram

$$\begin{array}{ccccc} n & & m & & n \times m \\ {\scriptstyle A}\downarrow & & {\scriptstyle B}\downarrow & & \downarrow{\scriptstyle A \otimes B} \\ k & & j & & k \times j \end{array}$$

whose fusion laws

$$[A \,|\, B] \otimes C = [A \otimes C \,|\, B \otimes C] \tag{36}$$

$$\left[\frac{A}{B}\right] \otimes C = \left[\frac{A \otimes C}{B \otimes C}\right] \tag{37}$$

capture its meaning block-wise. The Khatri-Rao product coincides with Kronecker for column vectors u and v,

$$u \vartriangle v = u \otimes v \tag{38}$$

and expands column-wise as shown by the *exchange law*

$$[A_1 \,|\, A_2] \vartriangle [B_1 \,|\, B_2] = [A_1 \vartriangle B_1 \,|\, A_2 \vartriangle B_2] \tag{39}$$

where A_i, B_i are suitably typed matrices. Other properties of this product are bilinearity

$$A \vartriangle (B + C) = (A \vartriangle B) + (A \vartriangle C) \tag{40}$$

$$(B + C) \vartriangle A = (B \vartriangle A) + (C \vartriangle A) \tag{41}$$

and unit:

$$! \vartriangle A \;=\; A \;=\; A \vartriangle ! \tag{42}$$

Since distributions are LS-vectors, one has $\mu \vartriangle \mu' = \mu \otimes \mu'$ (38). But, is $\mu \otimes \mu'$ a distribution? We calculate:

$$
\begin{array}{cl}
& ! \cdot (\mu \vartriangle \mu') = ! \\
\equiv & \quad \{ \text{ Khatri-Rao of vectors = Kronecker (38) } \} \\
& ! \cdot (\mu \otimes \mu') = ! \\
\equiv & \quad \{ \; ! \otimes ! = ! \; \} \\
& (! \otimes !) \cdot (\mu \otimes \mu') = ! \\
\equiv & \quad \{ \text{ Kronecker functor } \} \\
& (! \cdot \mu) \otimes (! \cdot \mu') = ! \\
\equiv & \quad \{ \; \mu \text{ and } \mu' \text{ are distributions } \} \\
& ! \otimes ! = ! \\
\equiv & \quad \{ \; ! \otimes ! = ! \text{ again } \} \\
& ! = !
\end{array}
$$

In summary, the pairing of two distributions is a distribution and the action of pairing two inputs a and b into (a, b) sent as input to a binary function in general corresponds, in the probabilistic setting, to having two input distributions and pairing them (using $\vartriangle$) before applying the binary probabilistic function. In symbols and full detail:

$$[\![(\mu, \mu') \ggg\!= f]\!] = [\![f]\!] \cdot ([\![\mu]\!] \vartriangle [\![\mu']\!])$$

where the pairing of distribution functions μ, μ' is the function obtained by "'zipping" the two functions with multiplication, cf. the `prod` operator in [EK06] [11]. Let us see a very simple example.

Example: probability of the sum. We want to illustrate the probabilistic extension of adding two numbers $n + m$, using the following toy example:

One spins two fair roulette wheels with 3 sectors labelled 1,2 and 3 and wants to know the probability of the sum of their outcome being a given number (between 1 and 6).

Each roulette being fair means that sectors 1, 2 or 3 have equal probability, that is, each wheel is captured by distribution μ such that $1 \xleftarrow{\mu^\circ} 3 = \left(\frac{1}{3} \;\; \frac{1}{3} \;\; \frac{1}{3}\right)$. Pairing two such roulettes yields distribution $3 \times 3 \xleftarrow{d \vartriangle d} 1$ which is a 9-cell column vector wholly filled with $\frac{1}{9}$. The sum function (restricted to inputs

[11] This is written $\mu \times \mu'$ in [Sok05].

at most 3) is LS-matrix

$$sum = \begin{pmatrix} 0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0\\ 0&1&0&1&0&0&0&0&0\\ 0&0&1&0&1&0&1&0&0\\ 0&0&0&0&0&1&0&1&0\\ 0&0&0&0&0&0&0&0&1 \end{pmatrix}$$

or, rendering types more explicit,

$sum =$	**(1,1)**	**(1,2)**	**(1,3)**	**(2,1)**	**(2,2)**	**(2,3)**	**(3,1)**	**(3,2)**	**(3,3)**
1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0	0	0
4	0	0	1	0	1	0	1	0	0
5	0	0	0	0	0	1	0	1	0
6	0	0	0	0	0	0	0	0	1

Then, composition

$$6 \xleftarrow{sum} 3 \times 3 \xleftarrow{d \triangle d} 1 \;=\; \begin{pmatrix} 0\\ 1/9\\ 2/9\\ 1/3\\ 2/9\\ 1/9 \end{pmatrix}$$

yields the distribution of the sum of the two roulette outcomes (1 impossible, 4 more likely than any other, 3 and 5 equally likely, etc).

Choice. In their programming language *pGCL*, McIver and Morgan [MM05] introduce notation

$$prog\ {}_p\!\oplus prog'$$

as a form of probabilistic choice between two branches of a program, the left hand side one being chosen with probability p and the other side with probability $1 - p$. In our setting, this corresponds to the static choice between two probabilistic functions f and g of the same type defined by

$$[\![f\ {}_p\!\diamond g]\!] = p \otimes [\![f]\!] + (1-p) \otimes [\![g]\!] \tag{43}$$

(we prefer notation ${}_p\!\diamond$ to ${}_p\!\oplus$ in order not to collide with the matrix-sum operator usually denoted by $\oplus$, see below) where $\otimes$ is the Kronecker product and p and $1 - p$ are scalars of type $1 \longleftarrow 1$. That $f\ {}_p\!\diamond g$ is probabilistic is easy to check:

$$\begin{array}{ll} & !\cdot [\![f\ {}_p\!\diamond g]\!] \\ = & \quad \{\ \text{definition (43) ; bilinearity ; parentheses } [\![_]\!] \text{ assumed }\ \} \\ & !\cdot (p \otimes f) + !\cdot ((1-p) \otimes g) \\ = & \quad \{\ p \text{ is a scalar }\ \} \\ & p \otimes (!\cdot f) + (1-p) \otimes (!\cdot g) \\ = & \quad \{\ f \text{ and } g \text{ are probabilistic }\ \} \\ & p \otimes ! + (1-p) \otimes ! \\ = & \quad \{\ \text{bilinearity }\ \} \\ & (p + 1 - p) \otimes ! \end{array}$$

$$= \quad \{ \text{ cancellation } \}$$

!

As an example of choice, consider the Monty-Hall contestant who will choose to swap door with probability 90%. The overall outcome of her/his decision will be captured by choice $swap\ {}_{0.9}\!\diamond\ id = \begin{pmatrix} 0.1 & 0.9 \\ 0.9 & 0.1 \end{pmatrix}$.

Choice has a number of properties which are easy to derive from (43) by linear algebra, for instance:

$$f\ {}_{p}\!\diamond\ f = f \tag{44}$$

$$f\ {}_{0}\!\diamond\ g = g \tag{45}$$

$$f\ {}_{1}\!\diamond\ g = f \tag{46}$$

$$f\ {}_{p}\!\diamond\ g = g\ {}_{1-p}\!\diamond\ f \tag{47}$$

$$f\ {}_{p}\!\diamond\ (g\ {}_{q}\!\diamond\ h) = (f\ {}_{p}\!\diamond\ g)\ {}_{q}\!\diamond\ (f\ {}_{p}\!\diamond\ h) \tag{48}$$

(Property (44) will help in checking (48) by starting to expand the leftmost f into $f\ {}_{q}\!\diamond f$.) The two fusion-laws of choice

$$(f\ {}_{p}\!\diamond\ g) \bullet h = (f \bullet h)\ {}_{p}\!\diamond\ (g \bullet h) \tag{49}$$

$$h \bullet (f\ {}_{p}\!\diamond\ g) = (h \bullet f)\ {}_{p}\!\diamond\ (h \bullet g) \tag{50}$$

emerge from bilinearity (12,13) and take a bit longer. We prove (49), the proof of (50) being similar:

$$[\![(f\ {}_{p}\!\diamond\ g) \bullet h]\!]$$

$$= \quad \{ \text{ (35) } \}$$

$$[\![f\ {}_{p}\!\diamond\ g]\!] \cdot [\![h]\!]$$

$$= \quad \{ \text{ (43) ; temporarily removing the parentheses } \}$$

$$((p \otimes f) + ((1-p) \otimes g)) \cdot h$$

$$= \quad \{ \text{ bilinearity } \}$$

$$(p \otimes f) \cdot h + ((1-p) \otimes g) \cdot h$$

$$= \quad \{ \ 1 \xleftarrow{id} 1 \ = 1 \text{ (a scalar) and } 1 \otimes M = M \ \}$$

$$(p \otimes f) \cdot (id \otimes h) + ((1-p) \otimes g) \cdot (id \otimes h)$$

$$= \quad \{ \text{ Kronecker product is a functor (51) } \}$$

$$(p \otimes (f \cdot h)) + ((1-p) \otimes (g \cdot h))$$

$$= \quad \{ \text{ (43) and (35) } \}$$

$$[\![(f \bullet h)\ {}_{p}\!\diamond\ (g \bullet h)]\!]$$

Mind the functorial property of the Kronecker product referred to above:

$$(A \otimes B) \cdot (C \otimes D) = (A \cdot C) \otimes (B \cdot D) \tag{51}$$

Choice also distributes over *junc*,

$$[f, g]\ {}_{p}\!\diamond\ [h, k] = [f\ {}_{p}\!\diamond\ h, g\ {}_{p}\!\diamond\ k] \tag{52}$$

for f (resp. g) and h (resp. k) of the same type, where $[_, _]$ denotes the junction of two functions in the standard way [BdM97]. This combinator translates to the matrix-*junc* combinator which puts matrices side

by side [12]:

$$[\![[f,g]]\!] = [[\![f]\!] \,|\, [\![g]\!]] \tag{53}$$

Concerning (52), we reason:

$$
\begin{array}{ll}
& [\![[f,g] \;{}_p\!\diamond\; [h,k]]\!] \\
= & \{ \;\; (43) \;\; \} \\
& p \otimes [\![[f,g]]\!] + (1-p) \otimes [\![[h,k]]\!] \\
= & \{ \;\; (53) \text{ twice ; } p \text{ and } 1-p \text{ are scalars } \} \\
& [p \otimes [\![f]\!] \,|\, p \otimes [\![g]\!]] + [(1-p) \otimes [\![h]\!] \,|\, (1-p) \otimes [\![k]\!]] \\
= & \{ \;\; f \text{ (resp. } g\text{) and } h \text{ (resp. } k\text{) are of the same type } \} \\
& [p \otimes [\![f]\!] + (1-p) \otimes [\![h]\!] \,|\, p \otimes [\![g]\!] + (1-p) \otimes [\![k]\!]] \\
= & \{ \;\; (43) \text{ twice } \} \\
& [[\![f \;{}_p\!\diamond\; h]\!] \,|\, [\![g \;{}_p\!\diamond\; k]\!]] \\
= & \{ \;\; (53) \;\; \} \\
& [\![[f \;{}_p\!\diamond\; h, g \;{}_p\!\diamond\; k]]\!]
\end{array}
$$

Sums of probabilistic functions. In linear algebra, the *junc*/*split* combinators give rise to a (bi)functor known as *sum*,

$$M \oplus N \;=\; [i_1 \cdot M \,|\, i_2 \cdot N] \;=\; \left[\begin{array}{c} M \cdot \pi_1 \\ \hline N \cdot \pi_2 \end{array}\right] \;=\; \left[\begin{array}{c|c} M & 0 \\ \hline 0 & N \end{array}\right] \tag{54}$$

which has type:

$$A \xrightarrow{M} C \qquad B \xrightarrow{N} D \qquad A+B \xrightarrow{M \oplus N} C+D$$

Direct sum is a standard linear algebra operator enjoying many useful properties. The following equation, termed the *absorption law*,

$$[M \,|\, N] \cdot (P \oplus Q) = [M \cdot P \,|\, N \cdot Q] \tag{55}$$

specifies how block operator $[\,|\,]$ absorbs direct sum $\oplus$, for suitably typed matrices M, N, P, Q. Mind also property:

$$(M+N) \oplus (P+Q) = (M \oplus P) + (N \oplus Q) \tag{56}$$

As it is easy to show that matrix $[\![f]\!] \oplus [\![g]\!]$ is LS for f and g probabilistic, it makes sense to invent the combinator $f \oplus g$ [13] such that $[\![f \oplus g]\!] = [\![f]\!] \oplus [\![g]\!]$. One of the useful properties of such a combinator establishes distribution over choice on both sides. We prove this for the right hand side,

$$h \oplus (f \;{}_p\!\diamond\; g) = (h \oplus f) \;{}_p\!\diamond\; (h \oplus g) \tag{57}$$

[12] This equality arises from the universality of coproducts in the two categories, sets and matrices.

[13] Note the innocent overloading. Coproduct notation $f+g$ cannot be used, since the target type of this is a sum of distributions while that of $f \oplus g$ is a distribution of sums.

the other proof being similar:

$$
\begin{array}{ll}
& [\![h \oplus (f \,{}_p\!\diamond\, g)]\!] \\
= & \{ \text{ definition of } f \oplus g \text{ ; (44) } \} \\
& [\![h \,{}_p\!\diamond\, h]\!] \oplus [\![f \,{}_p\!\diamond\, g]\!] \\
= & \{ \text{ (43) twice } \} \\
& (p \otimes h + (1-p) \otimes h) \oplus (p \otimes f + (1-p) \otimes g) \\
= & \{ \text{ (56) } \} \\
& (p \otimes h \oplus p \otimes f) + ((1-p) \otimes h \oplus (1-p) \otimes g) \\
= & \{ \text{ } p \text{ and } 1-p \text{ are scalars } \} \\
& p \otimes (h \oplus f) + (1-p) \otimes (h \oplus g) \\
= & \{ \text{ (43) } \} \\
& [\![(h \oplus f) \,{}_p\!\diamond\, (h \oplus g)]\!]
\end{array}
$$

Probabilistic McCarthy conditional. As an extension of static choice, the probabilistic version of *if-then-else* can be introduced very much like in the AoP [BdM97], as a combinator denoted $p \rightarrow f, g$ and relying on coproducts, that is, on the *junc* combinator (19), and on so-called *guards*.

Given a predicate $Bool \xleftarrow{p} A$, the corresponding guard, denoted $p?$ and holding type $A + A \longleftarrow A$ is defined in the following way. Let $A \xrightarrow{[\![p]\!]} 1$ be the row-vector representation of the set where p holds true, as before. Coreflexive matrices $A \xleftarrow{\Phi_p, \Phi_{\neg p}} A$ defined by

$$
\Phi_p = [\![p]\!] \vartriangle id \tag{58}
$$

$$
\Phi_{\neg p} = [\![\neg p]\!] \vartriangle id \tag{59}
$$

provide diagonal representations of the corresponding vectors. The guard $p?$ associated to p is then defined by

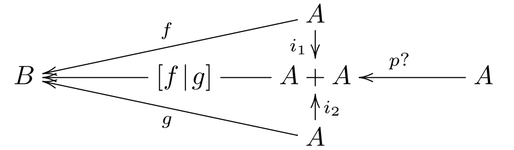
$$
A + A \xleftarrow{p?} A \;=\; \left[\frac{\Phi_p}{\Phi_{\neg p}}\right] \tag{60}
$$

Finally, we rely on (60) to define the probabilistic McCarthy conditional

$$
[\![p \rightarrow f, g]\!] = [[\![f]\!] \,|\, [\![g]\!]] \cdot p? \tag{61}
$$

where types are depicted aside. By *divide-and-conquer* (25) one has the alternative definition:

$$
[\![p \rightarrow f, g]\!] = [\![f]\!] \cdot \Phi_p + [\![g]\!] \cdot \Phi_{\neg p} \tag{62}
$$



Before proceeding, we first need to check that $p \rightarrow f, g$ is probabilistic ($\equiv [\![p \rightarrow f, g]\!]$ is LS) wherever f and g are so. For this we need to check that Φ_p and $\Phi_{\neg p}$ are complementary:

$$
\begin{array}{ll}
& \Phi_p + \Phi_{\neg p} \\
= & \{ \text{ (58,59) } \} \\
& ([\![p]\!] \vartriangle id) + ([\![\neg p]\!] \vartriangle id) \\
= & \{ \text{ bilinearity (41) } \} \\
& ([\![p]\!] + [\![\neg p]\!]) \vartriangle id
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \text{ vectors } [\![p]\!] \text{ and } [\![\neg p]\!] \text{ form a partition } \} \\
& ! \vartriangle id \\
= & \{ \text{ ! is the unit of Khatri-Rao (42) } \} \\
& id
\end{array}
$$

The above is enough to show properties such as eg. $p \to f, f = id$, just by applying (62), factoring $[\![f]\!]$ to the left by linearity (12) and simplifying the rest to $\Phi_p + \Phi_{\neg p} = id$. Rule (62) also makes the check that conditionals of probabilistic functions are probabilistic (5) almost immediate:

$$
\begin{array}{ll}
& ! \cdot [\![p \to f, g]\!] \\
= & \{ \text{ (62) ; linearity } \} \\
& ! \cdot [\![f]\!] \cdot \Phi_p^\circ + ! \cdot [\![g]\!] \cdot \Phi_{\neg p}^\circ \\
= & \{ \text{ (5) twice ; linearity } \} \\
& ! \cdot (\Phi_p + \Phi_{\neg p}) \\
= & \{ \ \Phi_p + \Phi_{\neg p} = id \ \} \\
& !
\end{array}
$$

It is easy to provide linear algebra calculations for the laws of guards and conditions found in eg. [Gib97]. For instance, the proof that the *swap* matrix of section 4 negates a guard, that is $(\neg p)? = swap \cdot (p?)$, becomes considerably simpler than in [Gib97]:

$$
\begin{array}{ll}
& swap \cdot (p?) \\
= & \{ \ swap = [i_2 \,|\, i_1] \text{ ; (60) } \} \\
& [i_2 \,|\, i_1] \cdot \begin{bmatrix} \Phi_p \\ \hline \Phi_{\neg p} \end{bmatrix} \\
= & \{ \text{ divide and conquer (25) } \} \\
& i_2 \cdot \Phi_p + i_1 \cdot \Phi_{\neg p} \\
= & \{ \text{ + is commutative } \} \\
& i_1 \cdot \Phi_{\neg p} + i_2 \cdot \Phi_p \\
= & \{ \text{ divide and conquer again } \} \\
& [i_1 \,|\, i_2] \cdot \begin{bmatrix} \Phi_{\neg p} \\ \hline \Phi_p \end{bmatrix} \\
= & \{ \text{ reflexion (21) ; (60) again } \} \\
& (\neg p)?
\end{array}
$$

Probabilistic extensions of the McCarthy conditional fusion laws of [BdM97, Gib97] arise from (61), for instance

$$
f \bullet (p \to g, h) = p \to f \bullet g, f \bullet h \tag{63}
$$

which follows from *junc*-fusion (23). However, the other fusion-law, $(p \to g, h) \cdot f = p \cdot f \to g \cdot f, h \cdot f$ uses composite predicate $p \cdot f$ which, in the category of sets, satisfies property

$$
p? \cdot f = (f + f) \cdot (p \cdot f)? \tag{64}
$$

The extension of this to probabilistic functions requires f to be sharp, that is, $[\![f]\!]$ should be a Boolean matrix. That (64) does not extend beyond functions has already been pointed out in [Gib97].

Splitting and pairing. Recall that the two functions in choice $f\ {}_p\!\diamond g$ have to be of the same type. Is there an extension of choice for probabilistic functions $A \xrightarrow{f} B$ and $A \xrightarrow{g} C$ differing on the output type? Note that, using *splits* (20), one can build matrix $A \xrightarrow{\left[\frac{[\![f]\!]}{[\![g]\!]}\right]} B + C$ but its columns will add up to 2 and so this matrix won't represent a probabilistic function: one has to control the choice of output between B and C. Similarly to choice, one may define a new combinator,

$$[\![f\ {}_p\nabla\ g]\!] = \left[\frac{p \otimes [\![f]\!]}{(1-p) \otimes [\![g]\!]}\right] \tag{65}$$

which combines f and g in a way which ensures a probabilistic outcome:

$$
\begin{array}{ll}
& ! \cdot [\![f\ {}_p\nabla\ g]\!] \\
= & \{\ \text{definition}\ \} \\
& ! \cdot \left[\frac{p \otimes [\![f]\!]}{(1-p) \otimes [\![g]\!]}\right] \\
= & \{\ [!\,|\,!] = !\ ;\ \text{dropping parentheses}\ \} \\
& [!\,|\,!] \cdot \left[\frac{p \otimes f}{(1-p) \otimes g}\right] \\
= & \{\ \text{divide and conquer (25)}\ \} \\
& ! \cdot (p \otimes f) + ! \cdot ((1-p) \otimes g) \\
= & \{\ \text{as calculated with choice}\ \} \\
& !
\end{array}
$$

To check the difference between this combinator and choice compare $swap\ {}_{0.9}\!\diamond\ id$ given earlier on with

$$swap\ {}_{0.9}\nabla\ id = \begin{pmatrix} 0 & 0.9 \\ 0.9 & 0 \\ 0.1 & 0 \\ 0 & 0.1 \end{pmatrix}$$

Note that the outputs of f and g are combined in $f\ {}_p\nabla\ g$ in alternation, either B or C. The effect of actually *pairing* the outputs of both functions has already been considered for vectors, recall (38). It can be easily extended column-wise to probabilistic functions sharing the same input type: given probabilistic $A \xrightarrow{f} B$ and $A \xrightarrow{g} C$, we define their *pairing* $A \xrightarrow{f \vartriangle g} B \times C$ by:

$$[\![f \vartriangle g]\!] = [\![f]\!] \vartriangle [\![g]\!] \tag{66}$$

Again note the intentional overloading, this time over the symbol for Khatri-Rao product. The proof that the outcome of (66) is probabilistic has already been given for $A = 1$, that is, for f and g distributions. This can be regarded the base case of an inductive proof whose inductive step consists of splitting $[\![f]\!] = [f_1\,|\,f_2]$ and $[\![g]\!] = [g_1\,|\,g_2]$ where f_1, g_1 (ibid. f_2, g_2) are of the same type:

$$
\begin{array}{ll}
& ! \cdot [\![f \vartriangle g]\!] \\
= & \{\ (66)\ \} \\
& ! \cdot ([\![f]\!] \vartriangle [\![g]\!]) \\
= & \{\ \text{split matrices column-wise; temporarily remove } [\![_]\!] \text{ brackets}\ \}
\end{array}
$$

$$
\begin{aligned}
& \; ! \cdot ([f_1 \,|\, f_2] \vartriangle [g_1 \,|\, g_2]) \\
= & \quad \{ \text{ exchange law (39) } \} \\
& \; ! \cdot [f_1 \vartriangle g_1 \,|\, f_2 \vartriangle g_2] \\
= & \quad \{ \text{ (23) } \} \\
& \; [! \cdot (f_1 \vartriangle g_1) \,|\, ! \cdot (f_2 \vartriangle g_2)] \\
= & \quad \{ \text{ induction step: } ! \cdot (f_i \vartriangle g_i) = \; ! \; (i = 1, 2) \} \\
& \; [! \,|\, !] \\
= & \quad \{ \; [! \,|\, !] = \; ! \; \} \\
& \; !
\end{aligned}
$$

In general, the matrix transform of the pairing combinator inherits the properties of the Khatri-Rao product. This is, however, *not* a categorial product. Restricted to Boolean matrices, ie. relations, this is known as the "fork" operator [Fri02] of fork (relation) algebra. Its generalization to arbitrary matrices is discussed in [Mac12].

5. Probabilistic folds

Having defined the probabilistic extension of the AoP-standard combinators for alternation, pairing, summing, etc. we are in position to address the construction of recursive probabilistic functions of a particular kind: they process inductive structures such as natural numbers or finite lists by replacing the standard constructors of these data types by probabilistic functions.

These constructs are known as *folds*, or *catamorphisms* [BdM97, GHA01]. The algorithm which multiplies two natural numbers a and b, for instance,

$$
\begin{aligned}
& a * 0 = 0 \\
& a * (n + 1) = a * n + a
\end{aligned}
$$

is an example of such a fold, defined by induction on the second argument. This can be seen by expressing the section $(a*)$ in the following way, where $\underline{k}$ denotes the constant function which delivers k irrespectively of its actual input (a polymorphic function) and $succ\ n = n + 1$:

$$
\begin{aligned}
& (a*) \cdot \underline{0} = \underline{0} \\
& (a*) \cdot succ = (a+) \cdot (a*)
\end{aligned}
$$

The symbol $_ \cdot _$ denotes standard function composition. Note how constructors 0 and *succ* (which form the Peano algebra of the natural numbers) are replaced by 0 (itself) and $(a+)$. For $a = 1$, $(a*) = id$ because $(1+) = succ$. Following the notation of [BdM97], one writes

$$(a*) = (\!|[\underline{0}, (a+)]|\!) \tag{67}$$

where the catamorphism parentheses $(\!|_|\!)$ denote the combinator which emerges from such constructor substitution. This is because the two equations above can be merged into a single one, $(a*) \cdot [\underline{0}, succ] = [\underline{0}, (a+) \cdot (a*)]$, by resorting to junction of functions (53), whose right hand side may be split into the argument of the combinator and the recursive call,

$$(a*) \cdot [\underline{0}, succ] = [\underline{0}, (a+)] \cdot (id + (a*)) \tag{68}$$

where the sum of two functions is the bifunctor which transforms into matrix sum:

$$[\![f + g]\!] = [\![f]\!] \oplus [\![g]\!] \tag{69}$$

Since all functions in the equation are sharp and the matrix transform preserves sharp functions ($! \cdot [\![f]\!] = !$), one may draw the diagram aside which depicts equation (68) expressed in terms of matrices. Note that in° is a function because in is an isomorphism. (The matrix which represents an isomorphism f is both left and right stochastic, that is, $! \cdot [\![f]\!] = !$ and $[\![f]\!] \cdot !^\circ = !^\circ$.) Also note column vector $[\![\underline{0}]\!]$ with as many rows as natural numbers, all filled with 0s but the first. (Recall that the matrix-transform of a function f is such that cells with coordinates $(f\ x, x)$ are filled with 1s and all the others with 0s.)

$$\begin{array}{ccc} \mathbb{N}_0 & \xrightarrow{in^\circ = \left[\frac{[\![\underline{0}]\!]^\circ}{[\![succ]\!]^\circ}\right]} \atop \xleftarrow[in = [[\![\underline{0}]\!] \,|\, [\![succ]\!]]]{\cong} & 1 + \mathbb{N}_0 \\ {\scriptstyle [\![(a*)]\!]} \downarrow & & \downarrow {\scriptstyle id \oplus [\![(a*)]\!]} \\ \mathbb{N}_0 & \xleftarrow[[[\![\underline{0}]\!] \,|\, [\![(a+)]\!]]]{} & 1 + \mathbb{N}_0 \end{array}$$

Reading the diagram, one has:

$$[\![(a*)]\!] = [[\![\underline{0}]\!] \,|\, [\![(a+)]\!]] \cdot (id \oplus [\![(a*)]\!] \cdot \left[\frac{[\![\underline{0}]\!]^\circ}{[\![succ]\!]^\circ}\right]$$

$$= \quad \{ \text{ absorption (55) ; dropping parentheses for better parsing } \}$$

$$(a*) = [\underline{0} \,|\, (a+) \cdot (a*)] \cdot \left[\frac{\underline{0}^\circ}{succ^\circ}\right]$$

$$= \quad \{ \text{ divide and conquer (25) } \}$$

$$(a*) = \underline{0} \cdot \underline{0}^\circ + (a+) \cdot (a*) \cdot succ^\circ$$

This tells how the matrix for $(a*)$ is recursively filled up: first the outer-product of $\underline{0}$ by itself ($\underline{0} \cdot \underline{0}^\circ$, that is the everywhere-0 matrix apart from the 1 in cell $(0, 0)$), which is added to $(a+) \cdot \underline{0} \cdot \underline{0}^\circ \cdot succ^\circ = \underline{a} \cdot \underline{1}^\circ$ (matrix with a 1 in cell labeled $(a, 1)$ and 0 otherwise), and so on. Note that each contribution of the fixpoint is a matrix which "fills an empty column", thus ensuring that no column ever adds up to more than 1.

Let us now inject a fault into $(a*)$, as we did in Section 1 for the factorial function:

$$\begin{array}{l} a * 0 =_p 0 \\ a * 0 =_{1-p} a \\ a * (n + 1) = a * n + a \end{array}$$

That is, with probability $1 - p$ the base case erroneously yields a as output instead of 0. Can we *measure* the impact of this base-case fault onto the whole algorithm? Intuitively, one may guess that, with the same probability $1 - p$, the function will compute $a * (b + 1)$ rather than $a * b$. That is, the base case's fault propagates to the whole algorithm in the same (quantitative) manner.

Is this intuition checkable? Below we will confirm the guess by calculation. For this purpose, we define three versions of the algorithm, the one which calculates $a * b$ as before (named *good*), the erroneous one (named *bad*) and the faulty one (named *faulty*) where the fault is expressed using choice (43):

$$\begin{array}{r} good = (\![[\underline{0} \,|\, (a+)]]\!) \\ bad = (\![[\underline{a} \,|\, (a+)]]\!) \\ faulty = (\![[\underline{0} \ {}_p\!\diamond\ \underline{a} \,|\, (a+)]]\!) \end{array}$$

The assertion we want to check is $faulty = good \ {}_p\!\diamond\ bad$. We reason:

$$faulty = good \ {}_p\!\diamond\ bad$$

$$\equiv \quad \{ \text{ definition } \}$$

$$(\![[\underline{0} \ {}_p\!\diamond\ \underline{a} \,|\, (a+)]]\!) = good \ {}_p\!\diamond\ bad$$

$$\equiv \quad \{ \text{ universal property of catamorphisms, for } \mathsf{F}X = id \oplus X \ \}$$

$$(good \ {}_p\!\diamond\ bad) \cdot in = [\underline{0} \ {}_p\!\diamond\ \underline{a} \,|\, (a+)] \cdot \mathsf{F}(good \ {}_p\!\diamond\ bad)$$

The calculation of this equality resorts to standard properties of catamorphisms [BdM97]:

$$
\begin{aligned}
& (good\ {}_p\diamond\ bad) \cdot in \\
= & \quad \{ \ \text{fusion (49)} \ \} \\
& (good \cdot in)\ {}_p\diamond\ (bad \cdot in) \\
= & \quad \{ \ \text{catamorphism cancellation (twice)} \ \} \\
& ([\underline{0} \,|\, (a+)] \cdot \mathsf{F} good)\ {}_p\diamond\ ([\underline{a} \,|\, (a+)] \cdot \mathsf{F} bad) \\
= & \quad \{ \ \text{absorption (55) over } \mathsf{F}X = id \oplus X \ \} \\
& [\underline{0} \,|\, (a+) \cdot good]\ {}_p\diamond\ [\underline{a} \,|\, (a+) \cdot bad] \\
= & \quad \{ \ \text{distribution (52)} \ \} \\
& [\underline{0}\ {}_p\diamond\ \underline{a} \,|\, ((a+) \cdot good)\ {}_p\diamond\ ((a+) \cdot bad)] \\
= & \quad \{ \ \text{fusion (50)} \ \} \\
& [\underline{0}\ {}_p\diamond\ \underline{a} \,|\, (a+) \cdot (good\ {}_p\diamond\ bad)] \\
= & \quad \{ \ \text{absorption (55), in the reverse direction} \ \} \\
& [\underline{0}\ {}_p\diamond\ \underline{a} \,|\, (a+)] \cdot \mathsf{F}(good\ {}_p\diamond\ bad)
\end{aligned}
$$

Note how (52) plays the central role above, thanks to bilinearity on the background. The example is simple but illustrative of the way matrix calculations proceed, handling matrices with no further effort as compared to handling relations as in standard AoP. This is so because of the sheer amount of structure which categories of matrices and relations have in common.

Probabilistic fold fusion. As a second example, consider the following situation: we want to compute the expression $1 + a * b$ knowing that $(1+)$ is performed by a faulty successor function which, with probability q, fails to increment its input,

$$fsucc = id\ {}_q\diamond\ succ$$

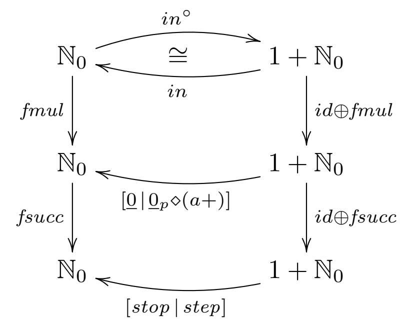
and another version of faulty multiplication,

$$fmul = (\!|[\underline{0} \,|\, \underline{0}\ {}_p\diamond\ (a+)]|\!) \tag{70}$$

which fails to perform the addition (yielding unit 0) with probability p. (Value a is assumed fixed to avoid extra parameters which add nothing to the problem.)

Unlike the previous example, this time the fault disturbs the inductive step and the exercise consists in predicting the behaviour of program

$$fprog = fsucc \cdot fmul \tag{71}$$



The diagram aside illustrates this situation. Parentheses $[\![\ ]\!]$ are omitted altogether this time, assuming the reader's familiarity with the previous diagram at this point (in is the same isomorphism in both diagrams). Thus all probabilistic functions are assumed implicitly represented by their matrix transform.

The diagram helps to understand the strategy to follow: we will try and fuse *fsucc* with *fmul* (71) using fold (catamorphism) *fusion* [BdM97]. The outcome will be $fprog = (\!|[stop \,|\, step]|\!)$ provided the lower rectangle commutes:

$$fsucc \cdot [\underline{0} \,|\, \underline{0}\ {}_p\diamond\ (a+)] = [stop \,|\, step] \cdot (id \oplus fsucc)$$

Solving this equality for the unknowns *stop* and *step* yields $stop = fsucc \cdot \underline{0} = (id\ {}_q\diamond\ succ) \cdot \underline{0}$, that is

$$stop = \underline{0}\ {}_q\diamond\ \underline{1}$$

thanks to choice-fusion (49). Concerning *step*:

$$fsucc \cdot (\underline{0}\ {}_p\!\diamond\ (a+)) \;=\; step \cdot fsucc$$

$\equiv$ { choice-fusion (50) ; $fsucc \cdot \underline{0} = stop$ }

$$stop\ {}_p\!\diamond\ (fsucc \cdot (a+)) \;=\; step \cdot fsucc$$

$\equiv$ { *fsucc* commutes with $(a+)$ since *succ* commutes with $(a+)$ }

$$stop\ {}_p\!\diamond\ ((a+) \cdot fsucc) \;=\; step \cdot fsucc$$

$\equiv$ { *stop* is (probabilistically) constant, thus $stop \cdot f = stop$ for any f ; (49) }

$$(stop\ {}_p\!\diamond\ (a+)) \cdot fsucc \;=\; step \cdot fsucc$$

$\Leftarrow$ { Leibniz }

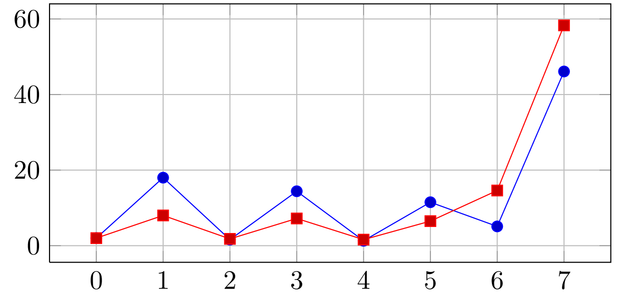
$$step = stop\ {}_p\!\diamond\ (a+)$$

In summary, equality (71) can be rewritten into

$$fsucc \cdot fmul = (\![stop \mid stop\ {}_p\!\diamond\ (a+)]\!) \quad , \quad \text{for } stop = \underline{0}\ {}_q\!\diamond\ \underline{1}$$

expressing the combined impact of the faults of the two functions: *fsucc* disturbs both the base case and the inductive step of *fmul* through *stop*, which can be regarded as a faulty constant (0 with probability q, 1 otherwise).

The encoding of both sides of the equality above in Haskell, relying on the distribution monad available from the PFP library, yields two programs which are input-output equivalent, as expected. The figure aside shows two distributions for the evaluation of expression $1 + 2 \times 3$, that is, for $a = 2$ and input 3. The two distributions (consistently produced by both programs) correspond to different probabilities in the faults injected: $p = 20\%, q = 10\%$ in the plot marked with bullets and $p = 10\%, q = 20\%$ in the other. The probability of yielding the correct output $(1 + 2 \times 3 = 7)$ is 46% in the first case and 58% in the second.



6. Related work

There is a trend towards *quantitative formal methods*. Quoting the Preface of [AMD+09]:

Quantitative Formal Methods deals with systems whose behaviour of interest is more than the traditional Boolean “correct” or “incorrect” judgment. (...) The aim of the workshop was to create a new forum where current and novel theories and application areas of quantitative methods could be discussed, together with the verification techniques that might apply to them.

Among such methods, probabilistic techniques are becoming more and more widespread. McIver and Morgan [MM05] develop a method for rigorous reasoning about probabilistic programs that includes a probabilistic calculus which, in the Hoare style, operates at the level of the program text. At programming level, reference [EK06] describes an approach to express probabilistic programs in Haskell and give a collection of modules that make up a probabilistic functional programming library based on the (finite) distribution monad. A similar monadic flavor can be found in the approach to quantum (functional) programming given in [MB01]. More recently, Gibbons and Hinze [GH11] have shown how to perform equational reasoning about programs that exploit both nondeterministic and probabilistic choice as part of a more ambitious plan to reason about effectful computations in general.

Sokolova [Sok05] develops a coalgebraic analysis of probabilistic systems in a way that connects two mainstream research areas: coalgebraic reasoning and probabilistic modeling and verification. This work builds upon foundational work on probabilistic bisimulation [LS91] and relates to quantitative Kleene coalgebra

[SBBR11]. Reference [MCM06] describes pKA, a probabilistic Kleene-style algebra based on a well known model of probabilistic/demonic computation.

Focusing on quantum programming and quantitative denotational semantics of probabilistic programs, the authors of [SRM08] adopt linear algebra techniques by regarding probabilistic programs as linear transformations over suitable vector spaces. This can be framed into another trend, that of using linear algebra techniques in computer science. In this trend, Trčka [Trc09] presents a unifying matrix approach to the notions of strong, weak and branching bisimulation ranging from labeled transition systems to Markov reward chains. Transition systems are triples made of an initial vector, a transition matrix and a termination vector. The technique developed in [DLS10] for evaluating high-availability standby redundant clusters resorts to Kronecker sums and products. An illustration of the method is given for the 1-plus-2 HA cluster. Finally, natural language semantics, too, is going vectorial, as nicely captured by the aphorism *nouns are vectors, adjectives are matrices* [BZ10]. In this field of "quantum linguistics", a compositional model of meaning is given in [CSC10] in which the grammatical structure of sentences is expressed in the category of finite dimensional vector spaces.

Categories of matrices can be traced back to [Mac98, MB99] with focus on either illustrating additive categories or establishing a relationship between linear transformations and matrices. In the area of process semantics, Bloom *et al* [BSW96] have developed a categorical, *machines as matrices* theory of concurrency. Work in this vein can be traced much earlier, back to Conway's work on *regular algebras* [Con71] and regular algebras of matrices, so elegantly addressed in [Bac04]. On the footsteps of this work, reference [MO11b] suggests a linear algebra approach to software components, be these weighted automata [SBBR11] or machines in the sense of [BSW96], using the "'matrices as arrows" typed approach. As illustration of this categorial approach to linear algebra, but in a different domain, reference [MO11a] shows how to implement data mining operations solely based on linear algebra operations. The Khatri-Rao product and its unit ! (recall section 4) play a major role in data consolidation constructions such as data cubes and pivot tables.

7. Conclusions and future work

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, unrelatedly and 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 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 will come as surprise to many people — 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"* [AM05] 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 [Sch10] is a big step forward towards conceptual unification between relations and matrices. But it is first and foremost category theory 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.

The category of relations, extended to an allegory in [FS90] and including that of functions as a subcategory, is the basis of the algebra of programming (AoP) which has reached the programming community through textbook [BdM97]. It shares a lot in common with any category of matrices (one cannot say "the" category of matrices because there are many, one per cell-type), namely self-duality and the existence of *biproducts* [Mac98] which explain how the cell-level algebra lifts to "blocks", scaling up notation in a very successful way.

The research question which motivates the work reported in the current paper splits in two other questions, in fact two sides of the same coin: (a) can the AoP be extended quantitatively in some useful way? (b) what happens to the discipline once we generalize from relations to matrices?

The answer leads us into linear algebra, which eventually provides a surprisingly simple framework for calculating with set-theory, probabilities, functions and relations, provided it is *typed* — as shown in [MO10]

— in the same way as the AoP. (Bi)Linearity, which is after all what elsewhere (eg. in Kleene algebra) is known as distribution of sequencing over branching, is central to the reasoning. So the acronym LAoP, for "linear algebra of programming" makes sense. Interestingly enough, it has been put forward already, albeit in a somewhat different setting, by Sernadas *et al* [SRM08], the key idea being *"to adopt linear algebra as the lingua franca of software verification"* [SG11].

Restricting to discrete probabilities spaces, the current paper applies the emerging LAoP to probabilistic functions, which are half way between relations and functions: they express the propensity, or likelihood of ambiguous, multiple outputs. The approach is based on the *matrix transform* which establishes an isomorphism between the sub-category of such functions and that of left-stochastic matrices. The reasoning in such transformed space is carried out in the same style as in the AoP, saving the practitioner from the intricacies of the distribution monad. Our examples of typed LA calculation range from elementary probability theory to calculations about recursive functions injected with random faults. However not very elaborate, these examples show the potential of LAoP for reasoning about quantitative aspects of probabilistic programming.

Future work. The application of linear algebra techniques to quantitative reasoning about recursive programs appears to be a promising area of research which can benefit much from that immense body of knowledge which linear algebra is. It will be interesting, for instance, to prove facts about programs written in the PFP library of [EK06] and to relate LAoP calculation to the techniques of [GH11].

Probabilistic functions can be regarded as refinements of relations. The abstraction of a probabilistic function f into a relation is given by $[\![f]\!]^{\uparrow}$, where $_^{\uparrow}$ is the operator which converts all non-zero entries in $[\![f]\!]$ to 1s, thus mapping $[\![f]\!]$ into a Boolean matrix (relation) [14]. Probabilistic refinement should be studied in this setting.

Take as example the definition of the less-or-equal ordering on the natural numbers as a relational fold: $(\leq) = (\![\underline{0}, \underline{0} \cup succ]\!)$. Read algorithmically, $(\leq)$ yields (in a non-deterministic way) any number at most a given number. A refinement step could replace union by choice, leading into the probabilistic implementation f such that $[\![f]\!] = (\![\underline{0}, \underline{0}\ {}_p\!\diamond succ]\!)$, for some p. For $p = 0$, $f\ n = n$; for $p = 1$, $f\ n = 0$. For all other values of p, $f\ n$ will generate (randomly) a number between 0 and n. What can one say about such a random number generator? What contribution can the LAoP give in this respect? Broadening scope, one should check the applicability of the algebra to recursion schemes more general and powerful than probabilistic folds, eg. unfolds (anamorphisms) and hylomorphisms [BdM97].

Note that probabilistic anamorphisms have been implicitly studied in the field of probabilistic coalgebras [Sok05], but not from a LAoP perspective. Work in this direction, in particular in extending the distribution functor to relations, has been drafted in [Oli11]. This should eventually link to the bisimulations for Markov reward chains of [Trc09].

Finally, knowing that relations can be generalised to semirings and Kleene algebra, it would be interesting to identify which similar algebraic structures generalize probabilistic functions. Such structures are likely to be related to pKA [MCM06] and to the tree Kleene algebras studied in [TF06].

Summing up, this paper opens a research direction which calls for a feasibility study, in one direction, and for further theoretical developments in another. This will keep researchers interested in the LAoP busy for a while.

Acknowledgements

The author is indebted to the anonymous referees for detailed and useful comments and suggestions which helped to improve this paper. In particular, the suggestion of axiom (15) to characterize the Boolean matrices of $Mat_{\mathbb{R}}$ is gratefully acknowledged. This research was carried out in the context of the Mondrian 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/108302/2008`.

[14] See [Win09]. In a sense, the columns of $[\![f]\!]^{\uparrow}$ are the vectors which capture the *support* of each output distribution of f. Operator $R^{\uparrow}$ and its adjoint $R^{\downarrow}$ (which converts non-zero entries into 0s) are proposed in [Win09] where defining so-called Goguen categories, which relate to Zadeh categories formed by fuzzy relations [KFM99].

References

[AM05] K.M. Abadir and J.R. Magnus. *Matrix algebra. Econometric exercises 1*. Cambridge University Press, 2005.

[AMD+09] S. Andova, A. McIver, P. R. D'Argenio, P. J. L. Cuijpers, J. Markovski, C. Morgan, and M. Núñez, editors. *Proceedings First Workshop on Quantitative Formal Methods: Theory and Applications*, volume 13 of *EPTCS*, 2009.

[Bac04] R.C. Backhouse. *Mathematics of Program Construction*. Univ. of Nottingham, 2004. Draft of book in preparation. 608 pages.

[BdM97] R. Bird and O. de Moor. *Algebra of Programming*. Series in Computer Science. Prentice-Hall International, 1997.

[BSW96] S.L. Bloom, N. Sabadini, and R.F.C. Walters. Matrices, machines and behaviors. *Applied Categorical Structures*, 4(4):343–360, 1996.

[BZ10] M. Baroni and R. Zamparelli. Nouns are vectors, adjectives are matrices: representing adjective-noun constructions in semantic space. In *Proceedings*, EMNLP '10, pages 1183–1193, Morristown, NJ, USA, 2010. Association for Computational Linguistics.

[Con71] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[CSC10] B. Coecke, M. Sadrzadeh, and S. Clark. Mathematical foundations for a compositional distributed model of meaning. *Linguistic Analysis*, 36(1-4):345–384, 2010.

[DLS10] S. Distefano, F. Longo, and M. Scarpa. Availability assessment of HA standby redundant clusters, 2010. 29th IEEE Int. Symp. on Reliable Distributed Systems.

[EK06] M. Erwig and S. Kollmansberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16:21–34, January 2006.

[Fri02] M.F. Frias. Fork algebras in algebra, logic and computer science, 2002. Logic and Computer Science. World Scientific Publishing Co.

[FS90] P.J. Freyd and A. Scedrov. *Categories, Allegories*, volume 39 of *Mathematical Library*. North-Holland, 1990.

[GH11] J. Gibbons and R. Hinze. Just do it: simple monadic equational reasoning. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, ICFP'11, pages 2–14, New York, NY, USA, 2011. ACM.

[GHA01] J. Gibbons, G. Hutton, and T. Altenkirch. When is a function a fold or an unfold? *Electronic Notes in Theoretical Computer Science*, 44(1), 2001.

[Gib97] J. Gibbons. Conditionals in distributive categories. Technical Report CMS-TR-97-01, School of Computing and Mathematical Sciences, Oxford Brookes University, January 1997.

[Heh11] E. Hehner. A probability perspective. *Formal Aspects of Computing*, 23:391–419, 2011.

[Hoo97] P. Hoogendijk. *A Generic Theory of Data Types*. PhD thesis, Univ. of Eindhoven, The Netherlands, 1997.

[KFM99] Y. Kawahara, H. Furusawa, and M. Mori. Categorical representation theorems of fuzzy relations. *Information Sciences*, 119(3–4):235–251, 1999.

[KS76] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer-Verlag, 1976. Originally printed by Van Nostrand, Princeton, 1960.

[LR99] G. Latouche and V. Ramaswami. *Introduction to Matrix Analytic Methods in Stochastic Modeling*. ASA-SIAM, 1999.

[LS91] K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

[LS97] B. Lawvere and S. Schanuel. *Conceptual Mathematics: a First Introduction to Categories*. Cambridge University Press, 1997.

[Mac98] S. MacLane. *Categories for the Working Mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer, September 1998.

[Mac12] H. Macedo. *Matrices as Arrows — Why Categories of Matrices Matter*. PhD thesis, University of Minho, 2012. (Submitted Jan. 2012).

[Mad91] R.D. Maddux. The origin of relation algebras in the development and axiomatization of the calculus of relations. *Studia Logica*, 50:421–455, 1991.

[MB99] S. MacLane and G. Birkhoff. *Algebra*. AMS Chelsea, 1999.

[MB01] S.C. Mu and R. Bird. Quantum functional programming, 2001. 2nd Asian Workshop on Programming Languages and Systems , KAIST, Dajeaon, Korea, December 17-18, 2001.

[MCM06] A. McIver, E. Cohen, and C. Morgan. Using probabilistic Kleene algebra for protocol verification. In Renate Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *LNCS*, pages 296–310. Springer Berlin / Heidelberg, 2006.

[MM05] A. McIver and C. Morgan. *Abstraction, Refinement And Proof For Probabilistic Systems*. Monographs in Computer Science. Springer-Verlag, 2005.

[MO10] H.D. Macedo and J.N. Oliveira. Matrices As Arrows! A Biproduct Approach to Typed Linear Algebra. In *MPC'10*, volume 6120 of *LNCS*, pages 271–287. Springer, 2010.

[MO11a] H.D. Macedo and J.N. Oliveira. Do the middle letters of "OLAP" stand for linear algebra ("LA")? Technical Report TR-HASLab:04:2011, INESC TEC and University of Minho, Gualtar Campus, Braga, 2011.

[MO11b] H.D. Macedo and J.N. Oliveira. Towards linear algebras of components. In *FACS 2010*, volume 6921 of *LNCS*, pages 300–303. Springer, 2011.

[Mor90] C. Morgan. *Programming from Specification*. Series in Computer Science. Prentice-Hall International, 1990. C.A.R. Hoare, series editor.

[Oli11] J.N. Oliveira. A look at the (linear) algebra of probabilistic functions, 2011. Contribution to the QAIS Start-up Workshop celebrating IBM-Portugal Scientific Prize 2010, Braga, 17th October 2011.

[Pra92] V. Pratt. Origins of the calculus of binary relations. In *Proc. of the 7th Annual IEEE Symp. on Logic in Computer Science*, pages 248–254, Santa Cruz, CA, 1992. IEEE Comp. Soc.

[Ros09] J. Rosenhouse. *The Monty Hall Problem: The Remarkable Story of Math's Most Contentious Brain Teaser*. Oxford University Press, 2009.

[RR98] C.R. Rao and M.B. Rao. *Matrix algebra and its applications to statistics and econometrics*. World Scientific Pub Co Inc, 1998.

[SBBR11] A. Silva, F. Bonchi, M.M. Bonsangue, and J.J.M.M. Rutten. Quantitative Kleene coalgebras. *Inf. Comput.*, 209(5):822–849, 2011.

[Sch10] G. Schmidt. *Relational Mathematics*. Number 132 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, November 2010.

[SG11] SQIG-Group. LAP: Linear algebra of bounded resources programs, 2011. IT & Tech. Univ. Lisbon.

[Sok05] A. Sokolova. *Coalgebraic Analysis of Probabilistic Systems*. Ph.D. dissertation, Tech. Univ. Eindhoven, Eindhoven, The Netherlands, 2005.

[SRM08] A. Sernadas, J. Ramos, and P. Mateus. Linear algebra techniques for deciding the correctness of probabilistic programs with bounded resources. Technical report, SQIG - IT and IST - TU Lisbon, 1049-001 Lisboa, Portugal, 2008. Short paper presented at LPAR 2008, Doha, Qatar. November 22-27.

[TF06] T. Takai and H. Furusawa. Monodic tree Kleene algebra. In Renate Schmidt, editor, *Relations and Kleene Algebra in Computer Science*, volume 4136 of *Lecture Notes in Computer Science*, pages 402–416. Springer Berlin / Heidelberg, 2006.

[Trc09] N. Trcka. Strong, weak and branching bisimulation for transition systems and Markov reward chains: A unifying matrix approach. *Proc. First Workshop on Quantitative Formal Methods: Theory and Applications*, volume 13 of *EPTCS*, 2009, pages 55–65.

[VM97] J. Voas and G. McGraw. *Software Fault Injection: Innoculating Programs Against Errors*. John Wiley & Sons, 1997. 416 pages.

[Win09] M. Winter. Arrow categories. *Fuzzy Sets and Systems*, 160(20):2893–2909, 2009.