Algebraic and Coalgebraic Methods in Software Development

J.N. Oliveira (UM)

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Foundations of Computing
2016-17 (updated: 2017/18)

1st Module: Basic category theory for the software sciences

Questions

Software is pre-science — **formal** but not fully **calculational**

Software is too diverse — many approaches, lack of unity

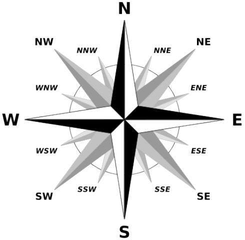
Software is too wide a concept — from assembly to quantum programming

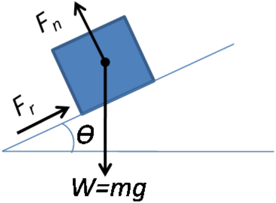
Can you think of a **unified** theory able to express and reason about software *in general*?

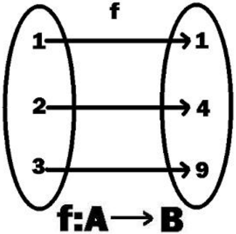
Put in another way:

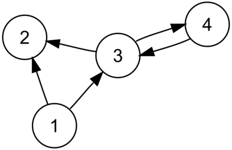
Is there a "lingua franca" for the software sciences?

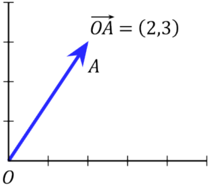
Check the pictures...



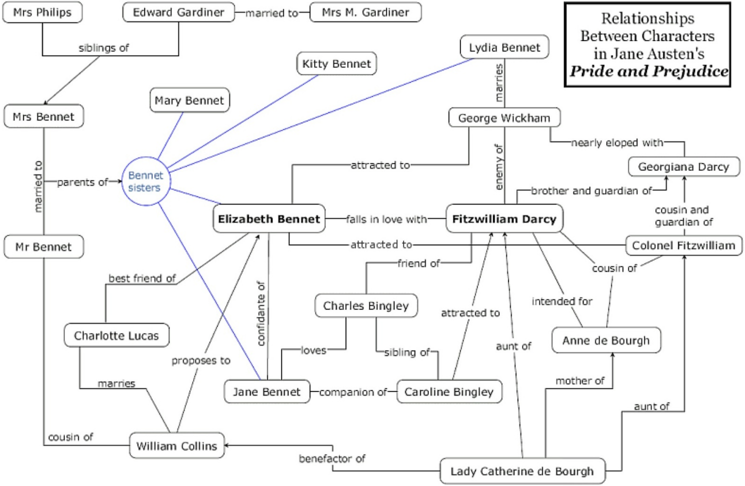






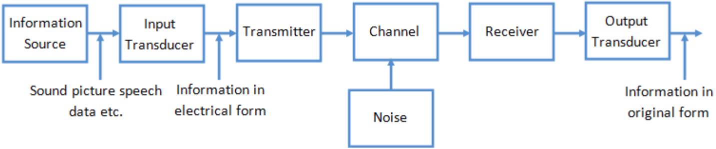


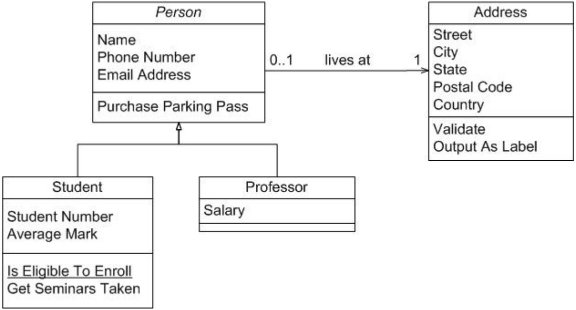
Check the pictures



(Wikipedia: **Pride and Prejudice**, by Jane Austin, 1813.)

Check the pictures





Check the pictures

Which **graphical** device have you found **common** to **all** pictures?

Your answer is likely to match what comes next...

Arrows everywhere

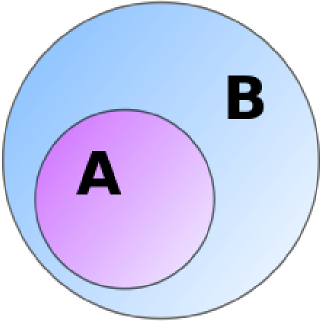
Arrows! Thus we identify a (graphical) ingredient **common** to describing (several) **different** fields of human activity.

For this ingredient to be able to support a **generic** theory of systems, mind the remarks:

- We need a **generic** notation able to cope with very distinct problem domains, e.g. **process** theory versus **database** theory, for instance.
- Notation is not enough — we need to **reason** and **calculate** about software.
- Semantics-rich **diagram** representations are welcome.
- System description may have a **quantitative** side too.

Back to basics

Recall your basic school maths. In set theory, for instance,



you wrote $A \subseteq B$ meaning to say that A *is a subset of* B.

Back to basics

Quite often one also uses **arrows**

$$
\begin{tikzcd}
B \\
A \arrow[u]
\end{tikzcd}
$$

to say the same thing, $A \subseteq B$.

Graphical notations (Venn diagrams, arrow notation) are useful.

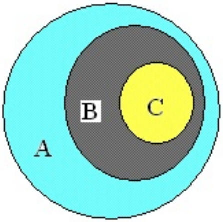
Take, for instance,

$A \subseteq A$ *holds (reflexivity)*
$C \subseteq B$ *and* $B \subseteq A$ *then* $C \subseteq A$ *holds (transitivity)*
$A \subseteq B$ *and* $B \subseteq A$ *then* $B = A$ *(anti-symmetry)*

Back to basics

Diagram for the **transitive** property:

$$\begin{tikzcd}
A & \\
& B \arrow[ul] \\
& C \arrow[u] \arrow[uul]
\end{tikzcd}$$



Diagrams for the other two properties:

$$\begin{tikzcd}
A \arrow[loop above]
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
A \arrow[r, bend left] \arrow[r, phantom, "="] & B \arrow[l, bend left]
\end{tikzcd}$$

Back to basics

Divisibility — write $n \sqsubseteq m$ to say that n *divides* m (in $\mathbb{N}$).

Natural number divisibility basic facts:

> $n \sqsubseteq n$ *holds (reflexivity)*
> $n \sqsubseteq m$ *and* $m \sqsubseteq k$ *then* $n \sqsubseteq k$ *holds (transitivity)*
> $n \sqsubseteq m$ *and* $m \sqsubseteq n$ *then* $m = n$ *(anti-symmetry)*

Again we may use **arrows** and **diagrams** to say the same thing, e.g.

$$\begin{tikzcd} k & m \arrow[l] \\ & n \arrow[ul] \arrow[u] \end{tikzcd}$$

to mean the middle property (and so on).

Back to basics

Statement $2 \sqsubseteq 6$ is valid but but it provides **no evidence** about **why** such a relationship holds.

We argue:

- $3 \cdot 2 = 6$;
- that is, $\exists\, k = 3$ such that $k \cdot 2 = 6$;
- that is, 6 is a **multiple** of 2.

In general,

$$n \sqsubseteq m \quad \text{iff} \quad \exists\, k \text{ st } m = k \cdot n$$

Why *so much ado for so little*? How about drawing

$$\begin{tikzcd} m \\ n \arrow[u, "k"'] \end{tikzcd}$$

to mean the same? Take k as the **witness** — evidence, **proof** — of the divisibility **relationship**.

Back to basics

This helps in providing evidence of the properties themselves by calculating **new** witnesses from **given** witnesses.

Such is the case of **transitivity**

$$\begin{tikzcd} k & m \arrow[l, "b"'] \\ & n \arrow[u, "a"'] \arrow[ul, "b \cdot a"] \end{tikzcd}$$

and **reflexivity**:

$$\begin{tikzcd} n \\ n \arrow[u, "1"'] \end{tikzcd}$$

Moreover:

$$\begin{tikzcd} n \\ 1 \arrow[u, "n"'] \end{tikzcd}$$

since 1 divides any number (etc).

A graphical, **constructive** way of stating divisibility properties.

Back to basics

Thus two well-known properties of multiplication, **associativity**

$$c \cdot (b \cdot a) = (c \cdot b) \cdot a \qquad (1)$$

and **identity**

$$1 \cdot a = a \cdot 1 = a \qquad (2)$$

are depicted aside in diagrammatic form.

$$\begin{tikzcd}
q & \\
k \arrow[u, "c"'] & m \arrow[l, "b"'] \arrow[ul, "c\cdot b"'] \\
& n \arrow[u, "a"'] \arrow[ul, "b\cdot a"]
\end{tikzcd}$$

$$\begin{tikzcd}
m & m \arrow[l, "1"'] \\
n \arrow[u, "a"] & n \arrow[l, "1"] \arrow[u, "a"'] \arrow[ul, "a"']
\end{tikzcd}$$

Note how the arrows **type** numbers with other numbers.

Why is this relevant?

We shall say that numbers depicted in this way, as **arrows** (between other numbers in this case) satisfying properties (1) and (2), form a **category**.

Again, "much ado for nothing"? Wait and see — the concept of a **category** *will prove very powerful and generic.*

Another way to put it, more computer science oriented:

Arrow $m \xleftarrow{k} n$ *means that number* k *has become* **typed** *by an* **input** *type* n *and an* **output** *type* m *(all natural numbers).*

Types play a **major** role in scientific software engineering, as we shall see.

Another category of numbers

This example of a category is "boring" — there are natural numbers everywhere, both labelling the arrows and their endpoints.

Is there any other construction in mathematics or computer science which we could describe by arrows

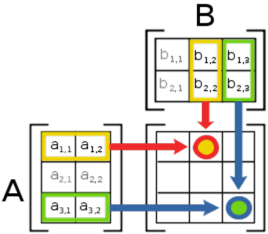
$$m \xleftarrow{k} n$$

where n, m are still natural numbers, but k is not one such number?

Yes — the Wikipedia describes one (click this link) as shown in the next slide.

Matrices as arrows

Recall matrix multiplication:



Index-wise definition

$$C_{ij} = \sum_{k=1}^{2} A_{ik} \times B_{kj}$$

The same in **arrow** notation

$$\begin{tikzcd} 3 & 2 \arrow[l, "A"'] & 3 \arrow[l, "B"'] \arrow[ll, bend left, "C=A\cdot B"] \end{tikzcd}$$

Index-free notation

$$C = A \cdot B$$

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Matrices as arrows

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Given

$$A = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}_{m \times n} \qquad m \xleftarrow{A} n$$

$$B = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}_{n \times k} \qquad n \xleftarrow{B} k$$

define

$$\begin{tikzcd} m & n \arrow[l, "A"'] & k \arrow[l, "B"'] \arrow[ll, bend left, "A \cdot B"] \end{tikzcd}$$

as matrix A multiplied by matrix B.

Matrices as arrows

As is well-known, matrix multiplication is **associative**,

$$C \cdot (B \cdot A) = (C \cdot B) \cdot A \quad (3)$$

with **identity**

$$id \cdot A = A \cdot id = A \qquad (4)$$

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

Each identity matrix $n \xleftarrow{id} n$ is the **diagonal** of size n, that is, $id_{j,i} \triangleq j = i$ under the $(0,1)$ encoding of the Booleans (aside).

Matrices as arrows

Summary:

- Matrices form a **category** whose **objects** are matrix dimensions and whose **arrows** $m \xleftarrow{A} n$, $n \xleftarrow{B} k$ are the matrices themselves.
- **Composition** $A \cdot B$ is matrix-multiplication.
- Each arrow $m \xleftarrow{A} n$ tells that matrix A has n-columns and m-rows.
- We say n is the **input** type of A and m the **output** type.
- Every **identity** $n \xleftarrow{id} n$ is the 1-diagonal of size n.
- Arrows (matrices) of types $n \xleftarrow{A} 1$ and $1 \xleftarrow{A} n$ are known as (respectively) column and row **vectors**.

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

What a category is

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

A category $\mathbb{C}$ is a mathematical structure made of **arrows** between **objects** (the end-points of arrows) where

- The set of arrows between two objects m and n is denoted by $\mathbb{C}\,(m, n)$.
- Writing $n \xleftarrow{a} m$, $m \xrightarrow{a} n$ or $a \in \mathbb{C}\,(m, n)$ means the same.
- Given arbitrary arrows $b \in \mathbb{C}\,(k, n)$ and $a \in \mathbb{C}\,(m, k)$, the composite arrow $b \cdot a$ always exists and belongs to $\mathbb{C}\,(m, n)$.
- The **identity** arrow $n \xrightarrow{id} n$ always exists, for each object n.
- Composition is associative (1) with id as unit (2).

Arrows are often called **morphisms**. $\mathbb{C}\,(m, n)$ are termed **homsets**.

Three categories thus far

Category	*Objects*	*Arrows*	*Composition*
$\mathbb{N}$	*naturals*	*naturals*	*multiplication*
$\mathbb{M}$	*naturals*	*matrices*	*MMM*
$\mathbb{I}$	*sets*	$\subseteq$	*see below*

Note that the homset

- $\mathbb{M}\ (m, n)$ may contain an arbitrary number of matrices
- $\mathbb{N}\ (m, n)$ contains either none or just one natural number, $\frac{n}{m}$ if it exists
- $\mathbb{I}\ (A, B)$ contains either none or just one arrow, which we have denoted by $\subseteq$; thus **composition** chains two $\subseteq$ facts.

The category of sets

Next we add to the group the very well-known **category** $\mathbb{S}$ of **sets** and **functions** between sets — for many people, the category "par excellence":

Category	*Objects*	*Arrows*	*Composition*
$\mathbb{N}$	*naturals*	*naturals*	*multiplication*
$\mathbb{M}$	*naturals*	*matrices*	*MMM*
$\mathbb{I}$	*sets*	$\subseteq$	*see above*
$\mathbb{S}$	*sets*	*functions*	*function composition*

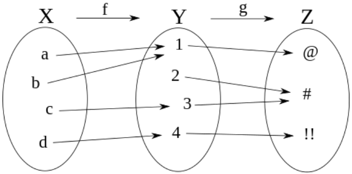
Category $\mathbb{S}$ is the theoretical basis of **functional programming**.

The details of $\mathbb{S}$ are given in the following slide.

The category of sets

In the $\mathbb{S}$ category,

- the **identity** $A \xrightarrow{id} A$ in $\mathbb{S}$ is the *copy-the-input* function $id\ (a) = a$;
- arrow **composition** $X \xrightarrow{f} Y \xrightarrow{g} Z$ is the expected $(g \cdot f)\ (x) = g\ (f\ (x))$ pictured aside.



$(g \cdot f)\ x = g\ (f\ x)$

Homset $\mathbb{S}\ (X, Y)$ is the set of all (total) functions from X to Y.

Functional programming

Some programming languages implement category $\mathbb{S}$ in a rather simple way, notably Haskell:

```
Prelude> :type id
id :: a -> a
Prelude> :type (.)
(.) :: (b -> c) -> (a -> b) -> a -> c
```

Thus, for instance,

```
Prelude> id "Hello"
"Hello"
Prelude> id 3
3
Prelude> (sqrt . succ) 3
2.0
```

(But be warned that full Haskell requires more than category $\mathbb{S}$.)

Pairing two arrows

Back to category $\mathbb{N}$, consider the following diagram:

$$\begin{tikzcd}
18 & & 30 \\
& 1 \arrow[ul, "18"] \arrow[ur, "30"'] &
\end{tikzcd}$$

Fine, since 1 indeed divides any number.

In category theory (CT) jargon, we say that 1 is **initial** in $\mathbb{N}$. This means that there **always** is **exactly one** arrow from 1 to any n.

Diagram tells that 1 is a **common divisor** of 18 and 30.

But not the only one, check e.g.

$$\begin{tikzcd}
18 & & 30 \\
& 2 \arrow[ul, "9"'] \arrow[ur, "15"] & \\
& 1 \arrow[u, "2"] \arrow[uul, "18"] \arrow[uur, "30"'] &
\end{tikzcd}$$

Pairing two arrows

The diagram

$$\begin{tikzcd}
18 & & 30 \\
& 2 \arrow[ul, "9"'] \arrow[ur, "15"] & \\
& 1 \arrow[u, "2"] \arrow[uul, "18"] \arrow[uur, "30"'] &
\end{tikzcd}$$

tells that 2 is also a common divisor, and a larger one.

How far can we go towards **larger** common divisors?

Still another larger one, 6,

$$\begin{tikzcd}
18 & 6 \arrow[l, "3"'] \arrow[r, "5"] & 30 \\
& 2 \arrow[ul, "9"'] \arrow[u, "3"] \arrow[ur, "15"] & \\
& 1 \arrow[u, "2"] \arrow[uul, "18"] \arrow[uur, "30"'] &
\end{tikzcd}$$

but no more — 6 is the **greatest** common divisor (*gcd*) between 18 and 30:

$$gcd\ (18, 30) = 6$$

Pairing two arrows

A similar situation — e.g. diagram of the same shape — but in the set inclusion category $\mathbb{I}$ (omitting the $\subseteq$ label in each arrow):

$$
\begin{tikzcd}
\{a,b,c\} & \{b,c\} \arrow[l] \arrow[r] & \{b,c,d\} \\
& \{c\} \arrow[ul] \arrow[u] \arrow[ur] & \\
& \{\} \arrow[uul] \arrow[u] \arrow[uur] &
\end{tikzcd}
$$

In this case,

$$\{b,c\} = \{a,b,c\} \cap \{b,c,d\}$$

is the **greatest** common subset — known as **intersection**.

Pairing two arrows

Note how both $6 = gcd\ (18, 30)$ and $\{b, c\} = \{a, b, c\} \cap \{b, c, d\}$ are **limit** objects — you cannot find **larger** objects fitting in the diagrams.

To understand the name given in CT jargon to such limit objects,

$$\{a, b, c\} \longleftarrow \{a, b, c\} \cap \{b, c, d\} \longrightarrow \{b, c, d\}$$

and

$$18 \xleftarrow{3} \underbrace{gcd\ (18, 30)}_{6} \xrightarrow{5} 30$$

we will play once again with the same "V"-shaped arrow pattern, this time in the category $\mathbb{S}$ of sets — next slide.

Products

In $\mathbb{S}$, two pairs of functions f, g and i, j fitting in a diagram with the "V"-topology:

$$
\begin{tikzcd}
A & & B \\
& C \arrow[ul, "i"'] \arrow[ur, "j"] & \\
& D \arrow[uul, "f"] \arrow[u, "k"] \arrow[uur, "g"'] &
\end{tikzcd}
$$

Assuming $k : D \to C$ fits in the diagram too.

k "factors" **both** f and g .

The "limit factorization" of f and g occurs when $C = A \times B$, the Cartesian product of A and B,

$$
\begin{tikzcd}
A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\
& D \arrow[ul, "f"] \arrow[u, "k"] \arrow[ur, "g"'] &
\end{tikzcd}
$$

for $i = \pi_1, j = \pi_2$, the two **projections** $\pi_1\ (a, b) = a$ and $\pi_2\ (a, b) = b$.

Products (pairing)

What more can we say about k? From the diagram there is only one choice for k:

$$k\ d = (f\ d, g\ d)$$

That is, given two functions f and g in $\mathbb{S}$ such that

$$\begin{tikzcd} A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\ & D \arrow[ul, "f"] \arrow[u, "k"'] \arrow[ur, "g"'] & \end{tikzcd}$$

there is a unique k such that

$$\pi_1 \cdot k = f \qquad (5)$$

$$\pi_2 \cdot k = g \qquad (6)$$

Conversely, for any function $k \in \mathbb{S}(D, A \times B)$ — that is,

$$\begin{tikzcd} A \times B \\ D \arrow[u, "k"'] \end{tikzcd}$$

in $\mathbb{S}$ there is a unique pair of functions

$$\begin{tikzcd} A & & B \\ & D \arrow[ul, "f"] \arrow[ur, "g"'] & \end{tikzcd}$$

which fit into the diagram, as given by (5,6).

Products

Thus, there is a **bijection** between *pair-valued* functions and *pairs* of functions,

$$\begin{tikzcd} \mathbb{S}\,(D, A \times B) \arrow[r, bend left, "unsplit"] & \mathbb{S}\,(D, A) \times \mathbb{S}\,(D, B) \arrow[l, bend left, "\langle \cdot,\cdot \rangle"] \end{tikzcd}$$

$$k \overset{}{\underset{unsplit}{\longmapsto}} (f, g)$$

$$k \overset{\langle \cdot,\cdot \rangle}{\longleftarrow\!\!\!\dashv} (f, g)$$

where $unsplit\ k = (\pi_1 \cdot k, \pi_2 \cdot k)$.

Below we will prefer the "outfix" notation $k = \langle f, g \rangle$ instead of "prefix" notation $k = \langle f, g, \cdot \rangle$.

Products

Another way of capturing the same bijection is to write the **universal** property:

$$k = \langle f, g \rangle \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \pi_1 \cdot k = f \\ \pi_2 \cdot k = g \end{array} \right. \tag{7}$$

$$\begin{tikzcd} A & A \times B \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & B \\ & D \arrow[ul, "f"] \arrow[u, "k"'] \arrow[ur, "g"'] & \end{tikzcd}$$

Interpret (7) as explained on the right.

Given f and g as in the diagram,

- ($\Rightarrow$) **existence** — there is always some k fitting into the diagram
- ($\Leftarrow$) **uniqueness** — such a k is unique.

Products in $\mathbb{I}$

Omitting the $\subseteq$ labels, the product diagram in $\mathbb{I}$ is

$$\begin{tikzcd} A & A \cap B \arrow[l] \arrow[r] & B \\ & D \arrow[ul] \arrow[u] \arrow[ur] & \end{tikzcd}$$

The product bijection

$$\begin{tikzcd} \mathbb{I}\,(D, A \cap B) \arrow[rr, bend left, "unsplit"] & & \mathbb{I}\,(D, A) \times \mathbb{I}\,(D, B) \arrow[ll, bend left, "\langle \cdot, \cdot \rangle"] \end{tikzcd}$$

in this case instantiates to the universal property of **intersection**:

$$\begin{tikzcd} D \subseteq (A \cap B) \arrow[rr, bend left, "\Rightarrow"] \arrow[rr, phantom, "\Leftrightarrow"] & & (D \subseteq A) \wedge (D \subseteq B) \arrow[ll, bend left, "\Leftarrow"] \end{tikzcd}$$

Products in $\mathbb{N}$

Next, the same "V"-diagram and property in $\mathbb{N}$ with an example, abbreviating $gcd\ (x, y)$ by $x \triangledown y$:

$$
\begin{tikzcd}
m & m \triangledown n \arrow[l, "\frac{m}{m \triangledown n}"'] \arrow[r, "\frac{n}{m \triangledown n}"] & n \\
& p \arrow[ul, "a"] \arrow[u, "a \triangledown b"] \arrow[ur, "b"'] &
\end{tikzcd}
\qquad
\begin{tikzcd}
18 & 6 \arrow[l, "3"'] \arrow[r, "5"] & 30 \\
& 2 \arrow[ul, "9"] \arrow[u, "3"] \arrow[ur, "15"'] &
\end{tikzcd}
$$

Universal property:

$$k = a \triangledown b \quad \Leftrightarrow \quad \begin{cases} \frac{m}{m \triangledown n} \cdot k = a \\ \frac{n}{m \triangledown n} \cdot k = b \end{cases}$$

Corollary ($k = a \triangledown b$):

$$\frac{m}{a} = \frac{n}{b} = \frac{m \triangledown n}{a \triangledown b}$$

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Products in $\mathbb{M}$

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Finally, still the same "V"-shape and property, now in $\mathbb{M}$:

$$\begin{tikzcd} m & m+n \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & n \\ & p \arrow[ul, "M"] \arrow[u, "\begin{bmatrix} M \\ N \end{bmatrix}"] \arrow[ur, "N"'] & \end{tikzcd}$$

NB: $\left[\frac{M}{N}\right]$ is the vertical stacking of two matrices M and N with the same number of columns p.

Universal property:

$$X = \left[\frac{M}{N}\right] \Leftrightarrow \begin{cases} \pi_1 \cdot X = M \\ \pi_2 \cdot X = N \end{cases} \tag{8}$$

What kind of matrices are π_1 and π_2? (Next slide)

Products in $\mathbb{M}$

Consider this instance of the diagram:

$$\begin{tikzcd} m & m+n \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] & n \\ & m+n \arrow[ul, "\pi_1"] \arrow[u, "\begin{bmatrix}\pi_1 \\ \pi_2\end{bmatrix}"] \arrow[ur, "\pi_2"'] & \end{tikzcd}$$

Necessarily,

$$\begin{bmatrix} \pi_1 \\ \hline \pi_2 \end{bmatrix} = id \qquad (9)$$

because of **uniqueness**.

Don't buy this argument?

Take the universal property,

$$X = \begin{bmatrix} M \\ \hline N \end{bmatrix} \Leftrightarrow \begin{cases} \pi_1 \cdot X = M \\ \pi_2 \cdot X = N \end{cases}$$

for $X = id$ and solve it for M and N. You get $M = \pi_1$ and $N = \pi_2$.

Products in $\mathbb{M}$

Equality (9) tells that π_1 and π_2 are complementary **fragments** of the identity matrix, for example $2 \xleftarrow{\pi_1} 2+3 \xrightarrow{\pi_2} 3$ in MATLAB:

```
>> p1(2,3)

    1     0     0     0     0
    0     1     0     0     0

>> p2(2,3)

    0     0     1     0     0
    0     0     0     1     0
    0     0     0     0     1
```

```
>> [ p1(2,3) ; p2(2,3) ]

    1     0     0     0     0
    0     1     0     0     0
    0     0     1     0     0
    0     0     0     1     0
    0     0     0     0     1
```

π_1, π_2 and *id* are examples of **Boolean** matrices — they contain either 0s or 1s.

Reversing the arrows

Suppose one wants to investigate what it means to **reverse** the arrows of a diagram. Does it make sense? What does one get?

In $\mathbb{M}$, each arrow $m \xrightarrow{M} n$ can be converted into the reverse arrow $m \xleftarrow{M^\circ} n$ known as the **converse** or **transpose** of M, such that

$$(M \cdot N)^\circ = N^\circ \cdot M^\circ \tag{10}$$

$$(M^\circ)^\circ = M \tag{11}$$

Clearly, M° is M with rows swapped with columns.

Coproducts in $\mathbb{M}$

The converse of a **product** diagram in $\mathbb{M}$ is a so-called **coproduct** diagram,

$$
\begin{tikzcd}
m \arrow[r, "\pi_1^\circ"] \arrow[dr, "M"'] & m+n \arrow[d, "{[M|N]}"] & n \arrow[l, "\pi_2^\circ"'] \arrow[dl, "N"] \\
& p &
\end{tikzcd}
$$

NB: $[M|N]$ is the **horizontal** gluing of two matrices M and N with the same number of rows p.

Universal property:

$$
X = [M|N] \;\Leftrightarrow\; \left\{ \begin{array}{l} X \cdot i_1 = M \\ X \cdot i_2 = N \end{array} \right. \tag{12}
$$

where $i_1 = \pi_1^\circ$ and $i_2 = \pi_2^\circ$ are known as **injections**.

Coproducts in $\mathbb{M}$

Terminology:

Read $\left[\frac{M}{N}\right]$ *as "M split N" and* $[M|N]$ *as "M junc N".*

Duality:

$$[M|N]^\circ = \left[\frac{M^\circ}{N^\circ}\right] \tag{13}$$

Exchange law:

$$\left[\left[\frac{M}{N}\right] \middle| \left[\frac{P}{Q}\right]\right] = \left[\frac{[M|P]}{[N|Q]}\right] \tag{14}$$

(More in the sequel.)

Coproducts in other categories

This duality in $\mathbb{M}$ is quite strong and is called **self duality**. In other categories the objects involved may change.

In $\mathbb{I}$, **coproducts** correspond to set **union**, cf.

$$\begin{tikzcd}
& D & \\
& A \cup B \arrow[u] & \\
A \arrow[ur] \arrow[uur] & & B \arrow[ul] \arrow[uul]
\end{tikzcd}$$

In $\mathbb{N}$, coproducts correspond to **least common multiple**, cf.

$$\begin{tikzcd}
& 60 & \\
& 30 \arrow[u, "2"'] & \\
6 \arrow[ur, "5"'] \arrow[uur, "10"] & & 10 \arrow[ul, "3"] \arrow[uul, "6"']
\end{tikzcd}$$

Universal properties

$$(A \cup B) \subseteq D \quad \Leftrightarrow \quad A \subseteq D \wedge B \subseteq D$$

etc

Coproducts in $\mathbb{S}$

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

The coproduct diagram in $\mathbb{S}$:

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "f"'] & A+B \arrow[d, "{[f,g]}"] & B \arrow[l, "i_2"'] \arrow[dl, "g"] \\
& C &
\end{tikzcd}$$

NB: $A+B$ is the **disjoint union** of A and B under injections $i_1\ a = (1, a)$ and $i_2\ b = (2, b)$; the intuitive meaning of $[f, g]$ will be given later.

Universal property:

$$k = [f, g] \quad \Leftrightarrow \quad \left\{ \begin{array}{l} k \cdot i_1 = f \\ k \cdot i_2 = g \end{array} \right. \tag{15}$$

Combinator $[f, g]$ hides a kind of "if-then-else": either f or g to run depending on the type of the input.

Conditional arrows in $\mathbb{S}$

Given a **predicate** $A \xrightarrow{p} Bool$, define

$$p? : A \rightarrow A + A$$

$$p?\ a = \begin{cases} i_1\ a \Leftarrow p\ a \\ i_2\ a \Leftarrow \neg\ p\ a \end{cases}$$

Then arrow composition grants the existence of **conditional** arrows:

$$\textbf{if}\ p\ \textbf{then}\ f\ \textbf{else}\ g = [f, g] \cdot p?$$

These arrows are of the same type as their arguments f, g.

(Note how we are getting closer and closer to having a **programming language**...)

Diagrams as proofs

The **existence** and **uniqueness** of arrow k in (15) offers a nice, diagrammatic way of calculating properties.

For instance, adding another arrow h to the coproduct diagram entails, by **existence**:

$$
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "f"'] & A+B \arrow[d, "{[f,g]}"] & B \arrow[l, "i_2"'] \arrow[dl, "g"] \\
& C \arrow[d, "h"] & \\
& D &
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "f"'] \arrow[ddr, "h \cdot f"'] & A+B \arrow[d, "{[f,g]}"] & B \arrow[l, "i_2"'] \arrow[dl, "g"] \arrow[ddl, "h \cdot g"] \\
& C \arrow[d, "h"] & \\
& D &
\end{tikzcd}
$$

Diagrams as proofs

Still **existence**:

$$\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[ddr, "h \cdot f"'] & A+B \arrow[d, "{[f,g]}"] \arrow[dd, bend left, "{h \cdot [f,g]}"] & B \arrow[l, "i_2"'] \arrow[ddl, "h \cdot g"] \\
 & C \arrow[d, "h"] & \\
 & D &
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "h \cdot f"'] & A+B \arrow[d, bend right, "{[h \cdot f, h \cdot g]}"'] \arrow[d, bend left, "{h \cdot [f,g]}"] & B \arrow[l, "i_2"'] \arrow[dl, "h \cdot g"] \\
 & D &
\end{tikzcd}$$

Next, by **uniqueness**:

$$h \cdot [f, g] \quad = \quad [h \cdot f, h \cdot g] \tag{16}$$

This is known as the coproduct-**fusion** law.

Saving your brain

A similar sequence of diagrams will yield the **product**-fusion law

$$\langle f, g\rangle \cdot h \quad = \quad \langle f \cdot h, g \cdot h\rangle \tag{17}$$

in $\mathbb{S}$, etc. **Most importantly**:

Due to the **abstract equivalence** of all product or co-product diagrams, we can port these properties from $\mathbb{S}$ to other categories,

e.g. the **gcd**-fusion law in $\mathbb{N}$,

$$(a \triangledown b) \cdot c \quad = \quad (a \cdot c) \triangledown (b \cdot c)$$

the two fusion laws of blocked linear algebra in $\mathbb{M}$,

$$\left[\frac{M}{N}\right] \cdot Q \quad = \quad \left[\frac{M \cdot Q}{N \cdot Q}\right] \tag{18}$$

$$Q \cdot [M|N] \quad = \quad [Q \cdot M | Q \cdot N] \tag{19}$$

and so on and so forth.

Save your brain!

Summing up:

- *Saving one's brain* is perhaps the most **practical** outcome of CT: a single, **generic** construct instantiates to many — semantically disparate — concrete constructs.
- A single proof (using **diagrams**, if you like) replaces many repetitive proofs at instance level.
- **Products** and **coproducts** are themselves instances of more general constructs known as **limits** and **colimits** (cf. later in the course).
- A single, unified and **typed** (arrow) **language** for all domains.

More about this next.

Generalizing $\mathbb{M}$

The objects of $\mathbb{M}$ can be generalized from numeric dimensions (n, $m \in \mathbb{N}_0$) to arbitrary denumerable sets (types) (X, Y), taking

- **disjoint union** $X + Y$ for $m + n$,
- Cartesian product $X \times Y$ for $m \times n$
- Any (fixed) singleton type 1 for number $1 \in \mathbb{N}$.

Matrix multiplication — **composition** in $\mathbb{M}$ — is still well-defined since addition is commutative, so the order in the summation

$$y\ (M \cdot N)\ x \quad = \quad \langle \sum z\ ::\ (y\ M\ z) \times (z\ N\ x) \rangle \tag{20}$$

is irrelevant.

NB: *note that the* (b, a)*-cell of* **matrix** M *is denoted by* $b\ M\ a$ *and not by* $M_{b,a}$ *or any other notation. (The rationale behind this choice of notation will be explained later.)*

Subcategories

Are the **categories** we have seen completed **isolated worlds**?

No — we can relate/combine them. Let us show, as example, how $\mathbb{S}$ can be expressed in (generalized) $\mathbb{M}$.

Think of a function $f : A \to B$ in $\mathbb{S}$ and check how it can be represented by a matrix $M : A \to B$ in $\mathbb{M}$, say

$$M = [\![f]\!]$$

defined by

$$b\ [\![f]\!]\ a \quad = \quad \begin{cases} 1 \textbf{ if } b = (f\ a) \\ 0\ otherwise \end{cases} \tag{21}$$

Subcategories

Example: the function in $\mathbb{S}$

$$Bool \xrightarrow{\neg} Bool \;=\; \left\{ \begin{array}{l} \neg\ False = True \\ \neg\ True = False \end{array} \right.$$

is represented in $\mathbb{M}$ by the matrix

$$Bool \xrightarrow{[\![\neg]\!]} Bool \;=\; \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

— a representation we use all the time, albeit "informally".

The same matrix with the **typing** made explicit:

$$Bool \xrightarrow{[\![\neg]\!]} Bool \;=\; \begin{array}{r|cc} & False & True \\ \hline False & 0 & 1 \\ True & 1 & 0 \end{array}$$

Subcategories

Moreover, everything we can do in $\mathbb{S}$ can be done in $\mathbb{M}$, for instance **composition**

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

with **identity** $[\![id]\!] = id$ — cf. the following check in MATLAB that negation ($\neg$) is a bijection:

```
>> not = [0 1; 1 0]

>> not * not

ans =

     1     0
     0     1
```

Subcategories

However, not every **matrix** in $\mathbb{M}$ represents a **function** from $\mathbb{S}$.

Not even every **Boolean matrix** (containing zeros and ones only) represents a function, e.g.

$$M = \begin{bmatrix} 1 & 0 \\ 1 & 0 \end{bmatrix}$$

Here we don't know which output for the first argument to choose, and for the second argument the "function" is **undefined**...

We say that $\mathbb{S}$ is a **subcategory** *of $\mathbb{M}$.*

(This relationship will be later made more precise using so-called **functors** between categories.)

Subcategories

There is a nice way of checking whether a matrix represents a function on not.

Pick the **unique** function that one can think of, of type $A \to 1$ — (necessarily) a **constant** function.

This function is usually named "bang" (as it "destroys" every argument!) and written $! : A \to 1$.

A **Boolean** *matrix* $M : A \to B$ *in* $\mathbb{M}$ **uniquely** *represents a function of the same type* $A \to B$ *in* $\mathbb{S}$ *iff*

$$! \cdot M = ! \tag{22}$$

holds.

Note the *polymorphism* of the two copies of $!$.

Subcategories

MATLAB: checking that matrix

```
M =
     1   0   1
     0   1   0
```

represents a function:

```
>> [1 1] * M

ans =

     [ 1   1   1 ]
```

You can see above two copies of the polymorphic "bang" matrix.

Probabilistic functions ($\mathbb{P}$)

QUESTION: What does clause (22) mean in case we relax the **Boolean** requirement and let M hold positive real numbers?

One easily checks that e.g. the following matrix,

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

satisfies (22).

Thus we have found another interesting **subcategory** of $\mathbb{M}$ — that which includes all **probabilistic functions** ($\mathbb{P}$), instances of which are commonly known as **Markov chains**.

$\mathbb{S}$ is also a subcategory of $\mathbb{P}$ — as "pure" functions correspond to restricting to **Dirac distributions** — one sole 1 per column.

Distributions ($\mathbb{P}$)

What does a **morphism** of type $1 \longrightarrow A$ mean?

- In $\mathbb{N}$, $1 \xrightarrow{n} n$ denotes the **number** n itself
- In $\mathbb{S}$, $1 \xrightarrow{p} A$ means a **point**, since p can only yield one element (point) of A
- In $\mathbb{M}$, $1 \xrightarrow{v} A$ is known as a **column vector** (matrix with only one column)
- In $\mathbb{P}$, $1 \xrightarrow{\delta} A$ is called a **distribution**, for instance

$$1 \xrightarrow{\delta} 6 \quad = \quad \begin{bmatrix} 0 \\ 0.2 \\ 0.2 \\ 0.6 \\ 0 \\ 0 \end{bmatrix}$$

Probabilistic pairing

What does **arrow pairing** mean in $\mathbb{M}$ or $\mathbb{P}$? Check the diagram:

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

Probabilistic pairing

In $\mathbb{M}$ (also $\mathbb{P}$) pairing corresponds to the so called **Khatri-Rao product** $A \times B \xleftarrow{M \triangledown N} C$ of two matrices $A \xleftarrow{M} C$ and $B \xleftarrow{N} C$:

$$(a, b)\,(M \triangledown N)\; c \;=\; (a\; M\; c) \times (b\; N\; c) \tag{23}$$

Example in MATLAB:

```
>> kr([1,0,11;-3,-4,0;], [0,1,3;0,2,4;])
ans =

      0      0     33
      0      0     44
      0     -4      0
      0     -8      0
```

Probabilistic pairing

This time there are differences, however, compared to $\mathbb{S}$, as pairing in $\mathbb{P}$ is not perfect:

$$X = M \triangledown N \;\;\Rightarrow\;\; \left\{ \begin{array}{r} \mathit{fst} \cdot X = M \\ \mathit{snd} \cdot X = N \end{array} \right. \tag{24}$$

As ($\Leftarrow$) is not guaranteed, lossless decomposition may fail and

$$(\mathit{fst} \cdot X) \triangledown (\mathit{snd} \cdot X)$$

differs from X in general.

We say that **pairing** in $\mathbb{P}$ is a **weak**-product.

Note, however, that $[\![f]\!] \triangledown [\![g]\!] = [\![\langle f, g\rangle]\!]$, where f and g are "pure" functions.

Probabilistic pairing (entanglement)

The problem is that **reconstruction**

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

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

$$X : 2 \to 2 \times 3 \qquad X = \begin{bmatrix} 0 & 0.4 \\ 0.2 & 0 \\ 0.2 & 0.1 \\ 0.6 & 0.4 \\ 0 & 0 \\ 0 & 0.1 \end{bmatrix} \qquad (fst \cdot X) \triangledown (snd \cdot X) = \begin{bmatrix} 0.24 & 0.4 \\ 0.08 & 0 \\ 0.08 & 0.1 \\ 0.36 & 0.4 \\ 0.12 & 0 \\ 0.12 & 0.1 \end{bmatrix}$$

X is not recoverable from its projections: Khatri-Rao not **surjective**.

In **quantum** computing this situation is known as **entanglement** — entangled distributions on pairs cannot be projected into pairs of distributions.

Probabilistic programming

Consider the following instance of pairing, in $\mathbb{S}$,

$$12 \xleftarrow{add} 6 \times 6 \xleftarrow{a^{\triangledown} b} 1 \qquad (25)$$

where $add\ (x, y) = x + y$.

Clearly, the composition expresses the term $a + b$, since a and b stand for numbers in $\{1 \mathinner{..} 6\}$.

What does the **same diagram** mean in $\mathbb{P}$? In this case a and b are bound to be **distributions**.

One can think of a and b being two **dice** and of diagram (25) as expressing the **probability of the sum** of the faces shown (next slide).

Probabilistic programming

Assume both dice are **fair**, that is, the input distributions are **uniform**:

$$a = b = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}^{\circ}$$

Addition is represented by the matrix

$$y \, [\![add]\!] \, (a, b) = \textbf{if } y = a + b \textbf{ then } 1 \textbf{ else } 0$$

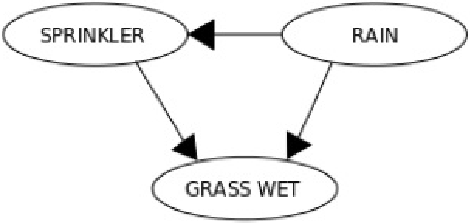
Then the arrow $12 \xleftarrow{add \cdot (a \triangledown b)} 1$ evaluates to the distribution aside, where outcome 1 is impossible, outcome 7 is the most likely, and so son.

$$\begin{bmatrix} 0 \\ 0.0278 \\ 0.0555 \\ 0.0833 \\ 0.1111 \\ 0.1389 \\ 0.1667 \\ 0.1389 \\ 0.1389 \\ 0.0833 \\ 0.0555 \\ 0.0278 \end{bmatrix}$$

Probabilistic modelling

Example adapted from

[`https://en.wikipedia.org/wiki/Bayesian_network`]



Control a **sprinkler** to wet the **grass** in case it does not **rain**.

Deterministic model in $\mathbb{S}$

$S = R = G = 2$

$sprinkler : R \to S$
$sprinkler\ r = \neg\ r$

$grass : S \times R \to G$
$grass\ (s, r) = s \vee r$

$rain \in \{0, 1\}$

$$\begin{tikzcd} G \times (S \times R) \\ S \times R \arrow[u, "grass^{\triangledown}\ id"'] \\ R \arrow[u, "sprinkler^{\triangledown}\ id"'] \\ 1 \arrow[u, "rain"'] \end{tikzcd}$$

Grass always wet:

$$grass\ (sprinkler\ r, r) = \neg\ r \vee r = True$$

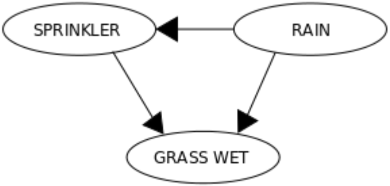
Altogether, two possible states $\{(1, (1, 0)), (1, (0, 1))\}$ of type:

$$G \times (S \times R) \xleftarrow{state} 1 \ = (grass \triangledown id) \cdot (sprinkler \triangledown id) \cdot rain$$

Bayesian networks

Previous model (in $\mathbb{S}$) is not realistic — the pictures actually found on Wikipedia are:

	SPRINKLER	
RAIN	T	F
F	0.4	0.6
T	0.01	0.99



RAIN	
T	F
0.2	0.8

		GRASS WET	
SPRINKLER	RAIN	T	F
F	F	0.0	1.0
F	T	0.8	0.2
T	F	0.9	0.1
T	T	0.99	0.01

Bayesian network ($\mathbb{P}$)

This corresponds to moving to $\mathbb{P}$ and letting

$$S = R = G = 2$$

$$S \xleftarrow{sprinkler} R = \begin{bmatrix} 0.60 & 0.99 \\ 0.40 & 0.01 \end{bmatrix}$$

$$R \xleftarrow{rain} 1 = \begin{bmatrix} 0.80 \\ 0.20 \end{bmatrix}$$

$$G \xleftarrow{grass} S \times R = \begin{bmatrix} 1.00 & 0.20 & 0.10 & 0.01 \\ 0 & 0.80 & 0.90 & 0.99 \end{bmatrix}$$

$$\begin{array}{c} G \times (S \times R) \\ \uparrow {\scriptstyle grass \triangledown id} \\ S \times R \\ \uparrow {\scriptstyle sprinkler \triangledown id} \\ R \\ \uparrow {\scriptstyle rain} \\ 1 \end{array}$$

The "same" state arrow

$$G \times (S \times R) \xleftarrow{state} 1 = (grass \triangledown id) \cdot (sprinkler \triangledown id) \cdot rain$$

now has a different meaning since the category became $\mathbb{P}$ (next slide).

Bayesian network ($\mathbb{P}$)

$$G \times (S \times R) \xleftarrow{state} 1 \quad =$$

G	S	R	
dry	off	*no*	0.4800
		yes	0.0396
	on	*no*	0.0320
		yes	0.0000
wet	off	*no*	0.0000
		yes	0.1584
	on	*no*	0.2880
		yes	0.0020

Moreover, we can define

$$1 \xleftarrow{grass\_wet} G \times (S \times R) = [0\ 1] \cdot fst$$

$$1 \xleftarrow{raining} G \times (S \times R) = [0\ 1] \cdot snd \cdot snd$$

etc. to obtain e.g. $\mathrm{P}_{state}(grass\_wet) = grass\_wet \cdot state = 44.84\%$.

Bayesian network querying

Conditional probabilities over a state distribution δ:

$$\mathrm{P}_\delta(a \mid b) = \frac{(a \times b) \cdot \delta}{b \cdot \delta} \quad \textit{given} \quad 1 \xleftarrow{a,b} S \xleftarrow{\delta} 1 \tag{26}$$

Boolean vectors a and b describe "event" sets.

Recall

$$\begin{tikzcd}
G \times (S \times R) \arrow[d, "grass_wet{,}raining"'] & & R \arrow[ll, "(grass^{\triangledown}\, id)\cdot(sprinkler^{\triangledown}\, id)"'] \\
1 & & 1 \arrow[u, "rain"'] \arrow[ull, "state"'] \arrow[ll, "\frac{(grass_wet \times raining)\cdot state}{grass_wet \cdot state}"]
\end{tikzcd}$$

Forwards: $\mathrm{P}_{state}(grass\_wet \mid raining) = 80.19\%$

Backwards: $\mathrm{P}_{state}(raining \mid grass\_wet) = 35.77\%$

Summary thus far

Diagrams central to category theory.

They can express **abstract** properties or abstract **models** of problems.

A diagram modeling a problem captures its essence, or abstract structure.

Keeping the diagram, more elaborate semantics for the problem can be expressed **just** by changing category:

"Keep definition, change category" principle (Oliveira and Miraldo, 2016)

Enriched categories

Given two numbers a and b, we can add them $(a+b)$, multiply them $(a \times b)$ etc.

Likewise, given two matrices $n \xleftarrow{M,N} m$ we can add them $(M+N)$, e.g.

$$\begin{pmatrix} 1 & 0 & 0 \\ -9 & 3 & 12 \\ 0 & -1 & 0 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 \\ -7 & 6 & 16 \\ 0 & 0 & 0 \end{pmatrix}$$

and multiply them $(M \times N)$:

$$\begin{pmatrix} 1 & 0 & 0 \\ -9 & 3 & 12 \\ 0 & -1 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -18 & 9 & 48 \\ 0 & -1 & 0 \end{pmatrix}$$

But, matrices are **arrows** (morphisms) — does it make sense to "add / multiply arrows"?

Enriched categories

Yes it does, in so-called **enriched**-categories: categories with **extra** mathematical **structure**.

Recall that a homset $\mathbb{C}(m, n)$ in a category $\mathbb{C}$ is a set.

An **enrichment** consists in adding some algebraic structure to such sets.

Some care is needed concerning the interplay between such enriched $\mathbb{C}(m, n)$ and the basic structure, namely **composition**.

Abelian categories (next slide) are particularly interesting cases of enriched categories.

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Abelian categories

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

$\mathbb{M}$ is **Abelian** because every homset $\mathbb{M}(m, n)$ forms an additive Abelian group (**Ab**-category) such that composition is bilinear relative to $+$:

$$M \cdot (N + L) = M \cdot N + M \cdot L \tag{27}$$

$$(N + L) \cdot K = N \cdot K + L \cdot K \tag{28}$$

The Abelian structure grants $M + 0 = M$, where 0 is the all-0 matrix of its type.

The Abelian structure can be further enriched to a ring with $M \times N$ *given by the so-called* **Hadamard** *product:*

$$b\,(M \times N)\,a = (b\;M\;a) \times (b\;N\;a) \tag{29}$$

$M \times 1 = M$ *holds where* 1 *is the all-*1 *matrix of its type.*

NB: as before we assume $\mathbb{M}$ defined over the reals.

Abelian categories

The additive structure of $\mathbb{M}$ grants a number of laws, namely the so called **divide-and-conquer** law

$$[M|N] \cdot \left[\frac{P}{Q}\right] = M \cdot P + N \cdot Q \qquad (30)$$

which is the basis of (**parallel**) blocked linear algebra, and can also be written as

$$[M|N] \cdot [P|Q]^\circ = M \cdot P^\circ + N \cdot Q^\circ \qquad (31)$$

It turns out that

$$[M|N] = M \cdot \pi_1 + N \cdot \pi_2 \qquad (32)$$

$$\left[\frac{M}{N}\right] = i_1 \cdot M + i_2 \cdot N \qquad (33)$$

also hold.

Order enrichment

Categories can also be enriched by regarding homsets as ordered structures, for instance **partial orders**.

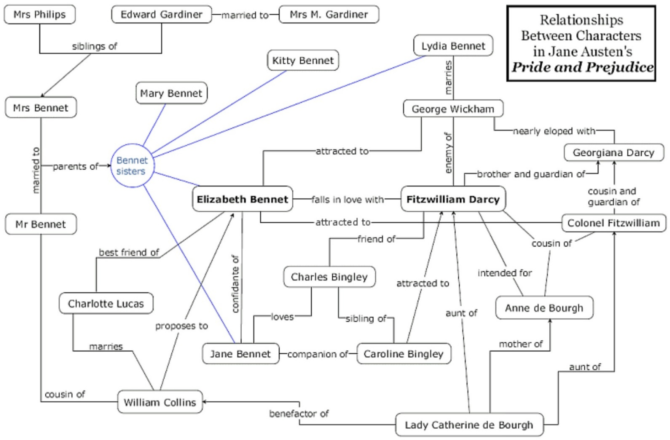
This is useful when we want to solve **recursive equations** in a category (more about this later).

The topic brings about another category — the category of **binary relations** $\mathbb{R}$.

This category is very useful to model real-life problems: relational **databases** rely on $\mathbb{R}$ by definition.

Everything is a relation...

... in **real** life — recall



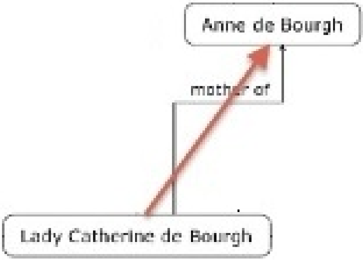
Arrow notation for relations

The picture is a collection of **relations** — aka. a **semantic network** — elsewhere known as a (binary) **relational system**.

However, in spite of the use of **arrows** in the picture (aside) not many people would write

$$mother\_of : People \rightarrow People$$

as the **type** of **relation** $mother\_of$.



The category of relations $\mathbb{R}$

Let the arrows of $\mathbb{S}$ be not only functions, say $A \xrightarrow{f} B$, but also relations $A \xrightarrow{R} B$.

In the same way assertion $b = f\ a$ may hold or not, so may $b\ R\ a$, the assertion that pair (b, a) belongs to R.

Thus extended, $\mathbb{S}$ becomes $\mathbb{R}$, the category of **binary relations**.

In $\mathbb{R}$, id is the **equality** relation; **composition** $R \cdot S$ is given by

$$b\,(R \cdot S)\,c \quad \Leftrightarrow \quad \exists\, a : b\ R\ a \wedge a\ S\ c \tag{34}$$

cf.

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

The category of relations $\mathbb{R}$

In general, the **converse** f° of a function f is a relation, not a function.

Thus $\mathbb{S}$ does not have converse morphisms, while $\mathbb{R}$ does: $a\ (R^\circ)\ b$ means the same as $b\ R\ a$ — as in $\mathbb{M}$, recall.

Like in $\mathbb{M}$, we have the laws

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \tag{35}$$

$$R^{\circ\circ} = R \tag{36}$$

So $\mathbb{R}$ is another example of a self-dual category. Arrows $A \to 1$ and $1 \to A$ both denote **sets**.

The category of relations $\mathbb{R}$

Category $\mathbb{R}$ provides a useful generalization of $\mathbb{S}$.

A rich terminology emerges simply by defining the **order**

$$R \subseteq S \quad \Leftrightarrow \quad R \cup S = S \tag{37}$$

on the homsets:

- R is **reflexive** iff $id \subseteq R$
- R is **symmetric** iff $R^\circ \subseteq R$
- R is **transitive** iff $R \cdot R \subseteq R$
- R is **injective** iff $R^\circ \cdot R \subseteq id$
- R is **simple** (aka. a partial function) iff $R \cdot R^\circ \subseteq id$
- R is **entire** (aka. total) iff $id \subseteq R^\circ \cdot R$
- R is **surjective** iff $id \subseteq R \cdot R^\circ$

The category of relations $\mathbb{R}$

Each homset $\mathbb{R}\,(A, B)$ forms a **Boolean algebra** under union ($\cup$), intersection ($\cap$) and complementation, plus a **topmost** relation $B \xleftarrow{\top} A$ and a **least** relation $B \xleftarrow{\bot} A$.

Pairing in $\mathbb{R}$

$$(a, b)\ (R \triangledown S)\ c \ = \ (a\ R\ c) \wedge (b\ S\ c) \tag{38}$$

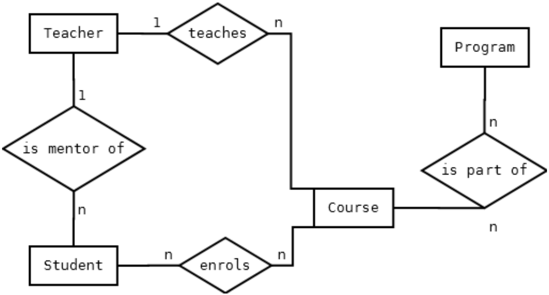
does not even form a weak-product. But its universal property takes advantage of the order-enriched structure:

$$\pi_1 \cdot X \subseteq R \ \wedge \ \pi_2 \cdot X \subseteq S \quad \Leftrightarrow \quad X \subseteq R \triangledown S \tag{39}$$

(This is an example of a so-called **Galois** connection.)

Exercise

So-called "**Entity-Relationship**" (ER) diagrams are commonly used to capture relational data schemas, e.g.[1]



Draw the same using morphism (arrows) in $\mathbb{R}$ and identify the properties of each relation in the diagram.

[1] Credits: `https://dba.stackexchange.com/questions`.

Relational programming

If $\mathbb{S}$ supports functional programming, $\mathbb{R}$ supports another programming paradigm: take the (simple) Prolog program

```
mother_child(trude, sally).

father_child(tom, sally).
father_child(tom, erica).
father_child(mike, tom).

parent_child(X, Y) :- father_child(X, Y).
parent_child(X, Y) :- mother_child(X, Y).

sibling(X, Y)      :- parent_child(Z, X), parent_child(Z, Y).
grand_parent(X, Y) :- parent_child(X, Z), parent_child(Z, Y).
```

Relational programming ($\mathbb{R}$)

Meaning of this program in category $\mathbb{R}$:

$$P \xleftarrow[\substack{father\_child \\ mother\_child \\ parent\_child}]{\substack{sibling \\ grand\_parent}} P \xleftarrow{\{trude, sally, \ldots\}} 1$$

Clauses:

$$mother\_child \cup father\_child \subseteq parent\_child \tag{40}$$

$$parent\_child^{\circ} \cdot parent\_child \subseteq sibling \tag{41}$$

$$parent\_child \cdot parent\_child \subseteq grand\_parent \tag{42}$$

Note how object P (type for people) is made explicit (typing!).

Relational programming ($\mathbb{R}$)

Running query

```
?- sibling(erica,sally)
```

cf. diagram

$$\begin{tikzcd} & 1 \arrow[dl, "erica"'] \arrow[d, "sally"] \\ P & P \arrow[l, "sibling"] \end{tikzcd}$$

corresponds to checking whether arrow $1 \xleftarrow{erica^\circ \cdot sibling \cdot sally} 1$ (a scalar in $\mathbb{R}$) is empty or not.

NB: *erica* and *sally* are atoms, therefore ("atomic") functions.

Relational programming ($\mathbb{R}$)

Checking:

$$
\begin{array}{ll}
 & erica^{\circ} \cdot sibling \cdot sally \\
\supseteq & \qquad \{ \ (41)\ ;\ (35) \ \} \\
 & (parent\_child \cdot erica)^{\circ} \cdot parent\_child \cdot sally \\
\supseteq & \qquad \{ \ (40) \ \} \\
 & (father\_child \cdot erica)^{\circ} \cdot father\_child \cdot sally \\
= & \qquad \{ \ \text{facts} \ \} \\
 & tom^{\circ} \cdot tom \\
= & \qquad \{ \ tom \text{ is an atom} \ \} \\
 & \top \\
\Box &
\end{array}
$$

What about quantum programming?

"Equation" *a la* Wirth:

(Quantum) Programs = (Quantum) Algorithms + (Quantum) Data Structures

Quantum algorithms based on elementary **components**, called **quantum gates**.

Classical bits generalize to quantum bits (**qubits**) — quantum data.

		CONTROL	
		Classic	Quantum
DATA	Classic	–	x
	Quantum	x	x

What about quantum programming?

In quantum programming, all computations are **reversible**.

This is expressed in linear algebra by so-called **unitary** matrices.

Standard **quantum programming gates**, used in **quantum circuits** (Nielsen and Chuang, 2011) can be expressed in $\mathbb{M}$, that is, in typed LA.

They can be **decomposed** into polymorphic, elementary matrix categorial **units**.

Pairing (Khatri-Rao $\triangledown$ + Kronecker products $\otimes$) is central to quantum data structuring.

From now on we extend matrices in $\mathbb{M}$ to hold **complex** numbers ($\mathbb{C}$) ans not just reals.

Unitary gates

A $\mathbb{C}$-valued matrix U is unitary iff $U \cdot U^* = U^* \cdot U = id$, where U^* is the **conjugate** transpose of U.

Thus all isomorphisms (reversible functions) are special cases of **unitary** matrices.

But **isomorphisms** admit further decompositions in terms of such matrices, for instance "the sqrt of not"

$$\neg = (\sqrt{\neg}) \cdot (\sqrt{\neg})$$

where

$$\sqrt{\neg} = \frac{1}{2}\left(\top + i\,(id - \neg)\right) = \frac{1}{2}\begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

Thus one gets into the wonderful world of **actual** quantum gates in which classical logic operations are no longer primitive.

Quantum processing

Quantum application — like function application, the outcome of processing quantum data s by quantum gate P is given by $P \cdot s$.

$$\begin{tikzcd} & 1 \arrow[dl, "P\cdot s"'] \arrow[d, "s"] \\ B & A \arrow[l, "P"] \end{tikzcd}$$

Qubits — The smallest (useful) A is 2, the Booleans — so a (qu)bit $2 \xleftarrow{s} 1$ is always a vector of the form $\left[\begin{smallmatrix} a \\ b \end{smallmatrix}\right]$.

'Ket' Notation — traditionally,

- $|0\rangle : 1 \to 2$ denotes the vector $\left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$ which represents point $\underline{0}$ (a bit holding 0).
- $|1\rangle : 1 \to 2$ denotes the vector $\left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$ which represents point $\underline{1}$ (a bit holding 1).

$|\phi\rangle$ notation

Since $\begin{bmatrix} a \\ b \end{bmatrix} = a \begin{bmatrix} 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, the notation $a\,|0\rangle + b\,|1\rangle$ is normally used to denote **qubit** $\begin{bmatrix} a \\ b \end{bmatrix}$.

A qubit $2 \xleftarrow{a\,|0\rangle + b\,|1\rangle} 1$ expresses a quantum **superposition** of the two truth values.

Complex numbers $a, b \in \mathbb{C}$ are called **amplitudes** and are such that $a^2 + b^2 = 1$.

Given two qubits $1 \xrightarrow{u} 2$ and $1 \xrightarrow{v} 2$, $1 \xrightarrow{u \triangledown v} 2 \times 2$ denotes their **pairing**.

This leads to an extension of the 'ket' notation (next slide).

$|\phi\rangle$ notation and pairing

$$
\begin{array}{ll}
& |0\rangle \triangledown |1\rangle \\
= & \qquad \{ \text{ thinking functional helps here } \} \\
& \underline{0} \triangledown \underline{1} \\
= & \qquad \{ \text{ constant functions } \} \\
& \underline{(0,1)} \\
= & \qquad \{ \text{ vector notation } \} \\
& \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \\
= & \qquad \{ \text{ extended `ket' notation } \} \\
& |01\rangle
\end{array}
$$

$|\phi\rangle$ notation and pairing

More generally, the qubit pairing $(a\,|0\rangle + b\,|1\rangle) \triangledown (c\,|0\rangle + d\,|1\rangle)$ yields, once converted to vector notation

$$
\begin{array}{cl}
& \left[\frac{a}{b}\right] \triangledown \left[\frac{c}{d}\right] \\
= & \{\ \text{Khatri-Rao}\ \} \\
& \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} \\
= & \{\ \text{vector addition}\ \} \\
& \begin{bmatrix} ac \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ ad \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ bc \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ bd \end{bmatrix}
\end{array}
$$

that is, $ac\,|00\rangle + ad\,|01\rangle + bc\,|10\rangle + bd\,|11\rangle$.

Qubit entanglement

The qubit pair

$$2 \times 2 \xleftarrow{\frac{|00\rangle + |01\rangle}{\sqrt{2}}} 1$$

is a well-known example of **entaglement** – you get

$$fst \cdot \left(\frac{|00\rangle + |01\rangle}{\sqrt{2}}\right) = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)$$
$$snd \cdot \left(\frac{|00\rangle + |01\rangle}{\sqrt{2}}\right) = \left(\frac{|0\rangle + |1\rangle}{\sqrt{2}}\right)$$

but

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \triangledown \frac{|0\rangle + |1\rangle}{\sqrt{2}} = 2 \times 2 \xleftarrow{(\frac{!}{2})^{\circ}} 1$$

is different from the original $2 \times 2 \xleftarrow{\frac{|00\rangle + |01\rangle}{\sqrt{2}}} 1$.

Classic (quantum) control

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

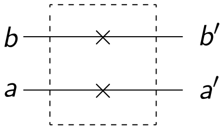
(Polymorphic) functional programming can play a nice role in quantum processing (perhaps not fully appreciated yet).

Think of the function $swap\ (a, b) = (b, a)$, that is, the **isomorphism**:

$$A \times B \xrightarrow{swap} B \times A \ = snd \triangledown fst.$$

For $A = B = 2$, this corresponds to the classical gate

		0	0	1	1
		0	1	0	1
0	0	1	0	0	0
0	1	0	0	1	0
1	0	0	1	0	0
1	1	0	0	0	1



SWAP Gates

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Applied to a qubit pair it will yield:

$$
\begin{array}{ll}
& swap \cdot (a\,|00\rangle + b\,|01\rangle + c\,|10\rangle + d\,|11\rangle) \\
= & \qquad \{\ \text{expland to vector notation}\ \} \\
& swap \cdot (a \begin{bmatrix}1\\0\\0\\0\end{bmatrix} + b \begin{bmatrix}0\\1\\0\\0\end{bmatrix} + c \begin{bmatrix}0\\0\\1\\0\end{bmatrix} + d \begin{bmatrix}0\\0\\0\\1\end{bmatrix}) \\
= & \qquad \{\ swap = snd \triangledown fst\ ;\ \text{vector addition}\ \} \\
& (snd \triangledown fst) \cdot \begin{bmatrix}a\\b\\c\\d\end{bmatrix} \\
= & \qquad \{\ \text{matrix-vector mutiplication; then back to } |\phi\rangle \text{ notation}\ \} \\
& a\,|00\rangle + c\,|01\rangle + b\,|10\rangle + d\,|11\rangle
\end{array}
$$

Quantum control

A well-known quantum gate is the **Hadamard gate**:

$$2 \xleftarrow{H} 2 \;=\; \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Applying this gate to qubit $u = a\,|0\rangle + b\,|1\rangle$:

$$\begin{tikzcd} 2 & 2 \arrow[l, "H"'] & 1 \arrow[l, "u"'] \arrow[ll, bend left, "H \cdot u"] \end{tikzcd}$$

Calculation:

$$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot (a\,|0\rangle + b\,|1\rangle) \;=\; \frac{1}{\sqrt{2}} \left(\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \cdot \left[\tfrac{a}{b}\right] \right) \;=$$
$$\frac{1}{\sqrt{2}} \begin{bmatrix} a+b \\ a-b \end{bmatrix} \;=\; \frac{a+b}{\sqrt{2}}\,|0\rangle + \frac{a-b}{\sqrt{2}}\,|1\rangle.$$

Summary

$$
\begin{array}{c}
\textit{Function} \quad (\mathbb{S}) \\
\big\downarrow \\
\textit{Probabilistic function} \quad (\mathbb{P}) \\
\big\downarrow {\scriptstyle \lfloor\cdot\rfloor} \\
\textit{Relation} \quad (\mathbb{R}) \\
\big\downarrow \\
\textit{Matrix} \quad (\mathbb{M})
\end{array}
$$

$$f = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$g = \begin{bmatrix} 1 & 0 & 0.5 & 0.1 \\ 0 & 0.7 & 0 & 0.9 \\ 0 & 0.3 & 0.5 & 0 \end{bmatrix}$$

$$f \subseteq R = \lfloor g \rfloor = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$\lfloor g \rfloor$ is called the **support** of g. Supports "convert" probabilistic functions into relations. Matrices in $\mathbb{M}$ can be unitary.

Pivot point

What is the **exact** meaning of the word “convert” in the previous slide?

This question also arises about matrix $[\![f]\!]$ (in category $\mathbb{M}$) “representing” function f in category $\mathbb{S}$.

The type of $[\![\_]\!]$ should be something like $\mathbb{S} \to \mathbb{M}$.

But $\mathbb{S}$ and $\mathbb{M}$ are **categories**, not mere **sets**.

This raises the need for **functors** — functions which “map arrows to arrows”.

Functors

Given categories $\mathbb{C}$ and $\mathbb{D}$, a functor $\mathbb{C} \xrightarrow{\mathfrak{F}} \mathbb{D}$ maps the arrows of $\mathbb{C}$ into the arrows of $\mathbb{D}$,

$$\mathbb{C}\,(a,b) \xrightarrow{\mathfrak{F}} \mathbb{D}\,(a,b)$$

$$\begin{tikzcd} \mathbb{C} \arrow[r, "\mathfrak{F}"] & \mathbb{D} \\ a \arrow[r, dotted, no head] \arrow[d, "f"'] & \mathfrak{F}\,a \arrow[d, "\mathfrak{F}\,f"] \\ b \arrow[r, dotted, no head] & \mathfrak{F}\,b \end{tikzcd}$$

such that

$$\mathfrak{F}\,id = id \tag{43}$$

$$\mathfrak{F}\,(g \cdot f) = (\mathfrak{F}\,g) \cdot (\mathfrak{F}\,f) \tag{44}$$

So $\mathfrak{F}$ "respects" the core structure of categories: **identity** and **composition**.

Functors

A well-known example of a functor in $\mathbb{S}$, dear to **functional programming**, is the operation which **maps** a function f over a list, $\mathfrak{F}\ f = f^*$ where $f^*\ x = [f\ a \mid a \leftarrow x]$, cf. the diagram

$$\begin{tikzcd} A \arrow[r, dotted, no head] \arrow[d, "f"'] & A^* \arrow[d, "f^*"] \\ B \arrow[r, dotted, no head] & B^* \end{tikzcd}$$

Clearly, (43) and (44) hold for this functor, that is,

$$id^*\ x = x$$
$$(g^* \cdot f^*)\ x = [(g \cdot f)\ a \mid a \leftarrow x] = (g \cdot f)^*$$

hold. Functor $\_^* : \mathbb{S} \rightarrow \mathbb{S}$ is an example of a **endo**-functor — a functor from a category to itself.

Bifunctors

Given in general three categories $\mathbb{C}$, $\mathbb{D}$ and $\mathbb{E}$, a **bifunctor** $\mathbb{C} \times \mathbb{D} \xrightarrow{\mathfrak{F}} \mathbb{E}$ is a binary functor

$$
\begin{tikzcd}
\mathbb{C} & \times & \mathbb{D} \arrow[r, "\mathfrak{F}"] & \mathbb{E} \\
a \arrow[d, "f"'] \arrow[r, dotted, no head] & c \arrow[d, "g"'] \arrow[r, dotted, no head] & \mathfrak{F}(a, c) \arrow[d, "\mathfrak{F}(f,g)"] \\
b \arrow[r, dotted, no head] & d \arrow[r, dotted, no head] & \mathfrak{F}(b, d)
\end{tikzcd}
$$

such that:

$$\mathfrak{F}(id, id) = id \tag{45}$$

$$\mathfrak{F}(h \cdot f, k \cdot g) = \mathfrak{F}(h, k) \cdot \mathfrak{F}(f, g) \tag{46}$$

Bifunctors

Wherever $\mathbb{C} = \mathbb{D} = \mathbb{E}$ we say $\mathfrak{F}$ is an **endo**-bifunctor.

Examples: in $\mathbb{M}$, **direct sum**

$$M \oplus N = \left[\begin{array}{c|c} M & 0 \\ \hline 0 & N \end{array}\right] \tag{47}$$

is an (endo)bifunctor defined by

$$M \oplus N \quad = \quad [i_1 \cdot M | i_2 \cdot N] \tag{48}$$

cf. the coproduct diagram

$$\begin{tikzcd} A \arrow[r, "i_1"] \arrow[d, "M"'] & A + B \arrow[d, "M \oplus N"] & B \arrow[l, "i_2"'] \arrow[d, "N"] \\ C \arrow[r, "i_1"] & C + D & D \arrow[l, "i_2"'] \end{tikzcd}$$

Bifunctors

Direct sum

Notably, direct sum (48) is present in all categories $\mathbb{M}$, $\mathbb{S}$ and $\mathbb{R}$, as coproducts of $\mathbb{S}$ lift to coproducts in $\mathbb{R}$:

$$
\begin{tikzcd}
A \arrow[r, "i_1"] \arrow[dr, "R"'] & A+B \arrow[d, "{[R,S]}"] & B \arrow[l, "i_2"'] \arrow[dl, "S"] \\
 & C &
\end{tikzcd}
\qquad
X = [R, S] \Leftrightarrow \left\{ \begin{array}{l} X \cdot i_1 = R \\ X \cdot i_2 = S \end{array} \right.
$$

Kronecker product

Also interesting is the fact that, in spite of not being a product, **pairing** in $\mathbb{M}$ leads to a bifunctor,

$$
M \otimes N \quad = \quad M \cdot \mathit{fst} \mathbin{\triangledown} N \cdot \mathit{snd} \tag{49}
$$

known as **Kronecker product**, which also extends to $\mathbb{S}$ and $\mathbb{R}$.

Composite Functors

As expected, functors compose with each other.

The most simple functors are the **identity** functor, which maps an arrow onto itself,

$$\mathfrak{I}\left(b \xleftarrow{f} a \right) \quad = \quad b \xleftarrow{f} a$$

and so-called **constant** functors: given an object k of a category $\mathbb{C}$, we define the constant functor $\mathfrak{K}$ as

$$\mathfrak{K}\left(b \xleftarrow{f} a \right) \quad = \quad k \xleftarrow{id} k$$

In $\mathbb{M}$ we will be particularly interested in the composite functor

$$\mathfrak{M}\, M \quad = \quad id \oplus M$$

which will be present in examples to follow.

Stop and think

Recall where we started from (broad picture):

- Divisibility ordering in $\mathbb{N}$ as example of a **reflexive** and **transitive** orders (**preorders**)
- We replaced each ordered pair by an **arrow** (witness)
- Thus preorders were "lifted" to **categories**.

In the same trend,

- what is the "lifting" of the concept of a **monotone function** between preorders, $a \leqslant b \Rightarrow (f\ a) \sqsubseteq (f\ b)$?

Well, we've just studied it:

Functors *between categories generalize* **monotone functions** *between preorders.*

Equations

Functors make it possible to think of solving **equations** in a categorial setting.

Starting point: we know that, given a **monotonic** function f we have techniques for solving the equation

$$x = f\ x$$

Above we have seen that **monotonic** functions between ordered structures scale up to **functors** between categories. So, what does the "categorial lifting" of $x = f\ x$,

$$x = \mathfrak{F}\ x$$

yield? Note that, in the CT setting, any solution to $x = \mathfrak{F}\ x$ is bound to be an **arrow**: what kind of arrow?

Functor equations

Let us draw a diagram for a candidate solution x_0,

$$\begin{tikzcd} A \arrow[d, "x_0"'] \\ B \end{tikzcd} \qquad \begin{tikzcd} \mathfrak{F}\ A \arrow[d, "\mathfrak{F}\ x_0"] \\ \mathfrak{F}\ B \end{tikzcd}$$

How do x_0 and $\mathfrak{F}\ x_0$ relate to each other?

We need to "bridge" them up,

$$\begin{tikzcd} A \arrow[r, "c"] \arrow[d, "x_0"'] & \mathfrak{F}\ A \arrow[d, "\mathfrak{F}\ x_0"] \\ B & \mathfrak{F}\ B \arrow[l, "a"] \end{tikzcd}$$

so that indeed there is an **equation** to solve:

$$x_0 = a \cdot (\mathfrak{F}\ x_0) \cdot c$$

In $\mathbb{S}$, x_0 can be regarded as a **recursive morphism** *— a* **program**.

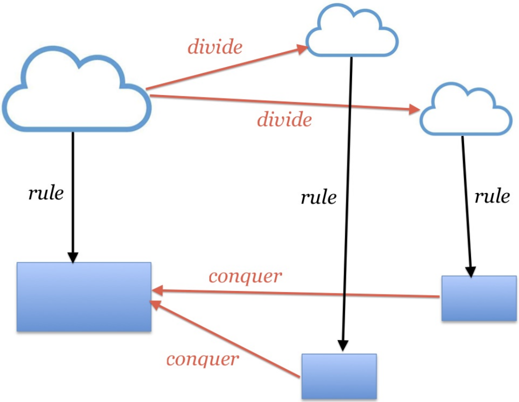
Questions: given a and c, does x_0 always exist? Is there a unique solution to $x_0 = a \cdot (\mathfrak{F}\ x_0) \cdot c$?

Divide & conquer programs

Compare diagram

$$
\begin{tikzcd}
A \arrow[r, "c"] \arrow[d, "x_0"'] & \mathfrak{F}\, A \arrow[d, "\mathfrak{F}\, x_0"] \\
B & \mathfrak{F}\, B \arrow[l, "a"]
\end{tikzcd}
$$

with the drawing aside.



This is how every **algorithm**, or **program** works...

(Dictionary) **Divide & rule** — *"the policy of maintaining control over subordinates by encouraging dissent between them"*.

Functor equations

In the $\mathbb{R}$ category:

As homsets form a complete Boolean algebras in $\mathbb{R}$, for monotonic $\mathfrak{F}$ the equation

$$x = a \cdot (\mathfrak{F}\ x) \cdot c$$

always has a least solution (Knaster-Tarski fixpoint theorem) termed **hylomorphism** *and denoted by* $[\![\ a, c\ ]\!]$.

In the $\mathbb{S}$ category:

$\mathbb{S}$ *is not so flexible because solutions have to be* **total** *functions. But for particular a and c we can find standard solutions in $\mathbb{S}$ termed* **catamorphisms** *and* **anamorphisms**, *as explained below.*

$\mathfrak{F}$ algebras and coalgebras

Terminology: in the equation

$$\begin{tikzcd} A \arrow[r, "c"] \arrow[d, "x_0"'] & \mathfrak{F}\, A \arrow[d, "\mathfrak{F}\, x_0"] \\ B & \mathfrak{F}\, B \arrow[l, "a"] \end{tikzcd}$$

- $B \xleftarrow{a} \mathfrak{F}\, B$ is referred to as an $\mathfrak{F}$-**algebra**
- $A \xrightarrow{c} \mathfrak{F}\, A$ is referred to as an $\mathfrak{F}$-**coalgebra**.

We will understand this terminology in a minute.

Before this, let us be aware that some $\mathfrak{F}$-(co)algebras are rather special.

$\mathfrak{F}$ algebras and coalgebras

A morphism $a \xrightarrow{f} b$ in a category $\mathbb{C}$ is an **isomorphism** if it has a "two-sided" inverse, namely another morphism $a \xleftarrow{g} b$ in the same category such that $g \cdot f = id$ and $f \cdot g = id$.

One way of recording such an isomorphism is by drawing

$$\begin{tikzcd} a \arrow[rr, bend left, "f"] \arrow[rr, phantom, "\cong"] & & b \arrow[ll, bend left, "g"] \end{tikzcd}$$

It can be shown that the isomorphisms in $\mathbb{R}$, for instance, are the functions whose converses are also functions — the so-called **bijections**.

$\mathfrak{F}$ algebras and coalgebras

To understand all this terminology, let us see an example in $\mathbb{R}$.

Take Peano's (1858-1932) definition of the natural numbers ($\mathbb{N}_0$):

- 0 is a natural number
- $n + 1$ is a natural number once n is so
- there are no more natural numbers.

We thus have a constant $0 \in \mathbb{N}_0$ and a natural number "factory" $succ : \mathbb{N}_0 \to \mathbb{N}_0$ such that $succ\ n = n + 1$.

0 can be represented by the constant function $zero\ x = 0$, of type $zero : A \to \mathbb{N}_0$, for some non-empty set A.

Peano algebra

Also note that *zero* and *succ* together generate a coproduct diagram:

$$\begin{tikzcd} A \arrow[r, "i_1"] \arrow[dr, "zero"'] & A + \mathbb{N}_0 \arrow[d, "{[zero, succ]}"] & \mathbb{N}_0 \arrow[l, "i_2"'] \arrow[dl, "succ"] \\ & \mathbb{N}_0 & \end{tikzcd}$$

Let us define functor $\mathfrak{F}\ X = A + X$, constant in A: $\mathfrak{F}\ f = id \oplus f$.

Re-draw the diagram using $\mathfrak{F}$:

$$\begin{tikzcd} A \arrow[r, "i_1"] \arrow[dr, "zero"'] & \mathfrak{F}\ \mathbb{N}_0 \arrow[d, "{[zero, succ]}"] & \mathbb{N}_0 \arrow[l, "i_2"'] \arrow[dl, "succ"] \\ & \mathbb{N}_0 & \end{tikzcd}$$

Thus $[zero, succ]$ is an $\mathfrak{F}$-**algebra**.

Yes! it packs the algebraic operators of $\mathbb{N}_0$ into a single arrow.

Peano algebra

The same algebra, rotated 90 degrees (in $\mathbb{R}$, cf. converse):

$$\begin{tikzcd} \mathbb{N}_0 \arrow[rr, bend left, "{[zero, succ]^\circ}"] \arrow[rr, phantom, "(?)"] & & \mathfrak{F}\,\mathbb{N}_0 \arrow[ll, bend left, "{[zero, succ]}"] \end{tikzcd}$$

Question: is $[zero, succ]$ an isomorphism?

- It is **surjective** — "there are no more natural numbers..."
- It is **not injective** — the A inputs are all ignored!

Clearly: it would be injective had A **only one** element...

Peano algebra

So we choose $A = 1$. Recall that 1 denotes a set with only one element (predefined, not relevant which one in particular).

$$\begin{tikzcd} \mathbb{N}_0 \arrow[rr, bend left, "{[zero,succ]^\circ}"] \arrow[rr, phantom, "\cong"] & & 1 + \mathbb{N}_0 \arrow[ll, bend left, "{[zero,succ]}"] \end{tikzcd}$$

By the way, object 1 discriminates $\mathbb{S}$ from both $\mathbb{R}$ and $\mathbb{M}$: the homset $\mathbb{S}\,(A, 1)$ is a singleton, a constant function which we have denoted by $! : A \to 1$.

In $\mathbb{R}$ the homset $\mathbb{R}\,(A, 1)$ contains many relations, all below $!$.

In $\mathbb{M}$ the homset $\mathbb{M}\,(A, 1)$ contains all row vectors with $\mid A \mid$-many columns.

Catamorphims

Back to diagram

$$\begin{tikzcd} A \arrow[r, "c"] \arrow[d, "x"'] & \mathfrak{F}\, A \arrow[d, "\mathfrak{F}\, x"] \\ B & \mathfrak{F}\, B \arrow[l, "a"] \end{tikzcd}$$

suppose **coalgebra** $c := \mathrm{in}^\circ$ exists as an isomorphism over the smallest possible object $A := I$:

$$\begin{tikzcd} I \arrow[r, bend left, "\mathrm{in}^\circ"] \arrow[r, phantom, "\cong"] & \mathfrak{F}\, I \arrow[l, bend left, "\mathrm{in}"] \end{tikzcd}$$

In this case, solution x **uniquely** depends on algebra a,

$$\begin{tikzcd} I \arrow[r, bend left, "\mathrm{in}^\circ"] \arrow[r, phantom, "\cong"] \arrow[d, "x"] & \mathfrak{F}\, I \arrow[l, bend left, "\mathrm{in}"] \arrow[d, "\mathfrak{F}\, x"] \\ B & \mathfrak{F}\, B \arrow[l, bend left, "a"] \end{tikzcd}$$

and we write $(\!|a|\!)$ to denote it in the corresponding **universal** property:

$$x = (\!|a|\!) \quad \Leftrightarrow \quad x \cdot \mathrm{in} = a \cdot \mathfrak{F}\, x$$

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Anamorphims

MAPi DOCTORAL PROGRAM IN COMPUTER SCIENCE

Back to diagram

$$\begin{tikzcd} A \arrow[r, "c"] \arrow[d, "x"'] & \mathfrak{F}\ A \arrow[d, "\mathfrak{F}\ x"] \\ B & \mathfrak{F}\ B \arrow[l, "a"] \end{tikzcd}$$

suppose **algebra** $a := \omega$ exists as an isomorphism over the largest possible $B := T$:

$$\begin{tikzcd} T \arrow[rr, bend left, "\omega^{\circ}"] \arrow[rr, phantom, "\cong"] & & \mathfrak{F}\ T \arrow[ll, bend left, "\omega"] \end{tikzcd}$$

In this case, solution x uniquely depends on coalgebra c,

$$\begin{tikzcd} A \arrow[rr, bend left, "c"] \arrow[d, "x"'] & & \mathfrak{F}\ A \arrow[d, "\mathfrak{F}\ x"] \\ T \arrow[rr, bend left, "\omega^{\circ}"] \arrow[rr, phantom, "\cong"] & & \mathfrak{F}\ T \arrow[ll, bend left, "\omega"] \end{tikzcd}$$

and we denote it by $[\![c]\!]$ in the corresponding **universal** property:

$$x = [\![a]\!] \quad \Leftrightarrow \quad x \cdot \omega^{\circ} = c \cdot \mathfrak{F}\ x$$

Summing up

- Programs can be of three different kinds, **catamorphisms**, **anamorphisms** or **hylomorphisms**.
- In $\mathbb{R}$, **initial** $\mathfrak{F}$-algebra I coincides with **terminal** $\mathfrak{F}$-algebra T and therefore the category has hylomorphisms,

$$[\![a, c]\!] = (\!|a|\!) \cdot [\!(c)\!]$$

- In $\mathbb{S}$, **initial** $\mathfrak{F}$-algebra I is smaller than **terminal** $\mathfrak{F}$-algebra T, and so $[\![a, c]\!] = (\!|a|\!) \cdot [\!(c)\!]$ is not always defined.
- $\mathbb{P}$ is "half way" between $\mathbb{R}$ and $\mathbb{S}$, but we need to study still another concept — that of a **monad** — to understand the relation between $\mathbb{S}$ and such categories.

Before this, some examples to help understand why the diagrams above are regarded as **programs**.

Peano programs (for-loops)

As earlier on, we play the game of adding arrows to diagrams and seeing what happens:

Add function f on $\mathbb{N}_0$:

$$\begin{tikzcd} \mathbb{N}_0 \arrow[rr, bend left, "{[zero,succ]^\circ}"] \arrow[rr, phantom, "\cong"] \arrow[d, "f"'] & & 1+\mathbb{N}_0 \arrow[ll, bend left, "{[zero,succ]}"] \\ B & & \end{tikzcd}$$

Since $\mathfrak{F}$ is a functor:

$$\begin{tikzcd} \mathbb{N}_0 \arrow[rr, bend left, "{[zero,succ]^\circ}"] \arrow[rr, phantom, "\cong"] \arrow[d, "f"'] & & 1+\mathbb{N}_0 \arrow[ll, bend left, "{[zero,succ]}"] \arrow[d, "id+f"] \\ B & & 1+B \end{tikzcd}$$

We can close the diagram provided we add another $(1+)$-**algebra** from $1+B$ to B (next slide).

Peano programs (for-loops)

Thus our first (recursive, but still abstract) program is born:

$$\begin{tikzcd} \mathbb{N}_0 \arrow[rr, bend left, "{[zero,succ]^\circ}"] \arrow[rr, phantom, "\cong"] \arrow[d, "f"'] & & 1+\mathbb{N}_0 \arrow[ll, bend left, "{[zero,succ]}"] \arrow[d, "id+f"] \\ B & & 1+B \arrow[ll, bend left, "g"] \end{tikzcd}$$

$$f = (\!|g|\!) \Leftrightarrow$$
$$f \cdot [zero, succ] = g \cdot (id + f)$$

Note that $g = [g_1, g_2]$, since it mediates a sum: $g_1 : 1 \to B$ will tell how the program **stops** while $g_2 : B \to B$ calls for **further** iterations.

An instance of this schema follows in the next slide.

Peano programs (for-loops)

Example: let $g = [zero, (n+)]$, where $(n+)\ x = n + x$, as expected. Then, using the universal property,

$$
\begin{array}{lll}
f = (\!|g|\!) & \Leftrightarrow & f \cdot [zero, succ] = [zero, (n+)] \cdot (id + f) \\
& \Leftrightarrow & \quad \{ \text{ fusion and absorption (coproducts in } \mathbb{S} \text{ or } \mathbb{R}) \ \} \\
& & [f \cdot zero, f \cdot succ] = [zero, (n+) \cdot f] \\
& \Leftrightarrow & \quad \{ \text{ coproduct equality } \} \\
& & \left\{ \begin{array}{l} f \cdot zero = zero \\ f \cdot succ = (n+) \cdot f \end{array} \right.
\end{array}
$$

Clearly, $f = (n\times)$. That is, we've synthesized the functional program

$$
\begin{array}{l}
n \times 0 = 0 \\
n \times (m + 1) = n + n \times m
\end{array}
$$

the same as: $n \times m = (\textbf{if } m = 0 \textbf{ then } 0 \textbf{ else } n + n \times (m - 1))$.

Peano programs (for-loops)

Another way to write the same program would be

$$(n\times) = \textbf{for}\ (n+)\ 0$$

by introducing a suggestive shorthand combinator

$$\textbf{for}\ g\ k = (\![\underline{k}, g]\!)$$

where $\underline{k}\ x = k$ denotes the constant function yielding k.

Example in Haskell:

```
*Nat> let mul n = for (n+) 0
*Nat> mul 34 23
782
```

Peano predicates

Another example for the same $\mathfrak{F}$, but in $\mathbb{R}$:

$$
\begin{array}{ll}
& (\geqslant) = (\!|[\top, succ]|\!) \\
\Leftrightarrow & \qquad \{ \text{ universal property (in } \mathbb{R}) \ \} \\
& (\geqslant) \cdot [zero, succ] = [\top, succ] \cdot (id + (\geqslant)) \\
\Leftrightarrow & \qquad \{ \text{ fusion, absorption, equality etc } \} \\
& \left\{ \begin{array}{l} (\geqslant) \cdot zero = \top \\ (\geqslant) \cdot succ = succ \cdot (\geqslant) \end{array} \right. \\
\Leftrightarrow & \qquad \{ \text{ introduce variables } \} \\
& \left\{ \begin{array}{l} n \geqslant 0 \Leftrightarrow true \\ n \geqslant m + 1 \Leftrightarrow \exists\, k : n = k + 1 \wedge k \geqslant m \end{array} \right.
\end{array}
$$

Cf. primitive induction.

Why monads

So far we have been able to encode, in instances of the CT framework, **neat** constructs and **elegant** programs.

What about **“dirty”** programs, that is, those which produce **side-effects** ?

What about **imperative** ones?

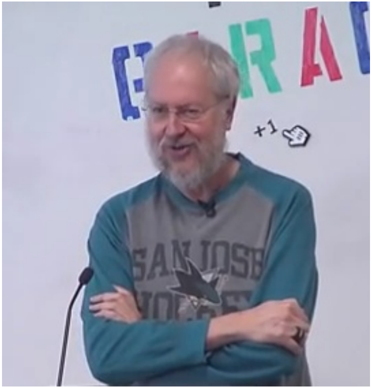
And what about **“faulty”** programs, that is, those which misbehave, e.g. because they run on defective hardware?

We need another CT concept, and a very relevant one — that of a **monad**. Our last topic in this module.

The monadic "curse"

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

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



Douglas Crockford (2013)

(https://www.youtube.com/watch?v=b0EF0VTs9Dci)

Why monads

Some patterns of arrow **composition** don't work because the output are "$\mathfrak{F}$-times" more complex than expected, e.g.

$$
\begin{tikzcd}
 & \mathfrak{F}\,B \arrow[d, dotted, no head] & A \arrow[l, "g"'] \arrow[dll, bend left] \\
\mathfrak{F}\,C & B \arrow[l, "f"'] &
\end{tikzcd}
\tag{50}
$$

Let e.g. $\mathfrak{F}\,B = E + B$ record the fact that g fails for some outputs, raising an **exception** in E, otherwise yielding a B.

In general, $\mathfrak{F}\,B$ (and $\mathfrak{F}\,C$ etc) carry some information about a **computational effect** which we have to **handle** but would like (technically) to **ignore**...

Example

Declare in Haskell ($\mathbb{S}$):

$$g\ a = [a+1, a-1]$$
$$f\ a = [\sqrt{a}, -\sqrt{a}]$$

This defines two arrows:

$$g :: Num\ t \Rightarrow t \rightarrow [t]$$
$$f :: Floating\ t \Rightarrow t \rightarrow [t]$$

which do not compose. We search for a **new** form of arrow composition $f \bullet g$ such that e.g.

$$(f \bullet g)\ 3 = [2.0, -2.0, 1.414213562, -1.414213562]$$

Output yields the square roots of the two natural numbers centered at 3.

Example

Programming $f \bullet g$:

$$f \bullet g\ a = concat\ [f\ b \mid b \leftarrow g\ a]$$

where $concat :: [[a]] \rightarrow [a]$ concatenates a list of lists.

This works because, in Haskell, **lists** form a **monad**

$\mathfrak{F}\ x = x^*$ *is not only a* **functor** *but also a* **monad**.

Our purpose in the slides to follow is to generalize $\mathfrak{F}\ x = x^*$ to other monads.

Remember that CT as a whole is based on two core notions:

- **composition** of arrows
- **identity** arrows.

Monads

So the way to go about the $\mathfrak{F}$-**inflated** arrows of (50) has to devise a form of composition and an identity.

For this we need a CT construction known as a **monad**:

Let $\mathfrak{F}$ be an (endo)functor in some category $\mathbb{C}$, such that the following arrows always exist, for any X:

$$X \xrightarrow{\eta} \mathfrak{F}\ X \xleftarrow{\mu} \mathfrak{F}^2\ X$$

subject to a number of **properties** *left out for the moment.*

Why are such arrows useful?

Monads

They enable us to *complete diagram* (50),

$$\begin{tikzcd}
\mathfrak{F}^2\, C \arrow[d, "\mu"'] & \mathfrak{F}\, B \arrow[l, "\mathfrak{F}\, f"'] \arrow[d, dotted, no head] & A \arrow[l, "g"'] \arrow[dll, bend left, "f \bullet g"] \\
\mathfrak{F}\, C & B \arrow[l, "f"'] &
\end{tikzcd}$$

where μ copes with the **nesting** of effects (exceptions on top of exceptions, for instance). The other arrow, $X \xrightarrow{\eta} \mathfrak{F}\, X$, converts a **pure** value into an effectful one.

Thus $f \bullet g$ can be regarded as a form of (monadic) **composition**.

Monads

Indeed, the monadic properties (which we once again skip for brevity) grant the expected properties, with η behaving as identity:

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

Now suppose $\mathfrak{F}\, X = X$, the identity functor:

$$\begin{tikzcd}
C \arrow[d, "id"'] & B \arrow[l, "f"'] \arrow[d, dotted, no head] & A \arrow[l, "g"'] \arrow[dll, bend left=60, "f \bullet g = f \cdot g"] \\
C & B \arrow[l, "f"'] &
\end{tikzcd}$$

Conclude that we have been working in a monad since the very beginning of this course — the **identity** monad!

The list monad

A monad instance dear to functional programmers is the **list** monad :

$$\mu\,[\,] = [\,]$$
$$\mu\,(l : t) = l +\!\!+\, \mu\, t$$

where $\mu = \mathit{concat}$ — list concatenation — and

$$\eta\, a = [a]$$

builds singleton lists.

If we ignore the ordering of elements in a list, this monad mimics bounded **non-determinism**. See the next slide for an evolution of this idea.

The (finite) powerset monad

The monad

$$X \xrightarrow{\eta} \mathfrak{P}\, X \xleftarrow{\mu} \mathfrak{P}^2\, X$$

is *par excellence* the one behind non-deterministic finite automata (NFSA), where $\mathfrak{P}\, X = \{\, S \mid S \subseteq X \,\}$. Its components are

$$\begin{array}{l} \mu\, \{\,\} = \{\,\} \\ \mu\, (\{l\} \cup t) = l \cup \mu\, t \end{array}$$

— union of a set of sets — and

$$\eta\, x = \{x\}$$

which builds singleton sets.

Monads pointwise

Here is a way of writing $f \bullet g$ in a (generic) pointwise manner

$$(f \bullet g)\ a = \mathbf{do}\ \{ b \leftarrow g\ a; \mathit{return}\ (f\ b) \}$$

where *return* is a synonym for η popular in monadic languages such as e.g. Haskell.

Likewise,

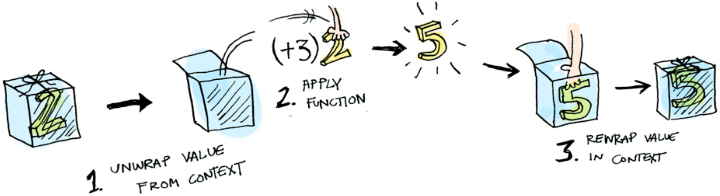
$$\mathbf{do}\ \{ a \leftarrow x; \mathit{return}\ (g\ a) \}$$

denotes the *application* of $A \xrightarrow{g} \mathfrak{F}\ B$ to monadic data x.

A final example: $\mathbf{do}\ \{ a \leftarrow x; b \leftarrow y; \mathit{return}\ (x + y) \}$ adds two numbers extracted from monadic data.

Monads pointwise

Here is a cartoon



for the calculation of $\mathfrak{F}\ (+3)\ x$, where $x = \mathit{return}\ 2$ is the monadic object which contains number 2 in monad $\mathfrak{F}$:

$$\textbf{do}\ \{a \leftarrow \mathit{return}\ 2; \mathit{return}\ (a+3)\}$$

Kleisli category

We observe that properties

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

where

$$f \bullet g = \mu \cdot (\mathfrak{F}\ f) \cdot g$$

offer a sub-category of $\mathbb{C}$, that made of $\mathfrak{F}$-inflated arrows only, that is, homsets of pattern $\mathbb{C}\ (A, \mathfrak{F}\ B)$.

Such a sub-category (usually denoted by $\mathbb{C}^\flat$) is known as the **Kleisli** category associated to monad $\mathfrak{F}$ in $\mathbb{C}$, as follows:

$$\mathbb{C}^\flat\ (A, B) \quad \cong \quad \mathbb{C}\ (A, \mathfrak{F}\ B)$$

Kleisli category

The best known example of a Kleisli category is $\mathbb{R} = \mathbb{S}^\flat$, induced by monad $\mathfrak{P}$:

$$\mathbb{R}\,(A, B) \quad \cong \quad \mathbb{S}\,(A, \mathfrak{P}\, B)$$

This simply tells that every **relation** can be represented by a **set**-valued function.

It can also be shown that $\mathbb{M}$ and $\mathbb{P}$ can be regarded as Kleisli categories of suitable **monads** in $\mathbb{S}$.

This is why $\mathbb{S}$ is, for many people, *"the category par excellence"*.

Wrapping up

The application of the two **CT** concepts of **functor** and **monad** to **programming** is perhaps the most significant development in the software sciences for the last 30 years.

To program with them one needs to know about **CT**, the **lingua franca** of software science.

Many useful monads can be found in the literature.

Haskell is among the languages that first incorporated **functors** and **monads** into themm.[2]

Python, Scala, Swift, F# have got there too; Java8 seems to have tried.

[2] See e.g. `https://hackage.haskell.org/package/base-4.9.0.0/docs/Control-Monad.html`.

Postlude

Sir Arthur Eddington (1882-1944):

> *"I cannot believe that anything so ugly as multiplication of matrices is an essential part of the scheme of nature"*

(in *"Relativity Theory of Electrons and Protons"*, 1936).

Serious warning to mathematicians and physicists — **notations** should be **elegant** :-)

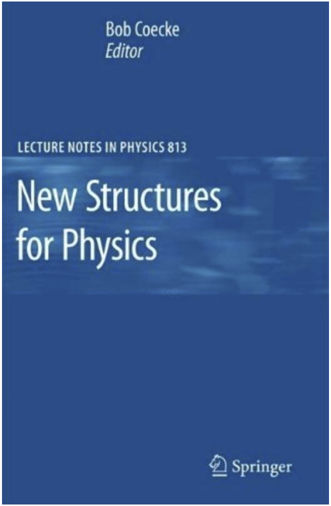
I agree — standard **linear algebra notation** is clumsy by **modern computer science** standards.

Postlude

Unfortunately, Sir A. Eddington did not live long enough to find the following answer to his complaint,

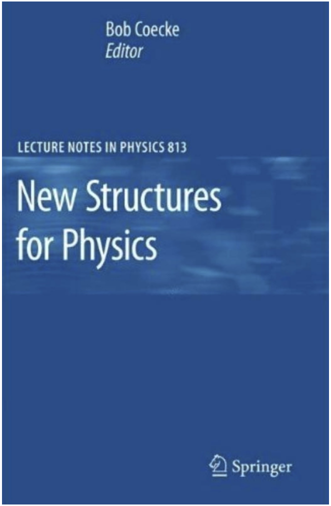
> **“New Structures for Physics”**, **Lect. Notes in Physics** *volume 813*

compiled by B. Coecke.



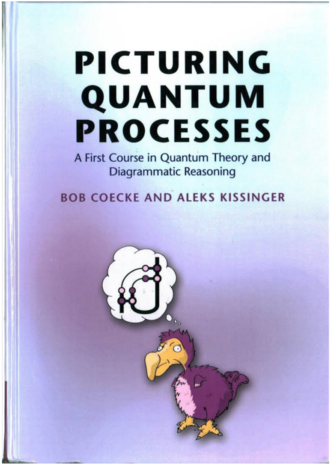
Postlude

The generic structures (**monoidal categories**) in which **quantum physics** are expressed in this book generalize the categories $\mathbb{R}$ and $\mathbb{P}$ that we have studied in this module.



Postlude

More recently, a quite accessible book by B. Coecke and Aleks Kissinger:



Local setting (HASLab)

This **unified way** of thinking has been a subject of research at the HASLab (INESC TEC & U.Minho) laboratory for quite a while, covering **application areas** as disparate as e.g.

- **Data mining** — (Macedo and Oliveira, 2015; Oliveira and Macedo, 2017)
- **Component-oriented** programming — (Oliveira and Miraldo, 2016)
- **Fault** propagation — (Oliveira, 2012)
- Managing **risk** in functional programming — (Murta and Oliveira, 2015)
- **Weighted automata** — (Oliveira, 2013)
- **Linear algebra** — (Macedo and Oliveira, 2013)

Textbooks

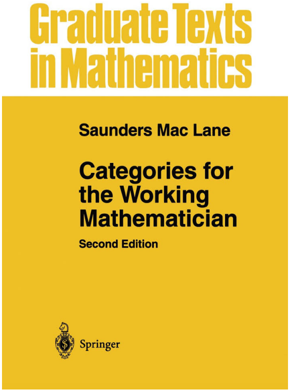
Having a look at one (or more!) of these textbook is highly recommended:

The standard *"bible"* on category theory: (MacLane, 1971), aside.

An enjoyable introduction to the same field: (Lawvere and Schanuel, 1997)

Another good book for computer scientists: (Pierce, 1991)

The standard "algebra of programming" textbook: (Bird and de Moor, 1997)



(There is much more on the web — just search for "Category Theory textbook").

References

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

B. Coecke, editor. *New Structures for Physics*. Number 831 in Lecture Notes in Physics. Springer-Verlag, 2011.

B. Lawvere and S. Schanuel. *Conceptual Mathematics: a First Introduction to Categories*. Cambridge University Press, 1997. ISBN 0521472490.

H.D. Macedo and J.N. Oliveira. Typing linear algebra: A biproduct-oriented approach. *SCP*, 78(11):2160–2191, 2013.

H.D. Macedo and J.N. Oliveira. A linear algebra approach to OLAP. *FAoC*, 27(2):283–307, 2015.

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

D. Murta and J.N. Oliveira. A study of risk-aware program transformation. *SCP*, 110:51–77, 2015.

M.A. Nielsen and I.L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, New York, NY, USA, 10th edition, 2011. ISBN 1107002176, 9781107002173.

J. N. Oliveira and H. D. Macedo. The data cube as a typed linear algebra operator. In *Proc. of the 16th Int. Symposium on Database Programming Languages*, DBPL '17, pages 6:1–6:11, New York, NY, USA, 2017. ACM.

J.N. Oliveira. Towards a linear algebra of programming. *FAoC*, 24(4-6): 433–458, 2012.

J.N. Oliveira. Weighted automata as coalgebras in categories of matrices. *Int. JFCS*, 24(06):709–728, 2013.

J.N. Oliveira and V.C. Miraldo. "Keep definition, change category" — a practical approach to state-based system calculi. *JLAMP*, 85(4): 449–474, 2016.

B. Pierce. *Basic category theory for computer scientists*. Foundations of computing. MIT Press, 1991. ISBN 978-0-262-66071-6.