Quantamorphisms and Quantum Program Calculi

J.N. Oliveira



DIUM



HASLab/INESC TEC

Seminars on Physics of Information (Course: EFA)

6th November 2026, Room E7-0.09
U.Minho, Braga, Portugal

The long title...

Quantamorphisms: towards quantum controlled recursion by calculation

Main reference

Journals & Magazines > IEEE Transactions on Software... > Volume: 48 Issue: 11

Compiling Quantamorphisms for the IBM Q Experience

Publisher: IEEE

Cite This PDF

Ana Neri ; Rui Soares Barbosa ; José N. Oliveira All Authors

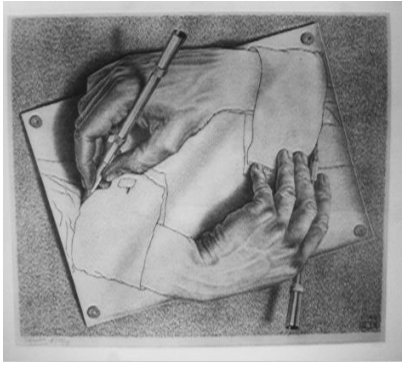
Context

Computing
Quantum programs (QP)
Recursive QP
Quantum-controlled recursive QP

Recursion

"(...) To **iterate** *is human, to* **recurse** *[is] divine."*

(Quote attributed to Peter Deutsch, the creator of the GhostScript interpreter.)

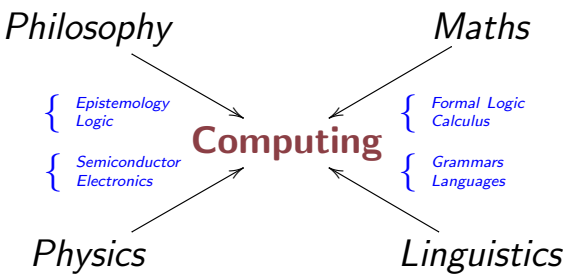


Drawing hands (1948) by M.E. Escher (1898-1972)

Part I

Classical computing

Blend of diverse bodies of knowledge:





(Source: The Computer Museum, USA)

90 years ago

1936

A. Turing — **abstract** *notion of what we now call a* **programmable computer** — *known as the* **Turing machine**.

1936

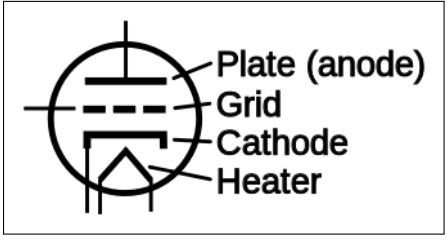
A. Church — λ*-calculus, the basis of* **functional** *programming.*



A. Turing (1912-1954)

Meanwhile

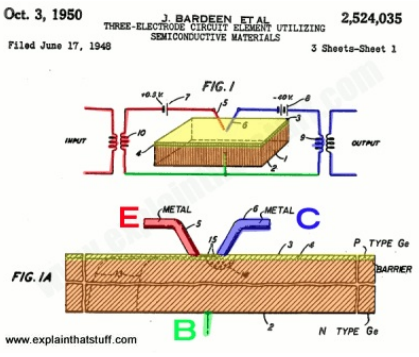




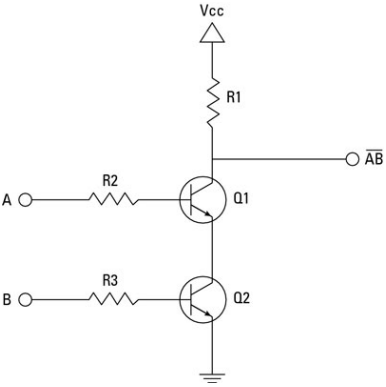
Vacuum tubes, triodes (1912)

(Credits: https://en.wikipedia.org/wiki/Triode)

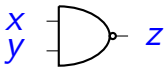
Transistors (1948)



Formal Logic becomes... physical 

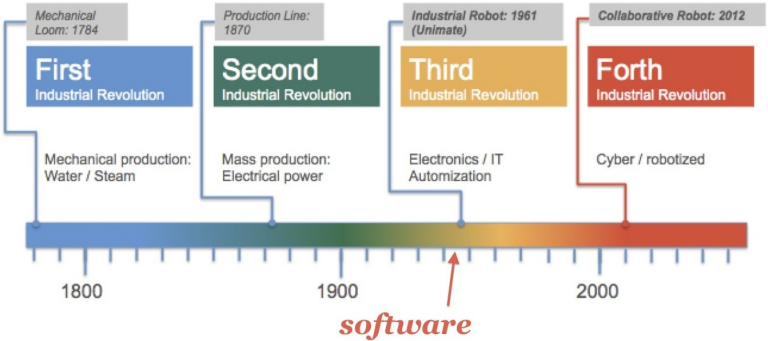


The "glorious **NAND** era":



$$\begin{cases} nand : \mathbb{B} \times \mathbb{B} \to \mathbb{B} \\ nand\ (x, y) = \neg\ (x \wedge y) \end{cases}$$

1950s = Hardware + software



(Adapted from nelmio Robotics Insight 2015)

Not everybody happy... (1972)

"(...) The major cause of the **software crisis** *is that the machines have become several orders of magnitude more powerful! (...)*

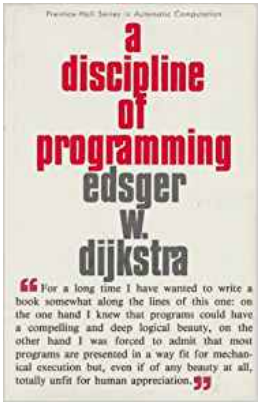
[Now that we] have gigantic computers, **programming** *has become an equally gigantic* **problem**.*"*

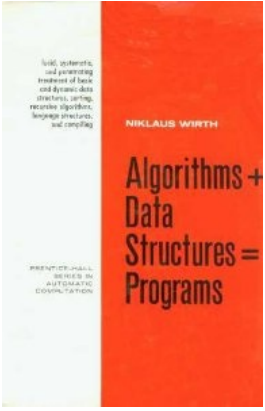
In EWD340 (1972).

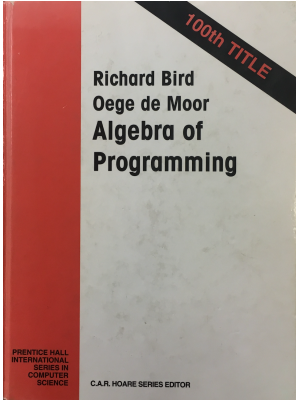


E.W. Dijkstra (1920-2002)

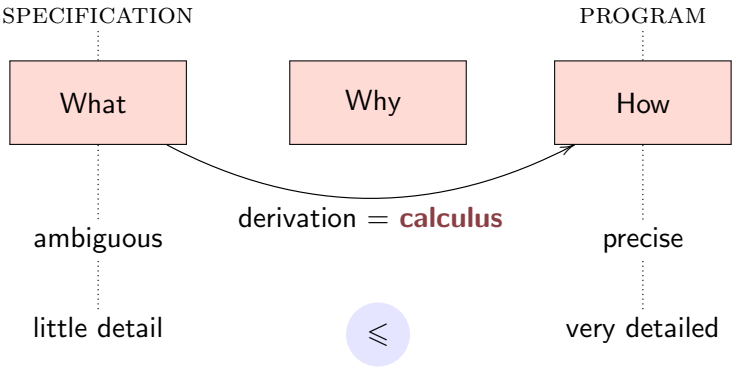
Software theory golden age (1976-96)







Formal methods (1970s - today)

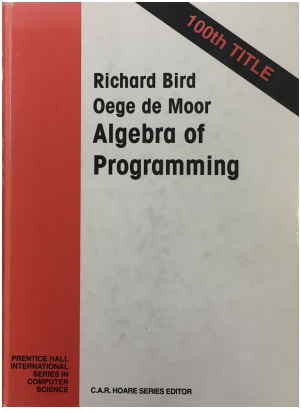


Algebra of Programming

Program calculation via **algebraic** manipulation.

Focus on **recursive** programs.

Which kind of **algebra**?



Algebra of Programming

Specifications

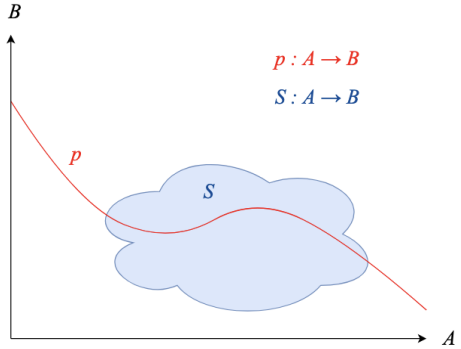
- Binary **relations** (S)

Programs

- **functions** (p)

Main practical difference:

- Relations are **checkable** (only)
- Functions are **executable**.



Algebra of Programming

Formal correctness:

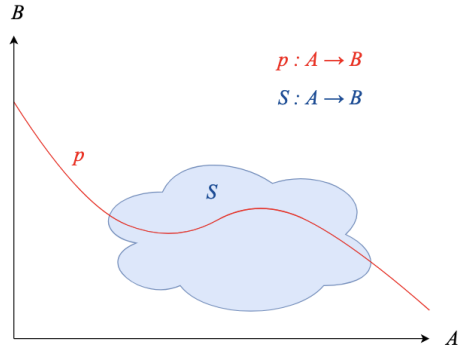
$$p \vdash S$$

that is,

$$\langle \forall\ a\ :\ \langle \exists\ b\ ::\ b\ S\ a \rangle :\ (p\ a)\ S\ a \rangle$$

In words,

> *wherever the* **specification** S *prescribes behaviour, the* **implementation** p *should yield one such prescribed behaviour.*



Algebra of Programming

The 'Boolean matrix' view of

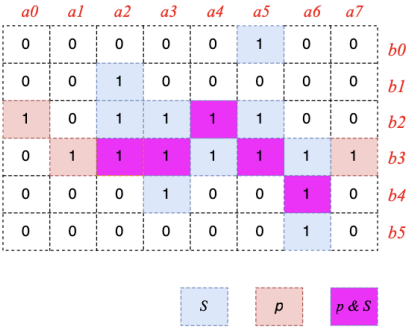
$$\langle \forall\ a\ :\ \langle \exists\ b\ ::\ b\ S\ a \rangle :\ (p\ a)\ S\ a \rangle$$

is illustrated aside.

NB: mind the **infix** notation $b\ S\ a$, instead of e.g. $(a, b) \in S$ etc.

In the matrix view,

- $b\ S\ a = 1$ iff the relationship holds
- $b\ p\ a = (b = p\ a)$ — since p is a function.



Quantum (40 years ago)

1985

> *David Deutsch (U. Oxford) describes the first universal* **quantum computer**.

Nowadays

> *Physics making it happen, again...*

What about **Quantum software**?



Source: IBM Q Experience website

Quantum programs

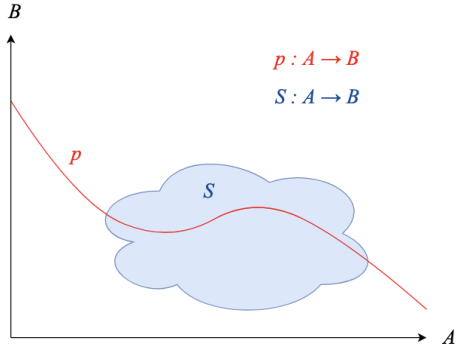
Question:

> *Do our good old* **classical** *programs run on* **quantum** *hardware too?*

No! Only some do so.

Sorry:

- No longer in the *glorious* **'NAND** *era*...
- **Formal logic**, **relations**, **functions**... too old fashioned.



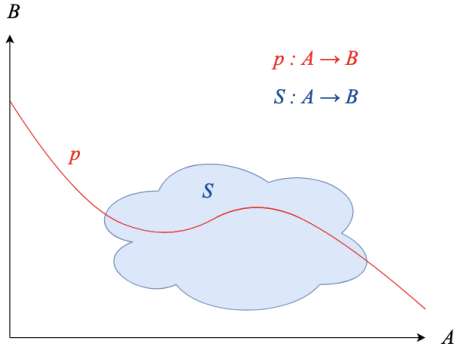
Quantum programs

Instead:

> *Hilbert spaces. Matrices and tensors. Unitary computations. That's what the new world is made of...* ☺

A provocative question:

> *Hmmm... is it all* **that different***?*



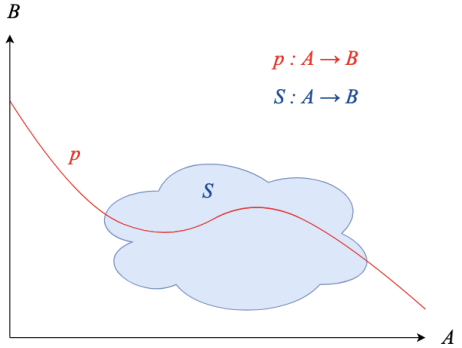
Quantum programs

Instead:

> *Hilbert spaces. Matrices and tensors. Unitary computations. That's what the new world is made of...* 

A provocative question:

> *Hmmm... is it all* **that different***?*



A kind of warning (1948)

"(...) [**Formal logic** *is] one of the technically most* **refractory** *parts of mathematics".*

The reason for this is that it deals with rigid, **all-or-none** *concepts, and has very little contact with the* **continuous** *concept of the real or of the complex number, that is, with* **mathematical analysis**.

(Talk at the Hixon Symposium on September 20, 1948, in Pasadena, California.)



John von Neumann (1903-1957)

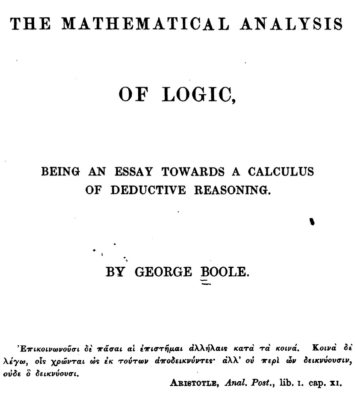
However, 101 years back (1847) ...

Formal Logic's birth certificate:

> *The Mathematical Analysis of Logic (...) (1847),*

Contrary to von Neuman criticism,

> *Boole's initial proposal was centered on a* **quantitative** *approach to Logic, rooted in* **Calculus***!*



THE MATHEMATICAL ANALYSIS

OF LOGIC,

BEING AN ESSAY TOWARDS A CALCULUS
OF DEDUCTIVE REASONING.

BY GEORGE BOOLE.

Ἐπικοινωνοῦσι δὲ πᾶσαι αἱ ἐπιστῆμαι ἀλλήλαις κατὰ τὰ κοινά. Κοινὰ δὲ λέγω, οἷς χρῶνται ὡς ἐκ τούτων ἀποδεικνύντες· ἀλλ' οὐ περὶ ἃ δεικνύουσιν, οὐδὲ ὃ δεικνύουσι.

ARISTOTLE, *Anal. Post.*, lib. I. cap. XI.

Back to 1847

For $x, y \in \{0, 1\}$, George Boole defines:

$$NOT(x) = \neg x = 1 - x$$

$$AND(x, y) = x \wedge y = xy$$

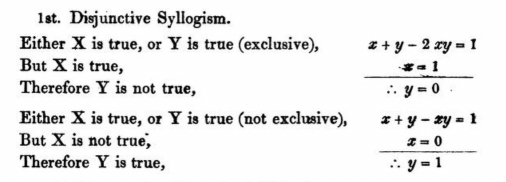
$$OR(x, y) = x \vee y = x + y - xy$$

and thus

$$NAND(x, y) = 1 - xy$$

$$XOR(x, y) = x + y - 2xy$$

and so on.



1st. Disjunctive Syllogism.

Either X is true, or Y is true (exclusive), $x + y - 2xy = 1$
But X is true, $x = 1$
Therefore Y is not true, $\therefore y = 0$

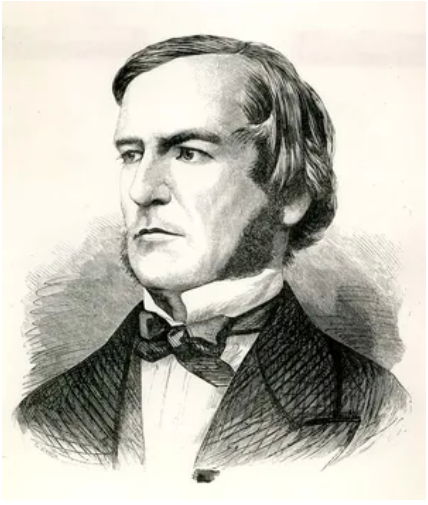
Either X is true, or Y is true (not exclusive), $x + y - xy = 1$
But X is not true, $x = 0$
Therefore Y is true, $\therefore y = 1$

Formal Logic = Calculus (1847)

"(...) The laws we have established under the symbolical forms $x\,(u+v) = x\,u + xv$ (...) are sufficient for the basis of a **Calculus** *(...) properties which they possess in common with symbols of* **quantity**, *and in virtue of which, all the processes of common* **algebra** *are applicable to the present system."*

This makes the history of Logic all the more interesting!

Implications?



George Boole (1815-1864)

Let us scale it up — the Fuzzy Logic "zoo"

Just work with the **whole interval** $[0, 1]$:

t-norm	$x \sqcap y$	$x \sqcup y$
Gödelian	$\min(x, y)$	$\max(x, y)$
Łukasiewicz	$\max(0, x + y - 1)$	$\min(x + y, 1)$
Algebraic product	$x \cdot y$	$x + y - x \cdot y$
Drastic	$\begin{cases} x & \text{if } y = 1 \\ y & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases}$	$\begin{cases} x & \text{if } y = 0 \\ y & \text{if } x = 0 \\ 1 & \text{otherwise} \end{cases}$
Einstein	$\frac{x \cdot y}{1+(1-x)\cdot(1-y)}$	$\frac{x+y}{1+x \cdot y}$

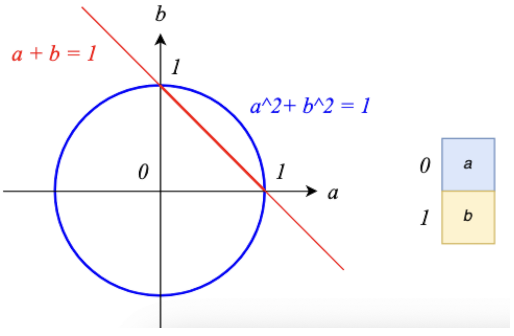
(Algebraic product **reinventing** Boole?)

Let us scale it up — Probabilistic Functions

	a0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
b0						0.1		
b1			0.1					
b2	1		0.7	0.3	0.5	0.1		
b3		1	0.2	0.3	0.5	0.8	0.25	1
b4				0.4			0.7	
b5							0.05	

Nicely captured by stochastic **matrices**.

Let us scale it up — Quantum



Nicely captured by (unitary) **matrices** of **complex** numbers.

You said... matrices??

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).



You said... Hilbert spaces??

Again John von Neumann, excerpt from a letter to Garrett Birkhoff, 1935:

> *"I would like to make a confession which may seem immoral: I do not believe absolutely in* **Hilbert space** *any more."*

(Quote taken from [1], chapter 5.)



John von Neumann (1903-1957)

Formal Logic



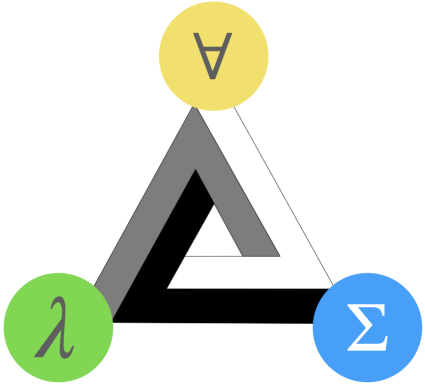
Calculus

Physics

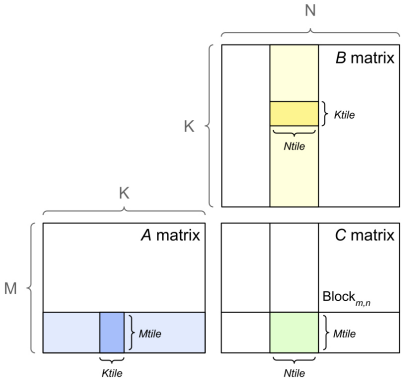
Towards a 'Lingua franca'

Not to displease either von Neumann or Eddington, we will work in a way that:

- disguises Hilbert spaces under a simpler framework that
- encompasses **functions**, **relations** and **matrices** under an **unified** notation.

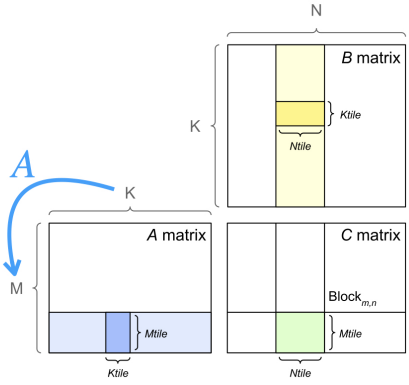


Regarding Eddington's disbelief...



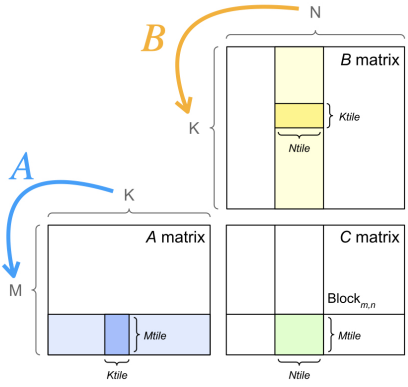
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Regarding Eddington's disbelief...



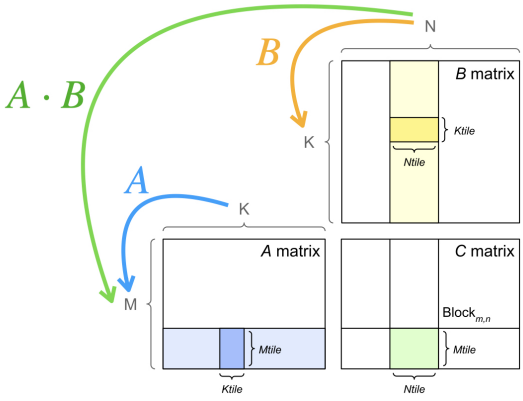
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Regarding Eddington's disbelief...



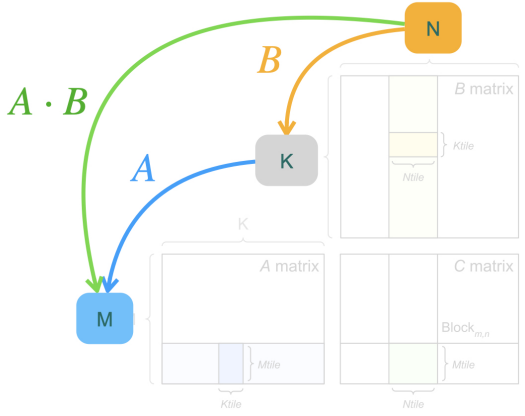
Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication

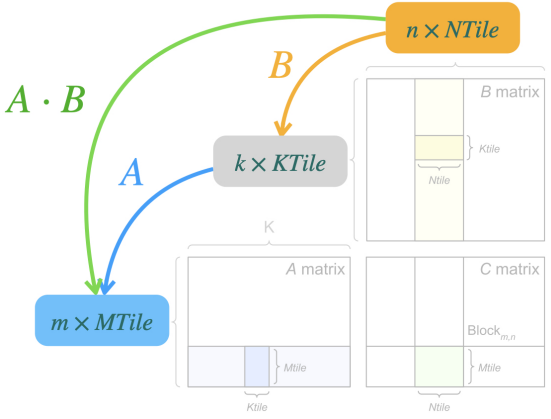


‘Matrices as *arrows*’



Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Matrix multiplication



‘Tensors as arrows’ etc

Credits: https://docs.nvidia.com/deeplearning/performance/dl-performance-matrix-multiplication/index.html

Categorial approach to Linear Algebra (2013)

ELSEVIER

Science of Computer Programming

Volume 78, Issue 11, 1 November 2013, Pages 2160-2191



Typing linear algebra: A biproduct-oriented approach

Hugo Daniel Macedo, José Nuno Oliveira

Show more

Categorial approach to Linear Algebra (2020)

RESEARCH-ARTICLE | FREE ACCESS |

Type your matrices for great good: a Haskell library of typed matrices and applications (functional pearl)

Authors: Armando Santos, José N. Oliveira Authors Info & Claims

Haskell 2020: Proceedings of the 13th ACM SIGPLAN International Symposium on Haskell • Pages 54 - 66
https://doi.org/10.1145/3406088.3409019

Published: 09 August 2020 Publication History

Check for updates

Related Artifact: Linear Algebra of Programming Library • July 2020 • software • https://doi.org/10.1145/3410239

Feedback

Categories of matrices — dimensions as types

Objects are matrix dimensions (natural numbers) regarded as **types**. Such types denote initial segments:

$$n = \{0\,..\,n-1\} \tag{1}$$

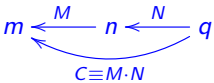
Thus

$0 = \{\}$ — the **empty** type
$1 = \{0\}$ — the **singleton** type
$2 = \{0, 1\}$ — the **Booleans** ...and so on.

Morphisms $n \xleftarrow{M} m$ are **matrices** typed by such numbers. **Identities** are (square) identity matrices, $n \xleftarrow{id} n$.

Categories of matrices — matrices as morphisms

Composition



(2)

is matrix-matrix multiplication:

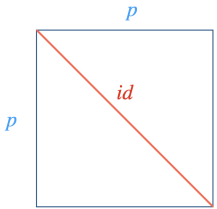
$$c\ (M \cdot N)\ a \quad = \quad \langle \Sigma\ b\ ::\ c\ M\ b \times b\ N\ a \rangle \tag{3}$$

NB: *note the infix notation for matrix cells, e.g.* $b\ M\ a$, *instead of the more usual* $M_{b\ a}$.

The **identity**

$$p \xrightarrow{id} p$$

is a **polymorphic** matrix and so $id \cdot M = M = M \cdot id$ holds.



Transposition etc

Converses exist as matrix transpositions, $a\ M^\circ\ b = b\ M\ a$.

The following laws hold: idempotence

$$(M^\circ)^\circ = M$$

and contravariance:

$$(M \cdot N)^\circ = N^\circ \cdot M^\circ$$

Vectors: column (resp. row) **vectors** are matrices of generic type $m \xleftarrow{v} 1$ (resp. $1 \xleftarrow{v} m$).

Index notation $v\,[i]$ will be used to abbreviate vector cells, eg. $(i\ v\ 0)$ written $v\,[i]$.

Categories of matrices — monoidal structures

Matrix **addition**:

$$c\ (M + N)\ a \quad = \quad c\ M\ a + c\ N\ a \tag{4}$$

Hadamard product:

$$c\ (M \times N)\ a \quad = \quad c\ M\ a \times c\ N\ a \tag{5}$$

Scalar product:

$$c\ (\alpha\ M)\ a \quad = \quad \alpha\ (c\ M\ a) \tag{6}$$

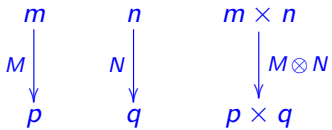
(Some underlying algebraic field assumed, typically $\mathbb{R}$ or $\mathbb{C}$.)

Product types

Product types:

$$n \times m = \{(b, a) \mid b \in n, a \in m\} \tag{7}$$

Tensor (Kronecker) product:



$$(y, z)\ (M \otimes N)\ (b, a) \equiv (y\ M\ b) \times (z\ N\ a) \tag{8}$$

NB: (8) generalizes **relational** product. But..., what is a **relation**?

Unifying notations

Relations are Boolean matrices, meaning that their cells live in $2 = \{0, 1\}$.

Boolean operations (over 2) are defined **arithmetically**, as advocated by Boole:

$$a \wedge b = ab$$
$$\neg\, a = 1 - a$$
$$a \vee b = a + b - ab$$
$$a \oplus b = a + b - 2\, ab$$

Moreover:

$$a \Rightarrow b = a \leqslant b$$

Thus 0 (resp. 1) is the unit of disjunction (resp. conjunction).

Unifying notations

Boolean 2-matrices can be regarded as binary **predicates**.

For instance, the matrix $n \xleftarrow{(\leqslant)} n$ captures the comparison of objects of type n.

NB: *infix notation $j\ M\ i$ comes in handy here — writing e.g. $(\leqslant)_{j\,i}$ instead of $i \leqslant j$ would be rather unnatural.*

Units: $n \xleftarrow{0} m$ (resp. $n \xleftarrow{1} m$) is the unit of $M + N$ (resp. $M \times N$), where $n \xleftarrow{0} m$ (resp. $n \xleftarrow{1} m$) is the smallest (resp. largest) 2-matrix of its type.

Row vector $1 \xleftarrow{1} m$ is often denoted by $!$ and is termed "*bang*".

Unifying notations

Functions (denoted by lowercase letters) are special cases of relations — exactly those such that:

$$y \; [\![f]\!] \; x = 1 \quad \equiv \quad y = f \; x \tag{9}$$

$[\![f]\!]$ denotes the matrix that represents the graph of f. Examples:

$$\left\{ \begin{array}{l} \neg : 2 \to 2 \\ \neg \; x = 1 - x \end{array} \right. \quad \equiv \quad 2 \xleftarrow{[\![\neg]\!]} 2 \; = \; \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$$

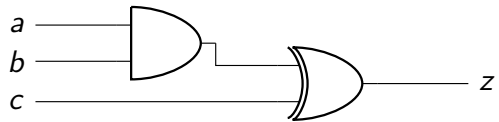
$$\left\{ \begin{array}{l} \oplus : 2 \times 2 \to 2 \\ x \oplus y = x + y - 2\,xy \end{array} \right. \quad \equiv \quad 2 \xleftarrow{[\![\oplus]\!]} 2 \times 2 \; = \; \begin{array}{c|cccc} & 0 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}$$

Unifying notations

Yet another example, which will be relevant later on,

$$\begin{cases} f : 2^2 \times 2 \to 2 \\ f\ ((a,b),c) = ab + c - 2\ abc \end{cases} \equiv [\![f]\!] = \begin{array}{c|cccccccc} & ((0,0),0) & ((0,0),1) & ((0,1),0) & ((0,1),1) & ((1,0),0) & ((1,0),1) & ((1,1),0) & ((1,1),1) \\ \hline 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{array}$$

cf. the classical circuit:



(10)

Unifying notations

Most often we drop the parentheses $[\![\cdot]\!] \ldots$ and abbreviate $[\![f]\!]$ to f, as e.g. in the folowing rules:

$$y(f \cdot N)x = \langle \Sigma\ z\ :\ y = f\ z :\ z\ N\ x \rangle \quad (11)$$

$$y(g^{\circ} \cdot M \cdot f)x = (g\ y)\ M\ (f\ x) \quad (12)$$

Law (12) is very useful wrt. pointwise index manipulation.

Example of a **specification** using (12) twice:

$Sort : a^* \to a^*$
$Sort = (bag^{\circ} \cdot bag) \times (ordered^{\circ} \cdot true)$

List sorting: same bag of elements in both lists, but the output one is ordered.

Unifying notations

This **unified notation** for matrices, relations and functions includes quantifiers:

'Eindhoven notation' for all **quantifiers** *($\Theta := \Sigma, \forall, \exists$, etc):*

$$\langle \Theta \text{ vars } :\ \phi :\ \psi \rangle$$

Predicate ϕ determines the **range** of the quantification; ψ is a numeric **term**.

One-point rule:

$$\langle \Sigma\ k\ :\ k = e :\ \gamma \rangle \quad = \quad \gamma\,[k := e] \tag{13}$$

In the **trading rule**

$$\langle \Sigma\ x\ :\ \psi \wedge \phi :\ \gamma \rangle \ = \ \langle \Sigma\ x\ :\ \phi :\ \psi \times \gamma \rangle \tag{14}$$

ψ is both regarded as a range **predicate** or as a $\{0, 1\}$ term.

Categories of matrices — order-enrichments

Natural order:

$$M \leqslant N \quad \equiv \quad M - N \text{ is positive} \tag{15}$$

Support order:

$$M \preceq N \quad \equiv \quad \lfloor M \rfloor \leqslant \lfloor N \rfloor \quad \text{where } y \ \lfloor M \rfloor \ x = (y \ M \ x \neq 0) \tag{16}$$

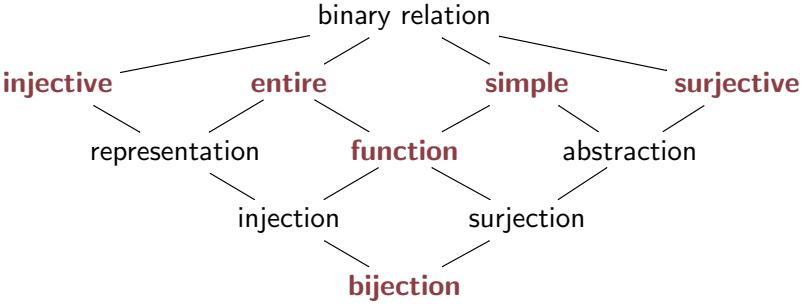
$\lfloor M \rfloor$ is called the **support** of M.

Both orders coincide for Boolean matrices, since $\lfloor M \rfloor = M$ for those matrices.

2-Matrix taxonomy

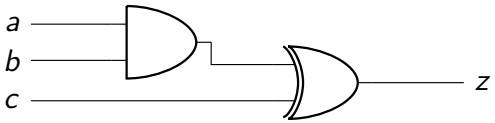
- **entire** — at least one 1 in every column
- **surjective** — at least one 1 in every row
- **simple** — at most one 1 in every column
- **injective** — at most one 1 in every row
- **bijective** — exactly one 1 in evey column and every row.

2-Matrix taxonomy



Reversibility

Back to



Can this be regarded as a **quantum circuit**?

Even simpler — can

$$\begin{cases} \oplus : 2 \times 2 \to 2 \\ x \oplus y = x + y - 2\,xy \end{cases} \quad \equiv \quad 2 \xleftarrow{[\![\oplus]\!]} 2 \times 2 \quad = \quad \begin{array}{c|cccc} & 0 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}$$

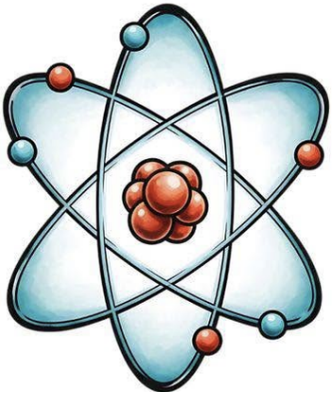
be regarded as a **quantum circuit**?

Reversibility

No! Both lose information (ie. are irreversible), cf.

> **Thermodynamics** — LANDAUER'S PRINCIPLE: *logically* **irreversible** *manipulation of information leads to an increase in entropy.*

Irreversibility conflicts with the principles of **quantum mechanics**.



Reversibility

A function f is **reversible** iff it is a bijection.

Then its **kernel** — $\ker f = f^\circ \cdot f$ — is the identity, where:

$$
\begin{array}{ll}
 & a' \ (\ker f)\ a = a' \ (f^\circ \cdot f)\ a \\
\equiv & \quad \{ \text{ composition; quantifier trading } \} \\
 & a' \ (\ker f)\ a = \langle \Sigma\ c\ :\ c = f\ a :\ c = f\ a' \rangle) \\
\equiv & \quad \{ \text{ one-point } \} \\
 & a' \ (\ker f)\ a = (f\ a = f\ a')
\end{array}
$$

Moreover, f° is also bijective.

Reversibility

Only **bijective** classical functional programs f are acceptable in the quantum paradigm:

$$f \text{ is bijective} \quad \equiv \quad \underbrace{f^\circ \cdot f}_{\ker f} = id \wedge \underbrace{f \cdot f^\circ}_{\ker f^\circ} = id \tag{17}$$

Example:

$$\begin{cases} \neg : 2 \to 2 \\ \neg\, x = 1 - x \end{cases} \quad \equiv \quad 2 \xleftarrow{[\![\neg]\!]} 2 \quad = \quad \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$$

This is trivially reversible since

$$x' \; (\ker \neg)\; x = (1 - x' = 1 - x) = (x' = x)$$

Reversibility

By contrast,

$$\left\{\begin{array}{l} \oplus : 2 \times 2 \to 2 \\ x \oplus y = x + y - 2\,xy \end{array}\right. \quad \equiv \quad 2 \xleftarrow{[\![\oplus]\!]} 2 \times 2 \quad = \quad \begin{array}{c|cccc} & 0 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}$$

is not reversible:

$$\ker(\oplus) = \ker \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} > \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Refining for reversibility

So, a REFINEMENT STEP is needed:

For $\oplus$ to become a quantum program one needs to make it **reversible.**

How? The following facts will help:

- $\ker f \geqslant id$ (reflexivity)
- $\ker f = \ker f^{\circ}$ (symmetry)
- the columns of $\ker f$ are either disjoint or the same (difuncionality).

Refining for reversibility

$$\ker(\oplus) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} > \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

We need to cancel the 1s outside *id*.

Another problem is that $\oplus : 2 \times 2 \to 2$ will never be reversible on cardinality grounds:

One needs type $2 \times 2 \to 2 \times 2$ *or isomorphic.*

Pairing

Given matrices $M : n \to m$ and $N : n \to k$, their **Khatri-Rao** product

$$M \triangledown N : n \to m \times k \tag{18}$$

is defined by

$$(y, z)\ (M \triangledown N)\ x \quad = \quad (y\ M\ x) \times (z\ N\ x) \tag{19}$$

For functions, this operation corresponds to **pairing** up their results:

$$(f \triangledown g)\ x = (f\ x, g\ x) \tag{20}$$

cf.

$$(y, z)\ (f \triangledown g)\ x \quad = \quad (y = f\ x) \times (z = g\ x)$$

etc.

Bra-ket notation

Every function of type $1 \to n$ is a **constant** function, denoted $\underline{k} : 1 \to n$, i.e.

$$\underline{k}\ 0 = k$$

Ket notation $|k\rangle$ is usually used for the column vector $[\![\underline{k}]\!]$.

Bra notation $\langle k|$ is used for the row vector $[\![\underline{k}]\!]^\circ$.

Since $\underline{x} \triangledown \underline{y} = \underline{(x, y)}$, one has $|x\rangle \triangledown |y\rangle = |(x, y)\rangle$, etc.

It is usual to write $|xy\rangle$ instead of $|(x, y)\rangle$, etc, etc.

Example aside.

$|10\rangle =$

	0
$(0,0)$	0
$(0,1)$	0
$(1,0)$	1
$(1,1)$	0

Refining for reversibility

For example, $(f \triangledown \oplus) : 2 \times 2 \to 2 \times 2$ for any $f : 2 \times 2 \to 2$.

Important:

Pairing *can only increase* **injectivity**:

$$\ker\,(f \triangledown g) = (\ker f) \times (\ker g) \tag{21}$$

Goal:

Find $f : 2 \times 2 \to 2$ *such that* $(\ker f) \times (\ker \oplus) = id$

Refining for reversibility

We start from some "kernel" ensuring 0s where $\ker \oplus$ has spurious 1s, and cancel zeros symmetrically, outside the diagonal:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \ker \pi_1$$

What we obtain is $\ker \pi_1$ where $\pi_1 \ (a, b) = a$:

$$\mathbb{B} \xleftarrow{\pi_1} \mathbb{B} \times \mathbb{B} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

(The first projection of a pair.)

Refining for reversibility

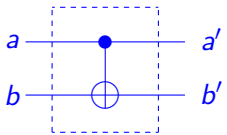
Then:

$$
\begin{array}{ll}
 & \pi_1 \triangledown \oplus \\
= & \quad \{ \text{ as matrices } \} \\
 & \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \triangledown \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix} \\
= & \quad \{ \text{ Khatri-Rao (19) } \} \\
 & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}
\end{array}
$$

Refining for reversibility

The outcome of our calculation is a well-known quantum gate,

$$\pi_1 \triangledown \oplus = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} =$$



called

$$CNOT : 2 \times 2 \to 2 \times 2$$

Summary: *CNOT* is a **refinement** of $(\oplus)$ towards reversibility.[1]

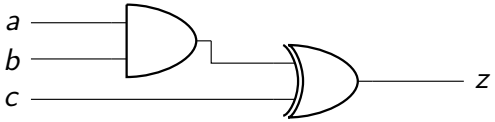
[1] **NB**: Other such refinements exist, eg. $\pi_2 \triangledown \oplus$.

Refining for reversibility

Functions f anf g such that $\ker\,(f \triangledown g) = id$ are said to be **complementary**.

π_1-**complementation** *plays a significant role in the* **design** *of quantum gates.*

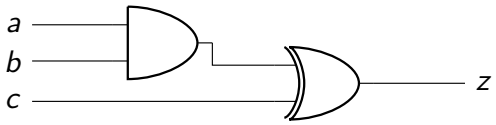
For instance, back to:



What do we get from it via π_1-**complementation**?

Design by π_1-complementation

$f\ ((a, b), c) =$



Let us regard f as a **specification**, which is a quite expressive one:

Instance	$ab + c - 2\ abc$	Gate
$b = 1$	$a + c - 2\ ac$	$a \oplus c$
$c = 0$	ab	$a \wedge b$
$c = 1$	$1 - ab$	$\neg\ (a \wedge b)$
$a = 0$	c	c
$ab = 1$	$1 - c$	$\neg\ c$

and son on.

1st complementation

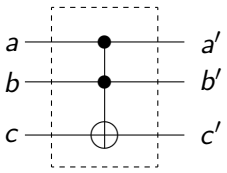
Via the same refinement process, one will obtain:

$$[\![\pi_1 \triangledown f]\!] =$$

	$((0,0),0)$	$((0,0),1)$	$((0,1),0)$	$((0,1),1)$	$((1,0),0)$	$((1,0),1)$	$((1,1),0)$	$((1,1),1)$
$((0,0),0)$	1	0	0	0	0	0	0	0
$((0,0),1)$	0	1	0	0	0	0	0	0
$((0,1),0)$	0	0	1	0	0	0	0	0
$((0,1),1)$	0	0	0	1	0	0	0	0
$((1,0),0)$	0	0	0	0	1	0	0	0
$((1,0),1)$	0	0	0	0	0	1	0	0
$((1,1),0)$	0	0	0	0	0	0	0	1
$((1,1),1)$	0	0	0	0	0	0	1	0

Toffoli (CCNOT)

The gate we obtain this time is the **Toffoli** gate,

$[\![\pi_1 \triangledown f]\!] =$ 

where $f\ ((a, b), c) = ab + c - 2\ abc$

Functionally:

$$ccnot : 2^2 \times 2 \to 2^2 \times 2$$
$$ccnot\ ((1, 1), c) = ((1, 1), 1 - c)$$
$$ccnot\ ((a, b), c) = ((a, b), c)$$

A dual view

Another way of looking at these gates is to recall the **direct sum** of two matrices,

$$M \oplus N = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \qquad \text{of type} \qquad \begin{array}{ccc} m & & n \\ {\scriptstyle M}\downarrow & & {\scriptstyle N}\downarrow \\ p & & q \end{array} \qquad \begin{array}{c} m+n \\ \downarrow{\scriptstyle M \oplus N} \\ p+q \end{array}$$

Clearly,

$$2 \times 2 \xleftarrow{cnot} 2 \times 2 \;=\; \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} = (\, 2 \xleftarrow{id} 2 \,) \oplus (\, 2 \xleftarrow{\neg} 2 \,)$$

NB: Don't forget that $2 \times n = n + n$. 

A dual view

So we can define, in general, the **control-gate** operator:

$$C : (n \to n) \to (n + n \to n + n)$$
$$C\ (X) = id \oplus X$$

As we have just seen:

$$cnot = C\ (\neg)$$

What do we get if we **double** the controlling effect? The Toffoli gate:

$$C\ (C\ (\neg)) = id \oplus (id \oplus \neg) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

A dual view

NB: **direct sum** is defined algebraically by

$$M \oplus N = [i_1 \cdot M \ , i_2 \cdot N]$$

where:

$$[M \ , N] = M \cdot i_1^\circ + N \cdot i_2^\circ$$

Both definitions are based on two **injections**

$$m \xrightarrow{i_1} m + n \xleftarrow{i_2} n$$

which are **orthogonal**,

$$i_1 \cdot i_2^\circ = 0$$

and jointly **surjective**:

$$i_1 \cdot i_1^\circ + i_2 \cdot i_2^\circ = id$$

The whole "dual view" arises from the **universal** property:

$$X = [M \ , N] \quad \equiv \quad \begin{cases} X \cdot i_1 = M \\ X \cdot i_2 = N \end{cases} \tag{22}$$

Meaning of $\pi_1 \triangledown f$ injectiveness

Back to π_1-**complementation**:

- What does it actually mean?

Expanding $\ker(\pi_1 \triangledown f) = id$ we get:

$$f(a,b) = f(a,b') \Rightarrow b = b'$$

In other words, f is **injective** on the **second** argument once the **first** is fixed.

Example:

$$\begin{array}{ll} & a \oplus b = a \oplus b' \\ \equiv & \quad \{\ a \oplus b = a + b - 2\,ab;\ \text{arithmetics}\ \} \\ & b = b' + 2\,a\,(b - b') \\ \Rightarrow & \quad \{\ \text{substitutions}\ \} \\ & \left\{ \begin{array}{l} a = 0 \Rightarrow b = b' \\ a = 1 \Rightarrow b = 2\,b - b' \ \equiv\ b = b' \end{array} \right. \end{array}$$

Generalizing 1st-complementation

Observing the pattern in

$$ccnot = \pi_1 \triangledown (\oplus \cdot (\wedge \otimes id))$$
$$cnot = \pi_1 \triangledown (\oplus \cdot (id \otimes id))$$

one may think of the **generalization** aside.

NB: the generalization also includes moving to arbibtrary types A, B which generalize the numeric ones (m, n, etc) we started from.

Given arbitrary (possibly not reversible) $f : A \to B$.

Is there a **reversible refinement** of f?

Yes — provided B is a monoid $(B; \theta, 0)$ such that $x\ \theta\ x = 0$.

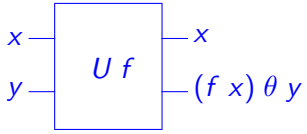
Then the 1st-complement of $\theta \cdot (f \otimes id)$,

$$\mathbf{U}\, f = \pi_1 \triangledown (\theta \cdot (f \otimes id))$$

is reversible.

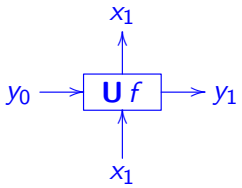
Generalizing 1st-complementation

Checking — for every f, $\mathbf{U}\,f$ is a *bijection* because it is its own inverse:



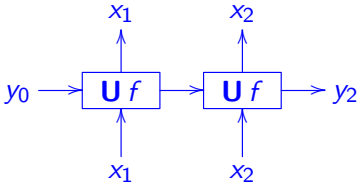
$$
\begin{aligned}
& \mathbf{U}\,f \cdot \mathbf{U}\,f = id \\
\equiv & \quad \{\ \mathbf{U}\,f\ (x,y) = (x,(f\ x)\ \theta\ y)\ \} \\
& \mathbf{U}\,f\ (x,(f\ x)\ \theta\ y) = (x,y) \\
\equiv & \quad \{\ \text{again } \mathbf{U}\,f\ (x,y) = (x,(f\ x)\ \theta\ y)\ \} \\
& (x,(f\ x)\ \theta\ ((f\ x)\ \theta\ y)) = (x,y) \\
\equiv & \quad \{\ \theta \text{ is associative and } x\ \theta\ x = 0\ \} \\
& (x,0\ \theta\ y) = (x,y) \\
\equiv & \quad \{\ 0\ \theta\ x = x\ \} \\
& (x,y) = (x,y)
\end{aligned}
$$

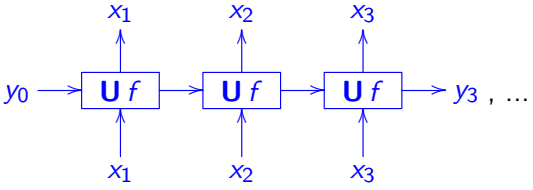
Chaining 1 *st*-complemented computations

Drawing 

instead of 

one may think of **chaining** such computations,

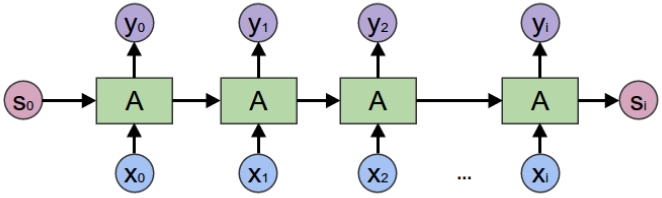


, then 

, ...

Similar construction in neural networks

By the way:



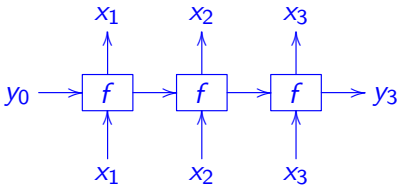
(RNN = accumulating maps[2])

[2]Source: **Neural Networks, Types, and Functional Programming** by C. Olah, 2015.

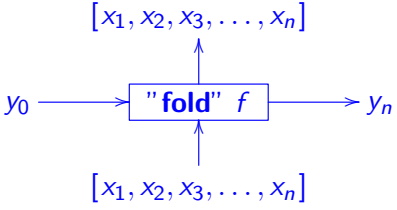
Part II

Towards reversible recursion

The chain...



suggests something like



, ...

Recursion

'Folding' is a well-known design pattern in **functional programming**.

For instance (in Haskell):

$$\Sigma = \mathit{foldr}\ (+)\ 0$$

where

$$\begin{aligned}&\mathit{foldr} :: (a \to b \to b) \to b \to [a] \to b\\&\mathit{foldr}\ f\ e\ [\,] = e\\&\mathit{foldr}\ f\ e\ (a : x) = f\ a\ (\mathit{foldr}\ f\ e\ x)\end{aligned}$$

Clearly: $(+)$ not reversible, Σ not reversible...

Recursion

Another version of *foldr*:

$$
\begin{aligned}
&\mathit{foldr}' :: (a \times b \to b) \to ([a] \times b) \to b \\
&\mathit{foldr}'\ g\ ([\,], e) = e \\
&\mathit{foldr}'\ g\ ((a : x), e) = g\ (a, \mathit{foldr}'\ g\ (x, e))
\end{aligned}
$$

Clearly: $\mathit{foldr}'\ g :: ([a] \times b) \to b$ not reversible on type grounds.

Question: for which h is the 1st-complementation of $\mathit{foldr}'\ h$ reversible?

$$
\begin{aligned}
&\mathit{foldr}'' :: (a \times b \to b) \to ([a] \times b) \to ([a] \times b) \\
&\mathit{foldr}''\ h = \pi_1 \triangledown (\mathit{foldr}'\ h)
\end{aligned}
$$

Towards reversible folds

We will need something more general:

$$
\begin{aligned}
&\mathit{foldr}' :: (b \to c) \to (a \times b \to c) \to ([a] \times b) \to c \\
&\mathit{foldr}'\ h_1\ h_2\ ([\,], e) = h_1\ e \\
&\mathit{foldr}'\ h_1\ h_2\ ((a : x), e) = h_2\ (a, \mathit{foldr}'\ h_1\ h_2\ (x, e))
\end{aligned}
$$

This corresponds to the diagram below,

$$
\begin{array}{ccc}
A^* \times B & \xleftarrow{\alpha} & B + A \times (A^* \times B) \\
{\scriptstyle (\!|h|\!)}\downarrow & & \downarrow{\scriptstyle id \oplus id \otimes (\!|h|\!)} \\
C & \xleftarrow[h]{} & B + A \times C
\end{array}
$$

where:

$$
\begin{aligned}
&h = [h_1\ , h_2] \\
&(\!|h|\!) = \mathit{foldr}'\ h_1\ h_2 \\
&\alpha = [\alpha_1\ , \alpha_2] \\
&\alpha_1\ b = ([\,], b) \\
&\alpha_2\ (a, (x, b)) = (a : x, b)
\end{aligned}
$$

Towards reversible folds

In pointfree notation,

$$A^* \times B \xleftarrow{\alpha} B + A \times (A^* \times B)$$

is the isomorphism

$$\alpha = [\mathsf{nil} \triangledown id \ , (\mathsf{cons} \otimes id) \cdot \mathbf{a}] \tag{23}$$

where

$$\begin{aligned} \mathsf{nil}\ _ &= [\,] \\ \mathsf{cons}\ (a, x) &= a : x \end{aligned} \tag{24}$$

and

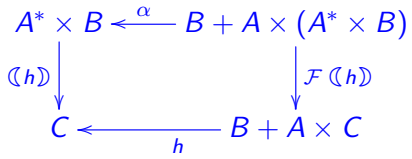
$$(A \times B) \times C \xleftarrow{\mathbf{a}} A \times (B \times C) \ = (id \otimes \pi_1) \triangledown (\pi_2 \cdot \pi_2) \tag{25}$$

Towards reversible folds

It can be shown that $(\!|h|\!)$ satisfies the following **universal** property

$$k = (\!|h|\!) \quad \equiv \quad k \cdot \alpha = h \cdot \mathcal{F}\ k \quad (26)$$

where $\mathcal{F}\ f = id \oplus id \otimes f$.



From (26,25) one can infer

$$A^* \xleftarrow{\pi_1} A^* \times B = (\!|\mathbf{in}|\!) \quad (27)$$

where $\mathbf{in} = [\mathsf{nil}\ , \mathsf{cons}]$ in the initial algebra of lists.[3] Moreover, a "banana-split" law holds:

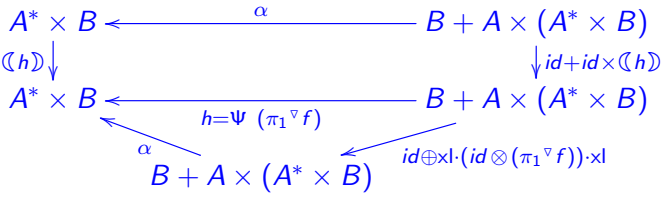
$$(\!|f|\!) \triangledown (\!|g|\!) = (\!|(f \times g) \cdot (\mathcal{F}\ \pi_1 \triangledown \mathcal{F}\ \pi_2)|\!), \quad (28)$$

[3] Details in

Promoting 1st-complementation

Suppose **non-injective** $f : A \times B \to B$ is complemented by $\pi_1 : A \times B \to A$.

The following diagram shows how to use $\pi_1 \triangledown f$ to build an envelope for $\mathit{foldr}\ \overline{f}\ b$:[4]



where

$$\Psi\ x = \alpha \cdot (id \oplus \mathsf{xl} \cdot (id \otimes x) \cdot \mathsf{xl})$$

resorting to isomorphism $A \times (B \times C) \xrightarrow{\mathsf{xl}} B \times (A \times C)$.

[4] NB: $\overline{f}$ is such that $\overline{f}\ a\ b = f\ (a, b)$. (Currying.)

Promoting complementation

It can be shown that:[5]

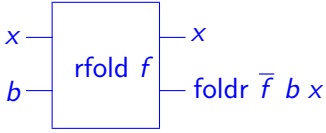
- Ψ preserves injectivity, as does $(\!|_|\!)$;
- $\pi_1 \triangledown f$ being **injective** ensures $\pi_1 \triangledown (\!|[id\ , f]|\!)$ is also **injective**.

Thus:

The π_1-complementation of f in $foldr\ \overline{f}\ b$ is promoted to the π_1-complementation of the fold itself.

So π_1-complementation is *propagated* inductively and we get the notion of a *reversible fold*, defined by

$$\textbf{rfold}\ f = (\!|\Psi\ (\pi_1 \triangledown f)|\!).$$



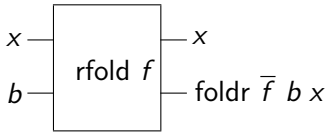
[5] Details in

Promoting complementation

Unfolding the definition we get, in standard Haskell notation:

$$\begin{aligned}
&\textbf{rfold} \;::((a,b)\to b)\to([a],b)\to([a],b)\\
&\textbf{rfold}\ f\ ([\,],b)=([\,],b)\\
&\textbf{rfold}\ f\ (a:x,b)=(a:y,f\ (a,b'))\ \textbf{where}\ (y,b')=\textbf{rfold}\ f\ (x,b)
\end{aligned}$$

We can therefore rely on the reversibility of **rfold** f



provided f is complemented by π_1.

Generalization

Given some $f : A \times B \to C \times B$, let us define $\langle\!|f|\!\rangle = (\!|\Psi\ f|\!)$ as pictured in the diagram aside:

$$\begin{array}{ccc} A^* \times B & \xleftarrow{\alpha} & B + A \times (A^* \times B) \\ {\scriptstyle \langle\!|f|\!\rangle}\downarrow & & \downarrow{\scriptstyle id + id \times \langle\!|f|\!\rangle} \\ C^* \times B & \xleftarrow[\Psi\ f]{} & B + A \times (C^* \times B) \end{array}$$

Converting to pointwise notation,

$$\begin{array}{l} \langle\!|\cdot|\!\rangle :: ((a, b) \to (c, b)) \to ([a], b) \to ([c], b) \\ \langle\!|f|\!\rangle\ ([\,], b) = ([\,], b) \\ \langle\!|f|\!\rangle\ (a : x, b) = (c : y, b'') \textbf{ where} \\ \quad (y, b') = \langle\!|f|\!\rangle\ (x, b) \\ \quad (c, b'') = f\ (a, b') \end{array}$$

we realize it corresponds to the standard `mapAccumR` function in Haskell 

Generalization

All in all, we have shown that, if f is reversible, then recursive $\langle\!|f|\!\rangle$ will also be reversible.

In case f is not reversible, $\langle\!|(\pi_1 \triangledown f)|\!\rangle$ will be so provided f is injective on the second argument, freezing the first.

Such is the case in $\langle\!|cnot|\!\rangle$, $\langle\!|ccnot|\!\rangle$ etc.

Let us now extend this result to the non-classical case.

Going into quantum

The matrix view of **functions** and **relations**, e.g. **controlled-not**...

$$\begin{cases} \mathit{cnot}\ (0,b) = (0,b) \\ \mathit{cnot}\ (1,b) = (1, 1-tb) \end{cases} =$$

	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,0)$	1	0	0	0
$(0,1)$	0	1	0	0
$(1,0)$	0	0	0	1
$(1,1)$	0	0	1	0

... can be extended **probabilistically**, as the following "evolution" of *cnot* shows:

	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,0)$	1	0	0	0
$(0,1)$	0	$\frac{1}{2}$	0	0
$(1,0)$	0	$\frac{1}{2}$	0	1
$(1,1)$	0	0	1	0

Going into quantum

Moving further, it can even become doubly **stochastic**, for instance:

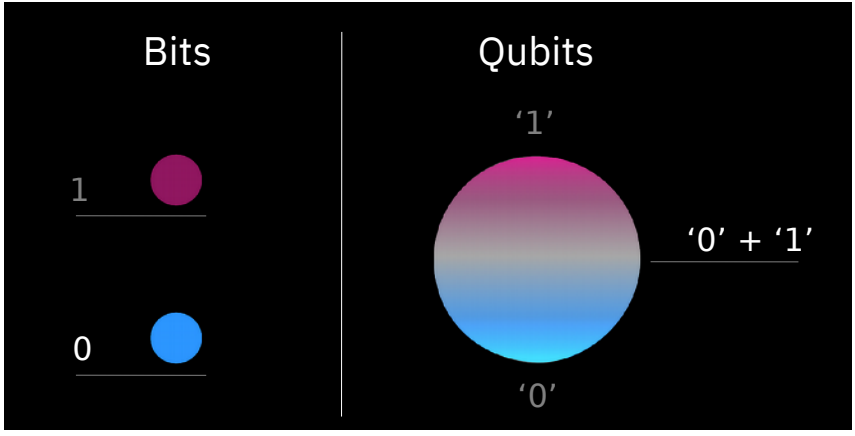
	$(0,0)$	$(0,1)$	$(1,0)$	$(1,1)$
$(0,0)$	$\frac{1}{2}$	0	$\frac{1}{2}$	0
$(0,1)$	0	$\frac{1}{2}$	0	$\frac{1}{2}$
$(1,0)$	0	$\frac{1}{2}$	0	$\frac{1}{2}$
$(1,1)$	$\frac{1}{2}$	0	$\frac{1}{2}$	0

Taking square roots, we may even think of...

$$bell = \begin{array}{c|cccc} & (0,0) & (0,1) & (1,0) & (1,1) \\ \hline (0,0) & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} & 0 \\ (0,1) & 0 & \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ (1,0) & 0 & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} \\ (1,1) & \frac{1}{\sqrt{2}} & 0 & -\frac{1}{\sqrt{2}} & 0 \end{array}$$

... reaching **unitary** evolutions, which is where **quantum** lives.

Quantum information



Credits: IBM Research AI & Q

From bijections to unitaries

Matrices in quantum computing represent **unitary transformations**:

A $\mathbb{C}$-valued matrix M is unitary iff

$$M \cdot M^\dagger = M^\dagger \cdot M = id \tag{29}$$

where $M^\dagger$ is the **conjugate** *transpose of M.*

Compare (29) with:

$$f \cdot f^\circ = f^\circ \cdot f = id$$

Conclude that **isomorphisms** are exactly the **classical** unitrary transformations.

Thus the theory developed above extends smoothly to quantamorphisms with **unitary** parameters.

From bijections to unitaries

This extension can be observed in many situations, eg.

- Unitary M preserves **inner products** between vectors, compared to
$$\langle w|\ M^{\dagger}\ M\ |w'\rangle = \langle w|w'\rangle$$
- Function m is **injective**,
$$m\,(w) = m\,(w') \ \equiv\ w = w'$$
that is:
$$\underline{w}^{\circ} \cdot m^{\circ} \cdot m \cdot \underline{w}' = \underline{w}^{\circ} \cdot \underline{w}'$$

(Recall that $|a\rangle$ is the vector representation of constant function $\underline{a}$ etc.)

In what comes next we shall see this "interplay" at work, but in a different way.

Monadic representation

We have so far regarded **functions** as special cases of **relations**, which in turn are Boolean **matrices**.

There is a "symmetric view":

Any (e.g. complex-valued) matrix $Q : m \to n$ can be represented by a function $q : m \times n \to \mathbb{C}$ where $q\,(i,j)$ is the cell-value $i\ Q\ j$.

By *currying* q we obtain $\overline{q} : m \to \mathbb{C}^n$. Now:

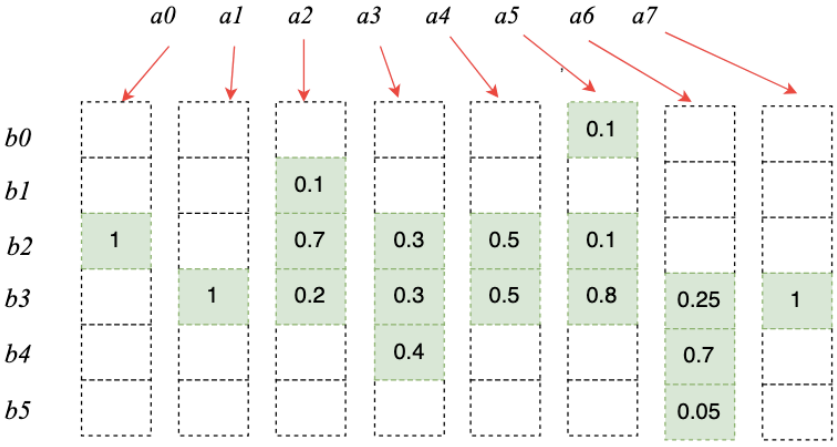
Type $\mathbb{C}^n$ is nothing but that of n-dimensional ($\mathbb{C}$-valued) **vectors**.

Thus $\overline{q}$ maps column indices to a **vector space**.

Example

	a0	*a1*	*a2*	*a3*	*a4*	*a5*	*a6*	*a7*
b0						0.1		
b1			0.1					
b2	1		0.7	0.3	0.5	0.1		
b3		1	0.2	0.3	0.5	0.8	0.25	1
b4				0.4			0.7	
b5							0.05	

Example


a0
a1
a2
a3
a4
a5
a6
a7
b0
b1
b2
b3
b4
b5
1
1
0.1
0.7
0.2
0.3
0.3
0.4
0.5
0.5
0.1
0.8
0.1
0.25
0.7
0.05
1

Monadic representation

For instance, the *Hadamard* gate

$$H = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{array} \tag{30}$$

is represented by the function

$$\begin{aligned} &had :: 2 \to Vec\ 2 \\ &had\ 0 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \\ &had\ 1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} \end{bmatrix} \end{aligned}$$

Monadic representation

Generalizing to arbitrary types, $q : A \times B \to \mathbb{C}$ leads to:

$$\overline{q} : A \to \underbrace{\mathbb{C}^B}_{Vec\ B} \tag{31}$$

It turns out that type $Vec\ B = \mathbb{C}^B$ forms a **monad**.

Thus matrices can be represented by **monadic functions**, with monad expressivity helping in easily building quantum programs and running (simulating) them.

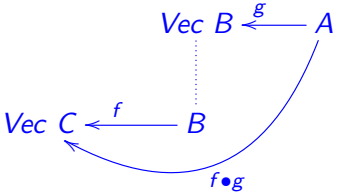
Monads in FP are used to **elegantly** *describe computations with* **effects**.

Here they will be used to elegantly "encapsulate" **quantum effects.**

Monadic representation

All matrix-categorial operators have a corresponding *Vec*-monadic counterpart.

For example, matrix multiplication — that is, **composition** — is represented as follows:



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

Simulating quantamorphisms

Another example: given $f : A \to Vec\ X$ and $g : B \to Vec\ Y$, the tensor product is represented by:

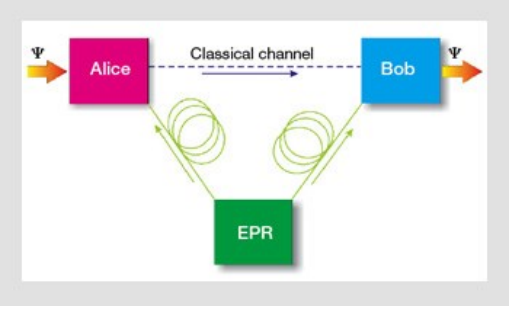
$$\begin{aligned}(f \otimes g)\ (a, b) = \mathbf{do}\ \{ \\ x \leftarrow f\ a; \\ y \leftarrow g\ b; \\ return\ (x, y)\ \}\end{aligned} \tag{32}$$

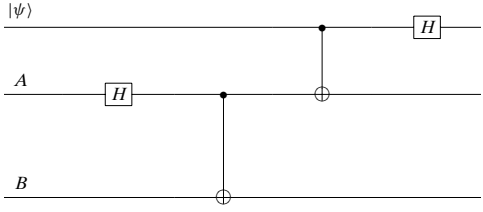
where $return\ x = |x\rangle$. Cf:

$$\begin{array}{ccccc} A & \cdots\cdots & C & \cdots\cdots & A \times C \\ {\scriptstyle f}\downarrow & & {\scriptstyle g}\downarrow & & \downarrow{\scriptstyle f \otimes g} \\ Vec\ D & \cdots\cdots & Vec\ E & \cdots\cdots & Vec\ (D \times E) \end{array}$$

Well-known example

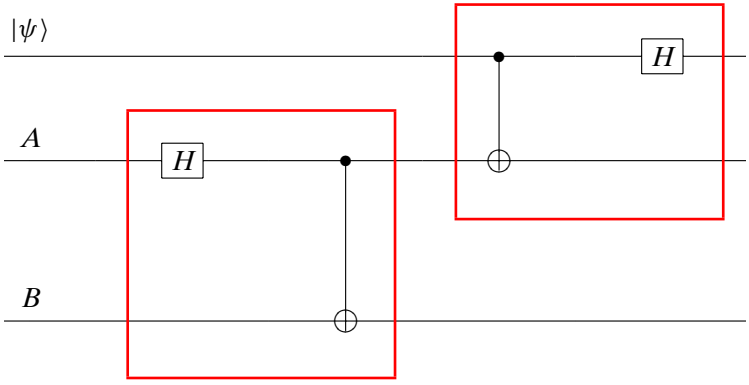
Bird's-eye view of the **structure** of a famous example (the "Alice" part of the teleportation protocol):



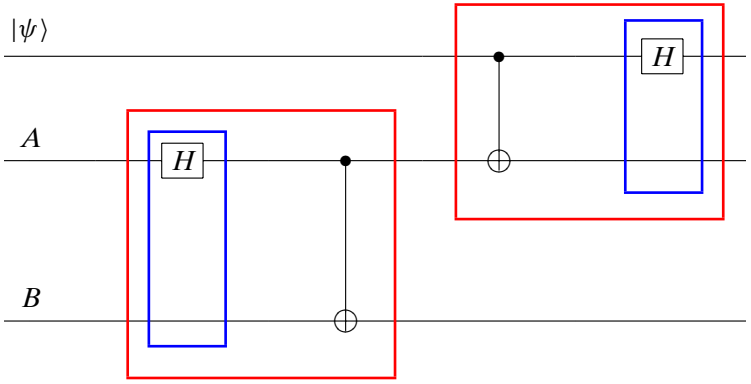


(Cf. **entangled** photon pairs)

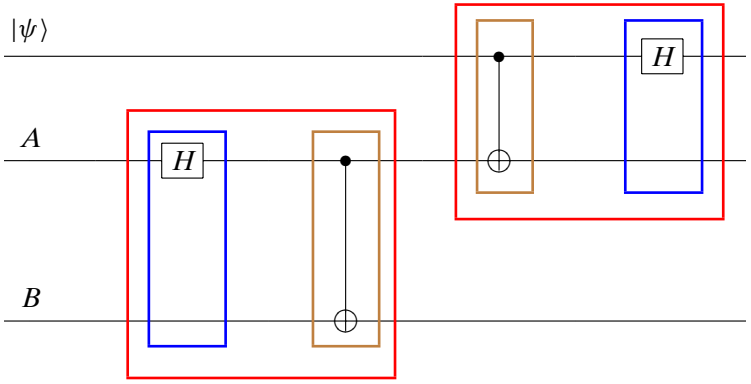
Well-known example



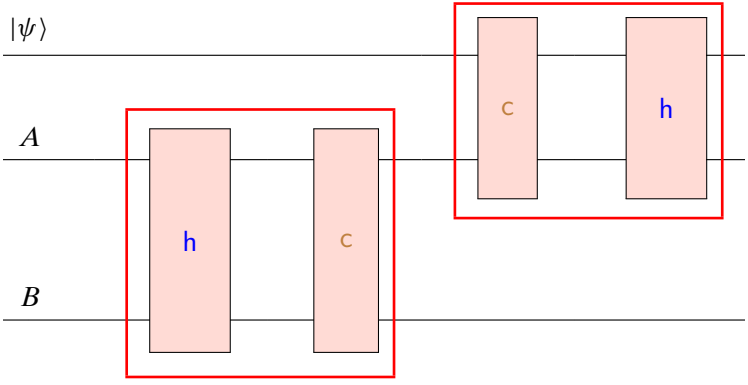
Well-known example



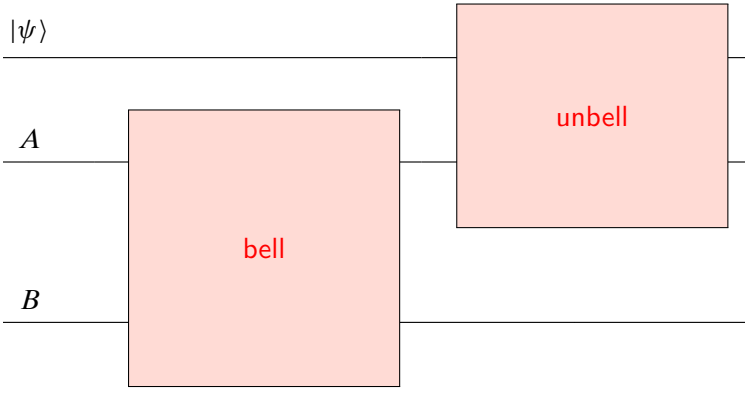
Well-known example



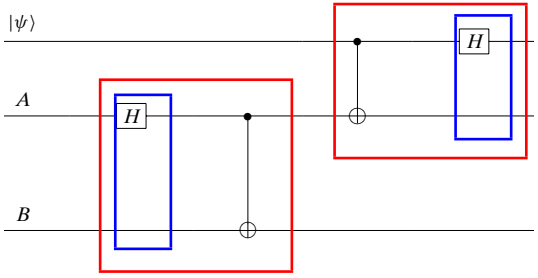
Well-known example



Well-known example



Monadic representation

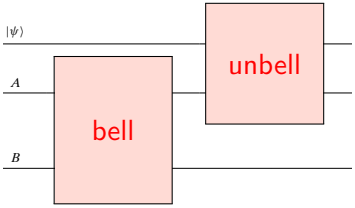


$$
\begin{aligned}
&\mathit{bell}\ (a, b) = \mathbf{do}\ \{ \\
&\quad x \leftarrow \mathit{had}\ a; \\
&\quad \mathit{return}\ (\mathit{cnot}\ (x, b)) \\
&\quad \}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{unbell}\ (c, a) = \\
&\quad \mathbf{let}\ (_, a') = \mathit{cnot}\ (c, a) \\
&\quad \mathbf{in\ do}\ \{ \\
&\qquad b \leftarrow h\ c; \\
&\qquad \mathit{return}\ (b, a')\ \}
\end{aligned}
$$

Monadic representation

$$
\begin{aligned}
&\mathit{alice}\ (c, (a, b)) = \\
&\quad \textbf{do}\ \{ \\
&\qquad (a', b') \leftarrow \mathit{bell}\ (a, b); \\
&\qquad (c', a'') \leftarrow \mathit{unbell}\ (c, a'); \\
&\qquad \mathit{return}\ (c', (a'', b')) \\
&\qquad \}
\end{aligned}
$$



Observation:

QP = *monadic functional programming, after all?*

(Monadic) quantamorphisms

Back to the main topic, we can encode **quantamorphisms** as monadic programs:

$$
\begin{aligned}
&(\!|\cdot|\!) :: ((a, b) \to \mathit{Vec}\ (c, b)) \to ([a], b) \to \mathit{Vec}\ ([c], b)\\
&(\!|f|\!)\ ([\,], b) = \mathit{return}\ ([\,], b)\\
&(\!|f|\!)\ (h : t, b) = \mathbf{do}\ \{\\
&\quad (t', b') \leftarrow (\!|f|\!)\ (t, b);\\
&\quad (h'', b'') \leftarrow f\ (h, b');\\
&\quad \mathit{return}\ (h'' : t', b'')\\
&\quad \}
\end{aligned}
$$

Qubit b is controlled according to a **list** of controls triggering the **quantum** operator f (unitary) along the computation. The outcome is unitary.

Quantamorphism simulation

Classic control example (only showing the non-zero amplitudes):

$$x = \langle\!|\, bell \,|\!\rangle\ ([1,1,1],0) =$$

$2^* \times 2$	
$([0,0,0],1)$	$\frac{1}{2\sqrt{2}}$
$([1,0,0],0)$	$-\frac{1}{2\sqrt{2}}$
$([0,1,0],0)$	$\frac{1}{2\sqrt{2}}$
$([1,1,0],1)$	$-\frac{1}{2\sqrt{2}}$
$([0,0,1],0)$	$\frac{1}{2\sqrt{2}}$
$([1,0,1],1)$	$-\frac{1}{2\sqrt{2}}$
$([0,1,1],1)$	$\frac{1}{2\sqrt{2}}$
$([1,1,1],0)$	$-\frac{1}{2\sqrt{2}}$

Quantamorphism simulation

Quantum control example:

$$y = \textbf{do}\ \{\, i \leftarrow x;\ \langle\!|\, bell \,|\!\rangle\ i \,\}$$

This superposes the quantamorphism with itself by passing **quantum** information x to the **control part** itself:

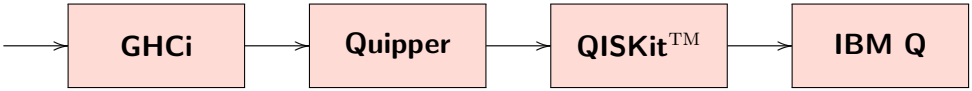
Outcome:

$$y = \begin{array}{c|c} 2^* \times 2 & \\ \hline ([1,0,0],0) & \frac{1}{2} \\ ([1,0,0],1) & \frac{1}{2} \\ ([0,1,1],0) & -\frac{1}{2} \\ ([0,1,1],1) & \frac{1}{2} \end{array}$$

(Only showing the non-zero amplitudes.)

Quantamorphism compilation

Tool-chain:



- **GHCi** — depending on the resources (number of qubits available), we select a range of values of the input that can be *represented* in such resources, generate the corresponding **unitary matrix**
- **Quipper** [2] — generates the **quantum circuit** from such a matrix
- **QISKit** — Python interface to the hardware, adding error-correction extra circuitry
- **IBM-Q** — the actual hardware where **QISKit** runs its code.

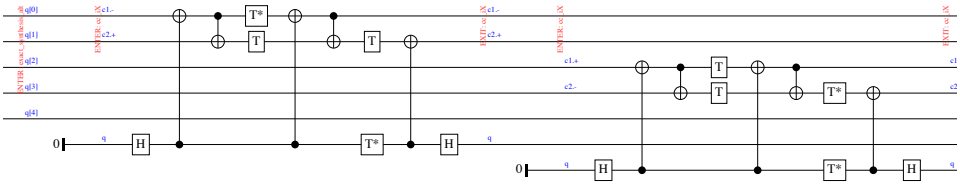
Compiling for 5 qubits

Matrix sent to Quipper for $(\!|bell|\!)$:

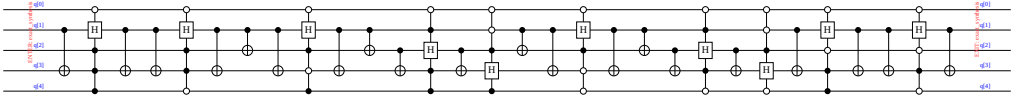
	$([],0)$	$([],1)$	$([0],0)$	$([0],1)$	$([0,0],0)$	$([0,0],1)$	$([1,0],0)$	$([1,0],1)$	$([1],0)$	$([1],1)$	$([0,1],0)$	$([0,1],1)$	$([1,1],0)$	$([1,1],1)$
$([],0)$	1	0	0	0	0	0	0	0	0	0	0	0	0	0
$([],1)$	0	1	0	0	0	0	0	0	0	0	0	0	0	0
$([0],0)$	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	0
$([0],1)$	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0
$([0,0],0)$	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0
$([0,0],1)$	0	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$
$([1,0],0)$	0	0	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$
$([1,0],1)$	0	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0
$([1],0)$	0	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	$-\frac{1}{\sqrt{2}}$	0	0	0	0
$([1],1)$	0	0	$\frac{1}{\sqrt{2}}$	0	0	0	0	0	$-\frac{1}{\sqrt{2}}$	0	0	0	0	0
$([0,1],0)$	0	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$
$([0,1],1)$	0	0	0	0	$\frac{1}{2}$	0	$\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	$-\frac{1}{2}$	0
$([1,1],0)$	0	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$	0
$([1,1],1)$	0	0	0	0	0	$\frac{1}{2}$	0	$-\frac{1}{2}$	0	0	0	$-\frac{1}{2}$	0	$\frac{1}{2}$

Quantum circuit

First part of quantum circuit generated from the given program:

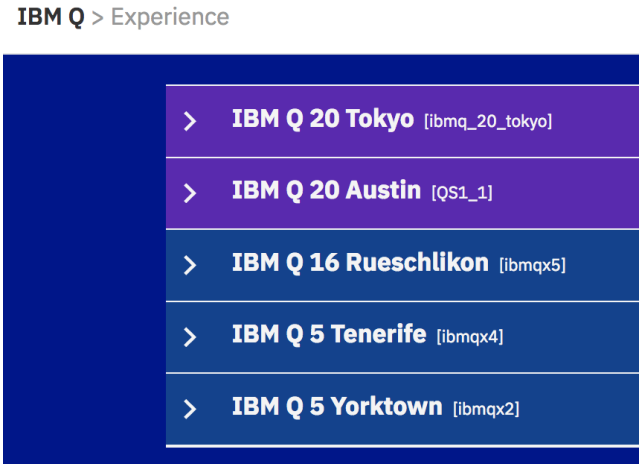


Simpler example — $(\!|had \otimes id|\!)$ compiles to a much simpler circuit:



(With thanks to: Ana Neri, Afonso Rodrigues, Rui S. Barbosa)

IBM Q-experience devices



Running the circuits on IBM-Q

Each job performs 1000 runs for the given input provided and returns the outcome of the measurements, see aside.

High percentage of errors, still.

2 em máquina real:

```
In [34]: backend = 'ibmqx2'   # Backend where you execute your progr
         circuits = ['Circuit']   # Group of circuits to execute
         shots = 1024           # Number of shots to run the program
         max_credits = 3          # Maximum number of credits to spe
         qp.set_api(Qconfig.APItoken, Qconfig.config['url']) # set t

         result_real = qp.execute(circuits, backend, shots=shots, ma

In [35]: result_real.get_counts('Circuit')

Out[35]: {'00000': 79,
          '00001': 47,
          '00010': 285,
          '00011': 92,
          '00100': 80,
          '00101': 42,
          '00110': 328,
          '00111': 71}

"00110"  32.0%
"00010"  27.8%
"00011"   9.0%
"00100"   7.8%
"00000"   7.7%
"00111"   6.9%
"00001"   4.6%
"00101"   4.1%
```

Wrapping up

Quantamorphisms — recursive quantum programming strategies dispensing with **measurements**.

Simulation and compilation thanks to **monadic** functional programming (Haskell).

Towards **correct-by-construction** recursvive reversible/quantum programs.



Source: IBM Q Experience website

Follow up work

Applications?

Can algorithms such as Grover's be expressed using quantamorphisms?

Fused compilation of quantamorphisms (simpler tool-chain)

Interested in the **dual view** using direct sums (coproducts) instead of tensors (products) — a topic of recent interest in the comunity, cf.

'Quantum Circuits Are Just a Phase', by Chris Heunen et al, POPL 2026. [3]

Cf. quantum controlled **if** ... **then** ... **else** ... in .

B. Coecke and A. Kissinger.
Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning.
Cambridge University Press, 2017.

A.S. Green, P.L. Lumsdaine, N.J. Ross, P. Selinger, and B. Valiron.
An introduction to quantum programming in Quipper.
CoRR, cs.PL(arXiv:1304.5485v1), 2013.

Chris Heunen, Louis Lemonnier, Christopher McNally, and Alex Rice.
Quantum circuits are just a phase.
Proc. ACM Program. Lang., 10(POPL):2586–2613, 2026.

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

A. Neri, R.S. Barbosa, and J.N. Oliveira.
Compiling quantamorphisms for the IBM Q experience.

IEEE Trans. Software Eng., 48(11):4339–4356, 2022.