First steps in a (linear) algebra of (quantum) functional programming

IFIP WG2.1 meeting #75

Montevideo, Uruguay, February 2017

J.N. Oliveira

HASLab
HIGH-ASSURANCE SOFTWARE LABORATORY
INESCTEC

INESC TEC & University of Minho
(QuantaLab initiative)

Context

QUANTA ⟨L|A|B⟩

- New initiative at Minho
- **Quantum computing** more and more on spot in the media
- Trying to address **quantum programming** from an **algebraic** (category-theoretical) perspective.
- Need to abstract from the physics level.

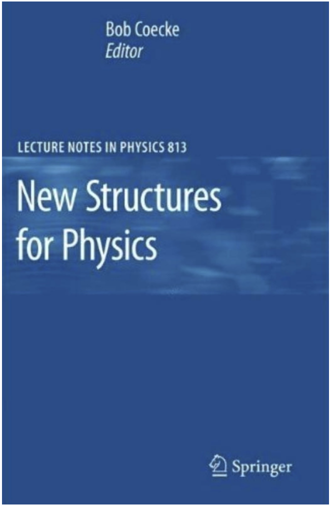
Literature is vast!

2011 (vintage year):

The “bible”, compiled by B. Coecke:

> **“New Structures for Physics”**, **Lect. Notes in Physics** *volume 813, 2011*

The generic structures (**monoidal categories**) in which **quantum physics** are expressed in this book accommodate typed linear algebras in a natural way.



Literature is vast!

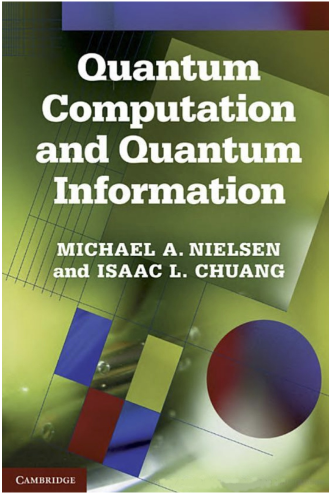
2011 (vintage year):

The 10th Anniversary Edition of the "classic" by Nielsen and Chuang.

But there is more — Selinger's **qPL** (2004) etc etc

Literature is vast.

This includes previous WG2.1 work in this field, see e.g. Mu and Bird (2001).



Opportunities for QP

"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	Control: Quantum
Data	Classic	–	x
	Quantum	x	x

What QP should (eventually) be

QP expected to evolve and reach maturity as the other bodies of knowledge in programming, requiring

- a **syntax**
- a **semantics**
- a **calculus**

This enables **program calculation**, the *scientific method* applied to software.

E.g., the **FP calculus** derives one version of the *Fibonacci recurrence relation* (aside) from the other.

$$
\begin{array}{l}
\mathit{fib}\ 0 = 0 \\
\mathit{fib}\ 1 = 1 \\
\mathit{fib}\ (n+2) = \mathit{fib}\ (n+1) + \mathit{fib}\ n
\end{array}
$$

$$
\begin{array}{l}
\mathit{fib}\ n = \\
\quad \mathbf{let}\ (x,y) = \mathbf{for}\ \mathit{loop}\ (0,1)\ n \\
\qquad \mathit{loop}\ (x,y) = (y, x+y) \\
\quad \mathbf{in}\ x
\end{array}
$$

Syntax, semantics, calculus (FP)

Syntax:

- **Sequential** composition ($f \cdot g$ seen above)
- **Parallel** composition ($f \triangledown g$) based on **pairing**
  $(f \triangledown g)\ x = (f\ x, g\ x)$
- **Alternative** composition $[f, g]$ — the dual of **pairing**
- Recursive solutions to **equations** involving the above.

Semantics, e.g. that of **pairing**

$$k = f \triangledown g \Leftrightarrow \left\{ \begin{array}{l} fst \cdot k = f \\ snd \cdot k = g \end{array} \right.$$

$$\begin{tikzcd} A & A \times B \arrow[l, "fst"'] \arrow[r, "snd"] & B \\ & C \arrow[ul, "f"] \arrow[u, "f \triangledown g"'] \arrow[ur, "g"'] & \end{tikzcd}$$

Syntax, semantics, calculus (FP)

Calculus, e.g. *fusion*,

$$(f \triangledown g) \cdot h = (f \cdot h) \triangledown (g \cdot h) \tag{1}$$

loss-less decomposition

$$k = (fst \cdot k) \triangledown (snd \cdot k) \tag{2}$$

reflection

$$fst \triangledown snd = id \tag{3}$$

pairwise *equality*

$$k \triangledown h = f \triangledown g \;\Leftrightarrow\; \left\{ \begin{array}{l} k = f \\ h = g \end{array} \right. \tag{4}$$

and so on.

Does this hold in the quantum world?

Quantum computations are all **reversible**.

This is expressed in linear algebra by **unitary** matrices (more about this later)

Isomorphisms are unitary and many quantum **gates** are isos

Yesterday we have seen how **pairing** differs from functions in the case of stochastic matrices

With matrices in general, **products** have to do with **coproducts** and not with **pairing** — this leads to **biproducts**.

Biproducts

In LA **coproducts** correspond to putting matrices side by side (horizontally):

$$X = [M|N] \Leftrightarrow \begin{cases} X.i_1 = M \\ X.i_2 = N \end{cases}$$

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

Products correspond to putting matrices side by side vertically:

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

$$\begin{tikzcd} A & A + B \arrow[l, "\pi_1"] \arrow[r, "\pi_2"] & B \\ & C \arrow[ul, "M"] \arrow[u, "{\left[\frac{M}{N}\right]}"] \arrow[ur, "N"'] & \end{tikzcd}$$

This is the basis of **block** operations in LA and leads to **direct sums**:
$M \oplus N = [i_1 \cdot M | i_2 \cdot N] = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix}$.

Quantum gates

Let us briefly show how standard **quantum programming gates**, used in **quantum circuits** (Nielsen and Chuang, 2011), can be expressed in typed LA.

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

Pairing (Khatri-Rao $\triangledown$ + Kronecker products $\otimes$)[1] is central to quantum data structuring.

Interestingly, **biproducts** can make quantum gates easier to understand and reason about, as I will briefly show.

[1]The latter is defined by $M \otimes N = (M \cdot fst) \triangledown (N \cdot snd)$.

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 $\left[\frac{a}{b}\right] = a \ \left[\frac{1}{0}\right] + b \ \left[\frac{0}{1}\right]$, the notation $a\,|0\rangle + b\,|1\rangle$ is normally used to denote **qubit** $\left[\frac{a}{b}\right]$.

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

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

$|\phi\rangle$ 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

$$
\begin{aligned}
& \begin{bmatrix} a \\ \hline b \end{bmatrix} \triangledown \begin{bmatrix} c \\ \hline d \end{bmatrix} \\
= \quad & \{ \text{ Khatri-Rao } \} \\
& \begin{bmatrix} ac \\ ad \\ bc \\ bd \end{bmatrix} \\
= \quad & \{ \text{ vector addition } \} \\
& \begin{bmatrix} ac \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ ad \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ bc \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ bd \end{bmatrix}
\end{aligned}
$$

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

$$\mathit{fst} \cdot \frac{|00\rangle + |01\rangle}{\sqrt{2}} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$
$$\mathit{snd} \cdot \frac{|00\rangle + |01\rangle}{\sqrt{2}} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

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

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:

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

Classic control (functional programming)

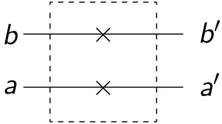
(Polymorphic) functional programming can play a nice role in quantum processing (perhaps not fully appreciated yet).

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

$$\begin{array}{cc|cccc} & & 0 & 0 & 1 & 1 \\ & & 0 & 1 & 0 & 1 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{array}$$



SWAP Gates

Applied to a qubit pair it will yield:

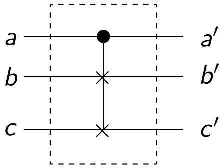
$$
\begin{array}{cl}
 & swap \cdot (a\ |00\rangle + b\ |01\rangle + c\ |10\rangle + d\ |11\rangle) \\
= & \qquad \{\ \text{expand to vector notation}\ \} \\
 & swap \cdot (a \begin{bmatrix}1\\0\\0\\0\end{bmatrix} + b \begin{bmatrix}0\\1\\0\\0\end{bmatrix} + c \begin{bmatrix}0\\0\\1\\0\end{bmatrix} +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

The Fredkin gate

This is a 3-(qu)bit gate, also known as "controlled"-SWAP. Let us see why.

Untyped description:

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



As a function (pointwise) of type $\mathit{fred} : 2 \times (2 \times 2) \to 2 \times (2 \times 2)$:

$$\begin{cases} \mathit{fred}\ (0, x) = (0, x) \\ \mathit{fred}\ (1, (b, c)) = (1, (c, b)) \end{cases}$$

Back to elementary AoP

$$\left\{ \begin{array}{l} fred \cdot (\underline{0} \triangledown id) = \underline{0} \triangledown id \\ fred \cdot (\underline{1} \triangledown id) = \underline{1} \triangledown swap \end{array} \right.$$

$$\Leftrightarrow \qquad \{ \text{ coproducts } \}$$

$$fred \cdot \underbrace{[\underline{0} \triangledown id | \underline{1} \triangledown id]}_{\alpha} = [\underline{0} \triangledown id | \underline{1} \triangledown swap]$$

$$\Leftrightarrow \qquad \{ \ \underline{1} \cdot swap = \underline{1} \text{ and so on } \}$$

$$fred \cdot \alpha = \alpha \cdot (id \oplus swap)$$

$$\Leftrightarrow \qquad \{ \text{ introduce } C = (id \oplus) \ \}$$

$$fred \cdot \alpha = \alpha \cdot C \ swap$$

that is, α translates *fred*kin into *C*ontrolled *swap*.

The α-isomorphism

The above is a faithful transformation

$$fred \xleftarrow{\alpha} C\ swap$$

because it relies on parametric **isomorphism**

$$\begin{aligned} &\alpha : A + A \rightarrow 2 \times A \\ &\alpha = [\underline{0} \triangledown id | \underline{1} \triangledown id]) \end{aligned} \tag{5}$$

that (by the exchange law) can also be written

$$\alpha = [\underline{0}|\underline{1}] \triangledown [id|id] \tag{6}$$

with **natural** property:

$$\alpha \cdot (M \oplus M) = (id \otimes M) \cdot \alpha \tag{7}$$

For free we obtain that *fred* is an **isomorphism** too.

The α-isomorphism

The translation from a pair-wise description of the Fredkin gate to a copair-wise one is so useful that we define, for every $M : 2 \times A \to B$, the corresponding $M' : A + A \to B$ defined by

$$M' = M \cdot \alpha \tag{8}$$

This transform has a number of immediate properties:

$$(M \triangledown N)' = M' \triangledown N' \tag{9}$$

$$id' = \alpha \tag{10}$$

$$!' = ! \tag{11}$$

$$fst' = ! \oplus ! \tag{12}$$

$$xor' = [id|\neg] \tag{13}$$

an so on.

The CNOT gate

Think now of implementing $\mathit{xor} : 2 \times 2 \to 2$ defined by

$$\mathit{xor}\ (0, a) = a$$
$$\mathit{xor}\ (1, a) = \neg\ a$$

Clearly, this gate is not **injective** — e.g. $\mathit{xor}\ (0,0) = \mathit{xor}\ (1,1)$ — and therefore not **reversible**.

The laws of quantum physics give no hope for non-reversible computations, so how can such an important classical gate be incorporated in quantum circuits?

The trick is to add a "garbage" bit to the output, which becomes type 2×2 making room for a reversible computation:

$$\mathit{xor}_{\_}\ (a, b) = (a, \mathit{xor}\ (a, b))$$

How can we be sure this is reversible (isomorphism)?

Easy (indirect) calculation

$$
\begin{array}{cl}
 & xor\_' \\
= & \{ \text{ definition } \} \\
 & (fst \triangledown xor)' \\
= & \{ \text{ laws given } \} \\
 & (! \oplus !) \triangledown [id | \neg] \\
= & \{ \ ! \triangledown M = M \text{ etc } \} \\
 & [i_1 | i_2 \cdot \neg] \\
= & \{ \text{ coproducts } \} \\
 & id \oplus \neg \\
= & \{ \text{ defined above } \} \\
 & C \ \neg
\end{array}
$$

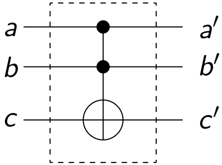
Because α is an isomorphism, f is **injective** iff $f' = f \cdot \alpha$ is so.

So the calculation on the left is an indirect way of proving that *fst* is a **constant complement** of *xor* (Matsuda et al., 2007).

No need for **pointwise** arguments!

Toffoli gate ($CCNOT$)

Diagram:



As a functional program:

$$
\begin{aligned}
&\mathit{toffoli} : (2 \times 2) \times 2 \to (2 \times 2) \times 2 \\
&\mathit{toffoli}\ ((1,1), c) = ((1,1), \neg\, c) \\
&\mathit{toffoli}\ ((a,b), c) = ((a,b), c)
\end{aligned}
$$

the same as

$$\mathit{toffoli}\ ((a,b), c) = ((a,b), \mathit{xor}\ (a \wedge b, c))$$

A very useful gate, cf.

$$
\begin{aligned}
&\mathit{toffoli}\ ((a,1), c) = ((a,1), \mathit{xor}\ a\ c) \text{ — } \textit{implements}\ \mathit{xor} \\
&\mathit{toffoli}\ ((a,b), 0) = ((a,b), a \wedge b) \text{ — } \textit{implements}\ \wedge \\
&\mathit{toffoli}\ ((1,1), c) = ((1,1), \neg\, c) \text{ — } \textit{implements}\ \neg
\end{aligned}
$$

Toffoli gate

Pointfree *toffoli*, as a linear algebra expression:

$$\begin{aligned} &\mathit{toffoli} : (2 \times 2) \times 2 \to (2 \times 2) \times 2 \\ &\mathit{toffoli} = \mathit{fst} \vartriangledown (\mathit{xor} \cdot (\wedge \otimes \mathit{id})) \end{aligned} \tag{14}$$

Again note how far the *"useful part'"* of *toffoli*

$$\mathit{xor} \cdot (\wedge \otimes \mathit{id}) \quad = \quad \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

is from being a **reversible** computation, as all quantum gates should be.

The trick will be the same — pair with constant complement $(\mathit{fst} \vartriangledown \_)$.

In general, functional **pairing** always increases **injectivity** because

$$(f \vartriangledown g)^\circ \cdot (f \vartriangledown g) = (f^\circ \cdot f) \times (g^\circ \cdot g) \tag{15}$$

and $f^\circ \cdot f$ (the **kernel** of f) measures the injectivity of f.

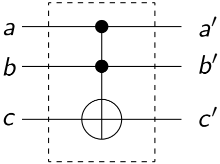
Toffoli gate

Altogether, for $A = 2$ in the type of α:

$$
\begin{tikzcd}
(2+2)+(2+2) \arrow[rr, "toffoli"] \arrow[d, "\alpha\oplus\alpha"'] & & (2+2)+(2+2) \\
(2\times 2)+(2\times 2) \arrow[d, "\alpha"'] & & (2\times 2)+(2\times 2) \arrow[u, "(\alpha\oplus\alpha)^\circ"'] \\
(2\times 2)\times 2 \arrow[rr, "X"'] & & (2\times 2)\times 2 \arrow[u, "\alpha^\circ"']
\end{tikzcd}
$$

Calculations (omitted for brevity) will derive, from the diagram above,

$$X = id \oplus (id \oplus \neg) = C\ (C\ \neg))$$



where $2 \xrightarrow{\neg} 2 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ — a much simpler version of (14).

Controlled-U gates

Generalizing, **controlled-U** gates are captured by the **generic** combinator:

$$C : (A \to B) \to (2 + A \to 2 + B)$$
$$C\ (U) = id \oplus U$$

where $2 \xrightarrow{id} 2$, whereby

$$fredkin = C\ swap$$
$$xor_{\_} = C\ \neg$$
$$toffoli = C\ (C\ \neg)$$

and so on. Moreover, $C\ \_$ is a **functor**:

$$C\ id = id \tag{16}$$
$$C\ (M \cdot N) = (C\ M) \cdot (C\ N) \tag{17}$$

Unitary gates

The isomorphisms (reversible functions) we have seen so far are special cases of so-called **unitary** matrices.

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

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.

My first (tentative) quantum 'adjoint fold'

Suppose we want to apply the Hadamard gate

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

n times to qubit q — a kind of **for**-loop:

$$\textbf{for}\ H\ n\ q$$

Expanding — and ignoring the quantum essence of H for the moment:

$$\textbf{for}\ H\ 0\ q = q$$
$$\textbf{for}\ H\ (n+1)\ q = H\ (\textbf{for}\ H\ n\ q)$$

My first (tentative) quantum 'adjoint fold'

From the type of **for** $H : \mathbb{N}_0 \to 2 \to 2$, or the isomorphic $\mathbb{N}_0 \times 2 \to 2$, we see that **for** H cannot be reversible.

But type $\mathbb{N}_0 \times 2 \to \mathbb{N}_0 \times 2$ would have room to accommodate one such reversible computation.

So we go for the uncurried version, paired with *fst* as before — the **constant complement** trick again — abbreviated to f:

$$f\ (0, q) = (0, q)$$
$$f\ (n + 1, q) = (n + 1, H\ (snd\ (f\ (n, q))))$$

Because H is not a function, we have to go **pointfree** (in LA) to have the right definition:

$$f \cdot (\underline{0} \otimes id) = \underline{0} \otimes id$$
$$f \cdot (succ \otimes id) = (succ \cdot fst) \triangledown (H \cdot snd \cdot f)$$

My first (tentative) quantum 'adjoint fold'

Putting the two lines together using **coproducts** and converse, we get the **recursive** matrix,

$$f = [\underline{0} \otimes id | (succ \cdot fst) \triangledown (H \cdot snd \cdot f)] \cdot (([\underline{0}|succ] \otimes id) \cdot \beta)^\circ$$

cf. diagram

$$\begin{tikzcd}
\mathbb{N}_0 \times 2 \arrow[d, "f"] & (1 + \mathbb{N}_0) \times 2 \arrow[l, "{[\underline{0}|succ]\otimes id}"'] & 1 \times 2 + \mathbb{N}_0 \times 2 \arrow[l, "\beta"'] \arrow[d, "id + fst \triangledown f"] \\
\mathbb{N}_0 \times 2 & & 1 \times 2 + \mathbb{N}_0 \times (\mathbb{N}_0 \times 2) \arrow[ll, "{[\underline{0}\otimes id | succ \times H \cdot snd]}"]
\end{tikzcd}$$

where β is the obvious isomorphism.

My first (tentative) quantum 'adjoint fold'

To test this quantum program in MATLab we fix an approximation to $\mathbb{N}_0$, for instance we restrict to n-bit numbers in $\{0\,..\,2^n-1\}$, taking $\beta = id$ valid at matrix level into account:

$$\begin{tikzcd}
2^n \times 2 \arrow[r, "([\underline{0}|succ]\otimes id)^\circ"] \arrow[d, "f"] & 1 \times 2 + 2^n \times 2 \arrow[d, "id + fst \triangledown f"] \\
2^n \times 2 & 1 \times 2 + 2^n \times (2^n \times 2) \arrow[l, "{[\underline{0}\otimes id|succ \times H \cdot snd]}"]
\end{tikzcd}$$

Then we iterate over

$$f = \underbrace{[\underline{0} \otimes id|(succ \cdot fst) \triangledown (H \cdot snd \cdot f)] \cdot ([\underline{0}|succ] \otimes id)^\circ}_{F\ f}$$

starting with $f_0 = \ 2^n \times 2 \xleftarrow{\perp} 2^n \times 2$ holding zeros only.

MatLab checking

In 2^n iterations we reach the fixpoint:

```
>> f4=[ (kron(z,id)) kr(s*fst(4,2),H*snd(4,2)*f3) ]*out

f4 =
    1.0000         0         0         0         0         0         0         0
         0    1.0000         0         0         0         0         0         0
         0         0    0.7071    0.7071         0         0         0         0
         0         0    0.7071   -0.7071         0         0         0         0
         0         0         0         0    1.0000         0         0         0
         0         0         0         0         0    1.0000         0         0
         0         0         0         0         0         0    0.7071    0.7071
         0         0         0         0         0         0    0.7071   -0.7071
```

Note the type $f_4 : 2^n \times 2 \to 2^n \times 2$.

Since it is a real valued, symmetric matrix, it is **unitary**.

MatLab checking

Suppose we want to check the outcome of **for** $H\ 3\ (|0\rangle - |1\rangle)$.

We obtain this by running $f_4 \cdot (\underline{3} \triangledown \begin{bmatrix} \frac{1}{-1} \end{bmatrix}) = f_4 \cdot (\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \triangledown \begin{bmatrix} \frac{1}{-1} \end{bmatrix})$, cf.

```
>> x=f4*kr(n3,[1;-1])
x =
         0
         0
         0
         0
         0
         0
         0
    1.4142
```

This corresponds to the pair $(3, |0\rangle + \frac{|1\rangle}{\sqrt{2}})$, as expected.

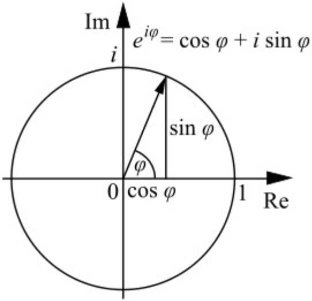
Phase-shift gates

The for-loop above gets more interesting when we replace the Hadamard gate by the so-called **phase shift gate** defined by

$$R_\phi = (\sigma\ \phi) \triangledown id = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\ \phi} \end{bmatrix}$$

where

$$\sigma\ \phi = [1\ e^{i\ \phi}].$$



Recalling $e^{i\ \phi} = cos\ \phi + i\ sin\ \phi$ the fixpoint of

$$f = \underbrace{[\underline{0} \otimes id | (succ \cdot fst) \triangledown (R_{\frac{\pi}{6}} \cdot snd \cdot f)] \cdot ([\underline{0}|succ] \otimes id)^{\circ}}_{F\ f}$$

for the same types is given in the next slide.

Iterating a phase-shift gate

$$f_4 = \left[\begin{array}{cc|cc|cc|cc}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.867 + 0.5i & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.5 + 0.867i & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & i
\end{array}\right]$$

Complex matrix f_4 is unitary.

Note the effect of the constant complement $(fst \triangledown \_)$ shifting the corresponding iteration of gate $R_{\frac{\pi}{6}}$ along the diagonal.

Summary

Very experimental still.

Previous work on stochastic folds in LA (Murta and Oliveira, 2015) showed that we can have **quantitative** algebras of programming

Would like to investigate the AoP of **unitary** (recursive) matrices now

Towards **correct by construction** quantum programs, who knows...

Need to investigate Ralph Hinze's "Adjoint folds" not in CCCs but rather in MCCs

Also studying previous WG2.1 work in the field (Mu and Bird, 2001)

Acknowledgement

The idea of a *"linear algebra of programming"*, which underlies this talk, was first put forward by Amílcar Sernadas (1952–2017), the key idea being

> *"to adopt linear algebra as the lingua franca of* **software verification***" (SQIG-Group, 2011).*

I thank Amílcar for his friendship and good advice, and his research group for this and so many other inspirations.

References

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

Ralf Hinze. Adjoint folds and unfolds — an extended study. *Science of Computer Programming*, 78(11):2108–2159, 2013. ISSN 0167-6423.

K. Matsuda, Z. Hu, K. Nakano, M. Hamana, and M. Takeichi. Bidirectionalization transformation based on automatic derivation of view complement functions, 2007. 12th ACM SIGPLAN International Conference on Functional Programming (ICFP 2007), Freiburg, Germany, October 1-3.

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

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

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

Peter Selinger. Towards a quantum programming language. *Mathematical. Structures in Comp. Sci.*, 14(4):527–586, August 2004. ISSN 0960-1295. URL https://doi.org/10.1017/S0960129504004256.

A. Sernadas, J. Ramos, and P. Mateus. Linear algebra techniques for deciding the correctness of probabilistic programs with bounded resources. Technical report, TU Lisbon, 1049-001 Lisboa, Portugal, 2008.

SQIG-Group. LAP: Linear algebra of bounded resources programs, 2011. IT & Tech. Univ. Lisbon. URL: http://sqig.math.ist.utl.pt/work/LAP.