Towards quantamorphisms — some thoughts on (constructive) reversibility

NII, Tokyo

23rd of January, 2018

J.N. Oliveira
(joint work with R.S. Barbosa and A. Neri)

INESC TEC & University of Minho

Thanks to NII!

Trip planned long ago...

Long friendship — Zhenjiang, can you remember GTTSE'05?

We had met before — cf. (Mu et al., 2004), which relates to this talk!

Summer School on

Generative and Transformational Techniques

in Software Engineering

4 - 8 July, 2005, Braga, Portugal

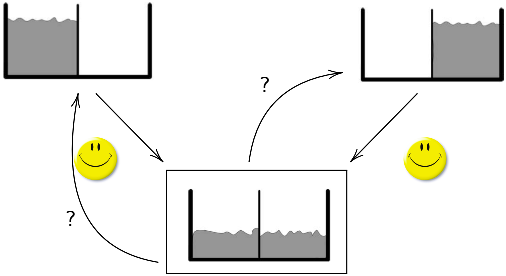
http://www.di.uminho.pt/GTTSE2005

Computing versus energy

Thermodynamics view of **computing**.

Green computing calling for less energy consumption.

LANDAUER'S PRINCIPLE: **irreversible** computation accounted for **energy consumption** (entropy).



Information is physical

Physics of information — a branch of science.

Quantum computing — a **quantum mechanics** view of computation (**bijective** transformations → **unitary** transformations).

Bidirectional programming (BX) 

AIM — achieve reversible / quantum programming **constructively**.

Inspiration from **functional** programming.

Algebra of Reversible / Quantum Programming? Yes — **LAoP**, a **linear** algebra of programming.

Ut facient opus signa

("Let symbols do the work")

[...] by the aid of symbolism, we can make transitions in reasoning almost **mechanically** by the eye
[...] Civilisation advances by extending the number of important operations which can be performed **without thinking** about them."

(Alfred Whitehead, 1911)

Start from BX (total, functional)

$$\begin{tikzcd}
 & S \arrow[dl, "id"'] \arrow[d, "get \triangledown id"] \\
S \arrow[d, "get"'] & V \times S \arrow[l, "put"'] \arrow[dl, "fst"] \\
V &
\end{tikzcd}$$

GETPUT:

$$get \cdot put = fst \tag{1}$$

PUTGET:

$$put \cdot (get \triangledown id) = id \tag{2}$$

Composition combinator:

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

Identity:

$$id\ x = x$$

Pairing combinator:

$$(f \triangledown g)\ x = (f\ x, g\ x)$$

Projections:

$$fst\ (a, b) = a \quad snd\ (a, b) = b$$

Calculating properties of *get+put*

PUTGET ensures that *put* is **surjective**,

$$\langle \forall\ s\ ::\ \langle \exists\ v, s'\ ::\ s = put\ (v, s') \rangle \rangle$$

since $\underbrace{f}_{surjective} \cdot \underbrace{g}_{injective} = id$ in general.

Moreover, *get* is also **surjective** and **uniquely** determined by *put*. Why and how?

To answer these questions we have to do our first generalization:

> *"(...) like the move from* **real** *numbers to* **complex** *ones, the move [from* **functions***] to* **relations** *increases our powers of expression"* (Bird and de Moor, 1997)

Calculating properties of *get+put*

We generalize $y = f\ x$ to $y\ R\ x$, and use the same arrows to denote both, e.g. $X \xrightarrow{f} Y$ and $X \xrightarrow{R} Y$.

Some people like writing $y\ R\ x \Leftrightarrow (y, x) \in R$, but we simply read $y\ R\ x$ as *"it is true that y is related to x by R"*; or simply, *"$y\ R\ x$ holds"*.

John Loves Mary. $2 < 3$. As simple as that.

To say that *Mary* is loved by *John* simply write *Mary Loves*$^\circ$ *John*.

In general: $y\ R\ x \Leftrightarrow x\ R^\circ\ y$ — this is the **converse** operation, or *passive voice*:

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ$$
$$id^\circ = id$$

Composition generalizes to $y\ (R \cdot S)\ x \Leftrightarrow \langle \exists\ z\ ::\ y\ R\ z \wedge z\ S\ x \rangle$.

Calculating properties of *get*+*put*

The other ingredient of the generalization is that **relations** are ordered by a partial order, $R \subseteq S \Leftrightarrow \langle \forall\, y, x \ :: \ y\, R\, x \Rightarrow y\, S\, x \rangle$.

Functions are the **only** relations f, g such that the following hold:

$$f \cdot R \subseteq S \Leftrightarrow R \subseteq f^{\circ} \cdot S \tag{3}$$

$$f \subseteq g \Leftrightarrow f = g \Leftrightarrow g \subseteq f \tag{4}$$

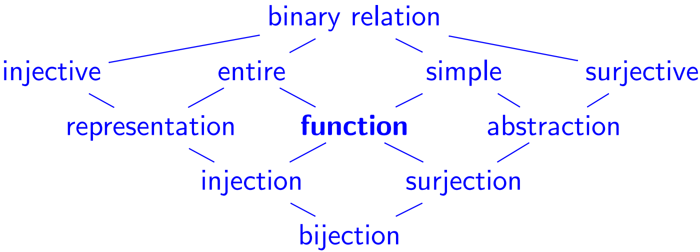
Convention: functions in **lowercase**, general relations in **uppercase**.

Taking converses,

$$R \cdot f^{\circ} \subseteq S \Leftrightarrow R \subseteq S \cdot f \tag{5}$$

also holds. Why do functions enjoy such nice **shunting** rules?

Relation bestiary



where

$$R \ \textbf{injective} \Leftrightarrow \underbrace{R^\circ \cdot R}_{\ker\ R} \subseteq id \qquad R \ \textbf{simple} \Leftrightarrow R^\circ \ \textbf{injective}$$

$$R \ \textbf{entire} \Leftrightarrow id \subseteq \underbrace{R \cdot R^\circ}_{\mathrm{img}\ R} \qquad R \ \textbf{surjective} \Leftrightarrow R^\circ \ \textbf{entire}$$

Relations as matrices

It helps if we depict relations using (Boolean) **matrices**, for instance **negation** (a bijection) $\neg = \begin{array}{c|cc} & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}$

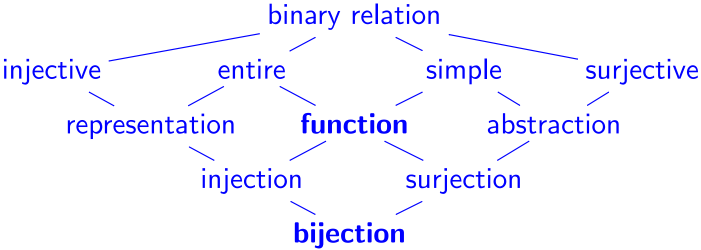
exclusive-or (surjective but not injective): $(\dot{\vee}) = \begin{array}{c|cccc} & 0 & 0 & 1 & 1 \\ & 0 & 1 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{array}$

and so on.

Functions have **exactly one** 1 in every column.

Bijections have exactly one 1 in every **column** and in every **row**.

Functions, bijections, etc



Thus

$$f \ \textbf{function} \Leftrightarrow \mathrm{img}\ f \ \subseteq\ id \wedge id \ \subseteq\ \ker\ f$$

$$f \ \textbf{bijection} \Leftrightarrow f^{\circ} \ \textbf{function} \Leftrightarrow \mathrm{img}\ f = id \wedge id = \ker\ f$$

These are the properties ensure the rules given earlier for functions.

Re-write GETPUT and PUTGET

By such rules, GETPUT re-writes to

$$get \cdot put = fst \quad \Leftrightarrow \quad \left\{ \begin{array}{l} put \;\subseteq\; get^{\circ} \cdot fst \\ fst \cdot put^{\circ} \;\subseteq\; get \end{array} \right.$$

and PUTGET to

$$put \cdot (g \mathbin{\triangledown} id) = id \quad \Leftrightarrow \quad g \mathbin{\triangledown} id \;\subseteq\; put^{\circ}$$

From this we infer:

- get is **surjective** — because put° and fst are so, and thumb rule: **larger than surjective is surjective**.
- put **determines** get — if some other get' exists, $get = get'$ — next slide.

put determines *get*

$$
\begin{array}{ll}
& true \\
\Leftrightarrow & \quad \{ \text{ PUTGET of new } get' \ \} \\
& put \subseteq get'^{\circ} \cdot fst \\
\Rightarrow & \quad \{ \text{ monotonicity } \} \\
& put \cdot (get \triangledown id) \subseteq get'^{\circ} \cdot fst \cdot (get \triangledown id) \\
\Leftrightarrow & \quad \{ \text{ PUTGET of first } get \ \} \\
& id \subseteq get'^{\circ} \cdot fst \cdot (get \triangledown id) \\
\Leftrightarrow & \quad \{ \text{ shunting, } fst \cdot (f \triangledown g) = f \ \} \\
& get' \subseteq get \\
\Leftrightarrow & \quad \{ \text{ function equality } \} \\
& get' = get
\end{array}
$$

$\square$

Bad *puts*...

However, some *put*s have no *get*. Why?

Recall GETPUT in version

$$fst \cdot put^{\circ} \subseteq get$$

As *get* is **simple**, and **smaller than simple is simple**, $fst \cdot put^{\circ}$ has to be simple too:

$$
\begin{array}{ll}
& fst \cdot put^{\circ} \ \textbf{simple} \\
\Leftrightarrow & \quad \{ \ R \ \textbf{simple} \Leftrightarrow R \cdot R^{\circ} \subseteq id \ \} \\
& fst \cdot put^{\circ} \cdot put \cdot fst^{\circ} \subseteq id \\
\Leftrightarrow & \quad \{ \ \text{shunting rules} \ \} \\
& put^{\circ} \cdot put \subseteq fst^{\circ} \cdot fst \\
\Leftrightarrow & \quad \{ \ \textbf{injectivity preorder}: \ R \leqslant S \Leftrightarrow \ker S \subseteq \ker R \ \} \\
& fst \leqslant put
\end{array}
$$

put more injective than *fst* (sorry!)

COUNTER-EXAMPLE:

Is **exclusive-or**

$$(\dot{\vee}) : 2 \times 2 \to 2$$
$$(\dot{\vee}) = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

a good *put*? **No**! — just compute

$$fst \cdot (\dot{\vee}^\circ) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \top$$

and observe that it is not simple.[1]

[1]We denote by $B \xleftarrow{\top} A$ the largest relation of type $B \longleftarrow A$.

put more injective than *fst* (sorry!)

The same counter-example using the injectivity preorder:

$$
\begin{array}{ll}
 & \mathit{fst} \leqslant (\dot{\vee}) \\
\Leftrightarrow & \quad \{ \; R \leqslant S \Leftrightarrow \ker S \subseteq \ker R \; \} \\
 & \ker\,(\dot{\vee}) \subseteq \ker\,\mathit{fst} \\
\Leftrightarrow & \quad \{ \; \text{kernel matrices} \; \} \\
 & \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \end{bmatrix} \subseteq \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \\
\Leftrightarrow & \quad \{ \; \text{pointwise inclusion} \; \} \\
 & \mathit{false}
\end{array}
$$

How to design a (good) put?

To obtain a good $put : V \times S \to S$,

- **refine** $fst : V \times S \to V$ according to the **injectivity preorder** — i.e. find put s.t. $fst \leqslant put$.
- Then obtain $get : S \to V$ by computing $fst \cdot put^{\circ}$.

Example: starting point for a good $2 \times 3 \xrightarrow{put} 3$ is

$$\ker\left(2 \xleftarrow{fst} 2 \times 3 \right) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Designing a good *put*

Note that $\ker\ put$ must have 3 **equivalence** classes ($\#S = 3$) because *put* is **surjective**.

Since $\ker\ fst$ has 2 equivalence classes (fst surjective, $\#V = 2$), the best we can do is to split one of these in two, eg.

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

that is:

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

Designing a good *put*

As this is a **good** *put* **by construction**, its *get* is immediately calculated:[2]

$$get = fst \cdot put^{\circ} = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

That is:

$$\begin{array}{l} put\ (a, 1) = 1 \\ put\ (a, 2) = put\ (a, 3) = 2 \\ put\ (b, \_) = 3 \end{array}$$

$$\begin{array}{l} get\ 1 = get\ 2 = a \\ get\ 3 = b \end{array}$$

(We make $V = \{a, b\}$ just for visualizing V and S differently.)

[2] Note that $fst \cdot put^{\circ}$ is always **entire** because *put* is **surjective**.

Designing a good *put*

Exercise: *How many good puts there are of type* $3 \times 4 \rightarrow 4$? *And what is the corresponding get? Start from*

$$ker\ (\ 3 \xleftarrow{fst} 3 \times 3\ ) = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

and refine.

Going partial

As in (Ko and Hu, 2018), BX become more general once we **drop totality** (entireness).

Thus *put* and *get* become just **simple** relations (= **partial functions**) P and G with $\textsc{GetPut}+\textsc{PutGet}$

$$P \subseteq G^{\circ} \cdot fst \tag{6}$$

$$G \triangledown id \subseteq P^{\circ} \tag{7}$$

by immediate generalization of what we had before:

$$put \subseteq get^{\circ} \cdot fst$$
$$get \triangledown id \subseteq put^{\circ}$$

Here is how $\textsc{GetPut}+\textsc{PutGet}$ (6,7) read with variables:

$$s' \; P \; (v, s) \Rightarrow v \; G \; s'$$
$$v \; G \; s \Rightarrow s \; P \; (v, s)$$

Going (more) injective

As we did with $fst \leqslant put$, we are now interested in further exploiting the **injectivity** preorder,

$$R \leqslant S \Leftrightarrow \ker\ S \ \subseteq \ \ker\ R$$

as a **refinement** ordering guiding us towards more and more **injective** computations — the way to **reversibility**.

This ordering is rich in properties, for instance it is upper-bounded[3]

$$R \triangledown S \leqslant X \quad \Leftrightarrow \quad R \leqslant X \wedge S \leqslant X \tag{8}$$

[3] Details in (Oliveira, 2014). **NB**: pairing generalizes to relations in the expected way: $(b, c)\ (R \triangledown S)\ a \Leftrightarrow b\ R\ a \wedge c\ S\ a$.

Going (more) injective

Therefore, by cancellation of (8), we have that **pairing** always **increases injectivity**:

$$R \leqslant R \triangledown S \quad \text{and} \quad S \leqslant R \triangledown S. \tag{9}$$

The inclusion $\ker\,(R \triangledown S) \;\subseteq\; (\ker\,R) \cap (\ker\,S)$ is in fact an equality

$$\ker\,(R \triangledown S) = (\ker\,R) \cap (\ker\,S)$$

itself a corollary of the more general:

$$(R \triangledown S)^{\circ} \cdot (Q \triangledown P) = (R^{\circ} \cdot Q) \cap (S^{\circ} \cdot P) \tag{10}$$

Injectivity **shunting laws** also exist, e.g.

$$R \cdot g \leqslant S \quad \Leftrightarrow \quad R \leqslant S \cdot g^{\circ}$$

Ordering functions by injectivity

Restricted to **functions**, $(\leqslant)$ is **universally** bounded by

$$! \leqslant f \leqslant id$$

where $1 \xleftarrow{\;!\;} A$ is the unique function of its type. (1 is the singleton type.) Moreover,

- A function is **injective** iff

  $$id \leqslant f$$

  Thus $f \triangledown id$ is always **injective** (9).

- Two functions f e g are said to be **complementary** wherever $id \leqslant (f \triangledown g)$.[4]

For instance, fst and snd are complementary since $fst \triangledown snd = id$.

---

[4] Cf. (Matsuda et al., 2007). Other terminologies are **monic pair** (Freyd and Scedrov, 1990) or **jointly monic** (Bird and de Moor, 1997).

Minimal complements

Minimal complements — *Suppose (a)* $id \leqslant f \triangledown g$ *; (b) if* $id \leqslant f \triangledown h$ *and* $h \leqslant g$ *then* $g \leqslant h$.

Then g *is said to be a* **minimal complement** *of* f *(Bancilhon and Spyratos, 1981).*

Minimal complements (not unique in general) characterize "what is missing" in the original function for **injectivity** to hold.

EXAMPLE: Non-injective $2 \xleftarrow{\dot{\vee}} 2 \times 2 = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$ has minimal complement $2 \xleftarrow{fst} 2 \times 2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$.

How can we be sure it is minimal?

Minimal complements

We start from

$$\ker\,(\dot{\vee}) = \mathsf{ker}\begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 1 & 0\end{bmatrix} = \begin{bmatrix}1 & 0 & 0 & 1\\ 0 & 1 & 1 & 0\\ 0 & 1 & 1 & 0\\ 1 & 0 & 0 & 1\end{bmatrix}$$

Clearly, $\ker\,g$ has to cancel all 1's that fall outside the diagonal,

$$\ker\,g = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

but this is an overkill — $g = id$ in this case!

We can add 1s where $\ker\,(\dot{\vee})$ has 0's, e.g. $\begin{bmatrix}1 & 1 & 1 & 0\\ 1 & 1 & 0 & 1\\ 1 & 0 & 1 & 1\\ 0 & 1 & 1 & 1\end{bmatrix}$ but this isn't a kernel anymore — why?

Minimal complements

Kernels of functions are **equivalence relations — reflexive** (cf. diagonal), **symmetric** and **transitive.**

How do we ensure this?

By ensuring that the matrix depicts a **rational**, or **difunctional** relation:

A relation R *is* **difunctional** *iff* $R \cdot R^{\circ} \cdot R \subseteq R$.

FACT: a symmetric+reflexive relation is an **equivalence** iff it is difunctional.

One can construct **difunctional** relations easily: just make sure that columns either don't intersect or are the same.

Ensuring difunctionality

Cancel zeros symmetrically, outside the diagonal:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} = \ker \mathit{fst}$$

Alternatively:

$$\begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \to \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = \ker \mathit{snd}$$

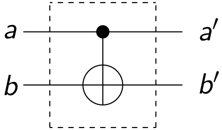
So, both *fst* and *snd* are **minimal complements** of $\dot{\vee}$.

Complementing $(\dot{\vee})$

What do we get by complementing $(\dot{\vee})$ with *fst*:

$$\mathbf{2} \times \mathbf{2} \xleftarrow{fst^{\triangledown}\dot{\vee}} \mathbf{2} \times \mathbf{2} \;=\; \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad ?$$

This is a well-known **bijection**, in fact a (classical) **quantum gate** known as **CNOT** (for "*controlled not*") and depicted as follows:



Why does it bear this name?

Complementing ($\dot{\vee}$)

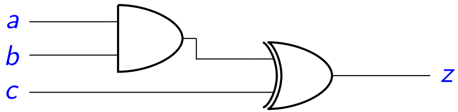
$$
\begin{array}{ll}
 & cnot = fst \triangledown (\dot{\vee}) \\
\Leftrightarrow & \qquad \{ \text{ pointwise } \} \\
 & cnot\ (a, b) = (a, a\ \dot{\vee}\ b)) \\
\Leftrightarrow & \qquad \{ \text{ since } 0\ \dot{\vee}\ b = b \text{ and } 1\ \dot{\vee}\ b = \neg\ b \ \} \\
 & \left\{ \begin{array}{l} cnot\ (0, b) = (0, b) \\ cnot\ (1, b) = (1, \neg\ b) \end{array} \right.
\end{array}
$$

Informally: **controlled** bit b is negated *iff* the **control** bit a is set.

Thus we have a **constructive** approach to designing this gate — we build it by **minimal** complementation. (Not the standard interpretation!)

Other *fst*-complementations

Take the classical circuit



Can it be made into a bijection in the same way?

The function implemented is

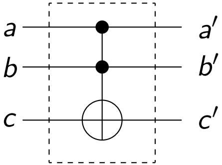
$$2^2 \times 2 \xrightarrow{f = \dot{\vee} \cdot (\wedge \otimes id)} 2 \quad = \quad \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Let us **complement** it with $2^2 \times 2 \xrightarrow{fst} 2^2$ again. (Next slide.)

Other *fst*-complementations

We get another **bijection**, known as the **CCNOT** gate :

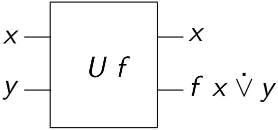
$$\mathit{ccnot} = \mathit{fst} \triangledown (\dot{\vee} \cdot (\wedge \otimes \mathit{id})) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$



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

Other *fst*-complementations

A famous device in quantum programming is the following evolution of the *CNOT* gate,



parametric on $2 \xrightarrow{f} 2$:

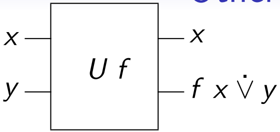
$$U\ f = fst \triangledown (\dot{\vee} \cdot (f \times id))$$

where

$$(f \times g)\ (a, b) = (f\ a, f\ b)$$

Clearly, $cnot = U\ id$.

Other *fst*-complementations



is **bijective** because it is its self inverse:

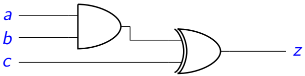
$$
\begin{array}{ll}
& (U\ f) \cdot (U\ f) = id \\
\Leftrightarrow & \qquad \{\ U\ f\ (x,y) = (x, f\ x \mathbin{\dot\vee} y)\ \} \\
& U\ f\ (x, f\ x \mathbin{\dot\vee} y) = (x,y) \\
\Leftrightarrow & \qquad \{\ \text{again } U\ f\ (x,y) = (x, f\ x \mathbin{\dot\vee} y)\ \} \\
& (x, f\ x \mathbin{\dot\vee} (f\ x \mathbin{\dot\vee} y)) = (x,y) \\
\Leftrightarrow & \qquad \{\ \dot\vee \text{ is associative and } x \mathbin{\dot\vee} x = 0\ \} \\
& (x, 0 \mathbin{\dot\vee} y) = (x,y) \\
\Leftrightarrow & \qquad \{\ 0 \mathbin{\dot\vee} x = x\ \} \\
& (x,y) = (x,y)
\end{array}
$$

$\square$

Pause

What have we achieved thus far?

A **constructive** approach to **reversibility** — instead of accepting (e.g. quantum gates as) "inventions", we start (**functionally**) from the functions that we want to make available, e.g.



and then **refine** them into reversible programs by pairing them with minimal **complements**.

That is, the original gate is taken as **specification**, the reversible one as **implementation**.

Never forget to **program from specifications** (Morgan, 1990).

The role of $A \xleftarrow{fst} A \times B$

We have seen that $A \xleftarrow{fst} A \times B$ plays a prominent role in the calculations thus far.

The starting point for calculating $S \xleftarrow{put} V \times S$

$$fst \leqslant put$$

is a property known as the **semi-injectivity** of put (Foster et al., 2007):

$$put(a, c) = put(a', c') \Rightarrow a = a'$$

(Just unfold $fst \leqslant put$ and go pointwise.)

fst is often a good minimal complement — can $(fst \triangledown \_)$ be extended **recursively**?

Going general (recursive)

Our examples have been fortunate in the sense that projection $A \times B \xrightarrow{fst} A$ was paired with a function of type $A \times B \longrightarrow B$, making room for a **bijection** of type $A \times B \longrightarrow A \times B$.

Suppose we want to offer **arbitrary** $f : A \rightarrow B$ in a **bijective** "envelope" (that's what reversible/quantum computing is all about).

The "smallest" (generic) type for such an enveloped function is $A \times B \rightarrow A \times B$.

Now suppose f is a recursive function, e.g. $f = \mathbf{foldr}\ g\ b$. How do we "constructively" build the corresponding (recursive, bijective) envelope of type $[A] \times B \rightarrow [A] \times B$?

Going general (folds)

Let us define $(\!|f|\!)\ (x, b) = \textbf{foldr}\ \overline{f}\ \ b\ x$ where $\overline{f}\ a\ b = f\ (a, b)$:

$$
\begin{aligned}
(\!|f|\!)\ ([\,], b) &= b \\
(\!|f|\!)\ (a : x, b) &= f\ (a, (\!|f|\!)\ (x, b))
\end{aligned}
$$

Thus

$$
\begin{tikzcd}
{[A] \times B} \arrow[d, "(\!|f|\!)"'] & & B + A \times ([A] \times B) \arrow[ll, "\alpha"'] \arrow[d, "id + id \times (\!|f|\!)"] \\
B & & B + A \times B \arrow[ll, "{[id\,, f]}"]
\end{tikzcd}
$$

NB:

$$X + Y = \{\, i_1\ x \mid x \in X \,\} \cup \{\, i_2\ y \mid y \in Y \,\}$$

is **disjoint** union of X and Y — thanks to $i_1 \cdot i_2^\circ = \bot$ — and $[R, S]$ is the **unique** relation X such that $X \cdot i_1 = R$ and $X \cdot i_2 = S$.

Going general ($\mathbb{N}_0$)

Let us start from a simpler fold, that over natural numbers ($\mathbf{for}\ f\ i\ n = f^n\ i$):

$$\mathbf{for}\ f\ i\ 0 = i$$
$$\mathbf{for}\ f\ i\ (n+1) = f\ (\mathbf{for}\ f\ i\ n)$$

Via the same procedure, this becomes

$$\begin{tikzcd}
\mathbb{N}_0 \times B \arrow[d, "(\!| f |\!)"'] & B + \mathbb{N}_0 \times B \arrow[l, "\alpha"'] \arrow[d, "id + (\!| f |\!)"] \\
C & B + C \arrow[l, "f"]
\end{tikzcd}$$

where (constant functions are denoted by $\underline{k}\ x = k$):

$$\alpha = [\underline{0} \triangledown id, succ \times id] = [\underline{0}, succ \cdot fst] \triangledown [id, snd]$$

Going general ($\mathbb{N}_0$)

Universal property (UP):

$$k = (\!|f|\!) \quad \Leftrightarrow \quad k \cdot \alpha = f \cdot (id + k) \tag{11}$$

Reflexion: $(\!|\alpha|\!) = id$; Projection:

$$
\begin{array}{ll}
& fst \cdot \alpha = [\underline{0}, succ \cdot fst] \\
\Leftrightarrow & \quad \{ \text{ fusion-}+ \ \} \\
& fst \cdot \alpha = [\underline{0}, succ]) \cdot (id + fst) \\
\Leftrightarrow & \quad \{ \text{ universal property } \} \\
& fst = (\!|[\underline{0}, succ]|\!)
\end{array}
$$

$\square$

Complementation $\mathbb{N}_0 \times B \xleftarrow{(\!|[\underline{0},succ]|\!)^{\triangledown}(\!|[id,f]|\!)} \mathbb{N}_0 \times B$ brings "**banana-split**" to mind...

Banana-split

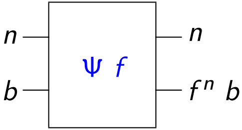
As with standard folds (catamorphisms) the " banana-split" rule states:

$$\llparenthesis f \rrparenthesis \triangledown \llparenthesis g \rrparenthesis = \llparenthesis (f \cdot (id + fst)) \triangledown (g \cdot (id + snd)) \rrparenthesis$$

For **any** $f : B \to B$, let us define

$$\mathbb{N}_0 \times B \xleftarrow{\Psi\ f} \mathbb{N}_0 \times B \ = fst \triangledown \llparenthesis [id, f] \rrparenthesis$$

That is, $\Psi\ f\ (n, b) = (n, f^n\ b)$ is a **for**-loop which keeps its input. We will show that it preserves **injectivity**.



First of all, we calculate $\Psi\ f$ following the standard style. (Next slide.)

Calculating $\Psi\ f$

$$
\begin{array}{cl}
& \Psi\ f \\
= & \{\ \Psi\ f = \mathit{fst} \triangledown (\![[id, f]]\!)\ ;\ \text{reflexion}\ \} \\
& (\![[\underline{0}, \mathit{succ}]]\!) \triangledown (\![[id, f]]\!) \\
= & \{\ \text{banana-split}\ \} \\
& (\![[\underline{0}, \mathit{succ} \cdot \mathit{fst}] \triangledown [id, f \cdot \mathit{snd}]]\!) \\
= & \{\ \text{exchange law}\ \} \\
& (\![[\underline{0} \triangledown id, \mathit{succ} \times f]]\!)
\end{array}
$$

From $\Psi\ f = (\![[\underline{0} \triangledown id, \mathit{succ} \times f]]\!)$ we derive, by the UP:

$$\Psi\ f \cdot (\underline{0} \triangledown id) = \underline{0} \triangledown id$$

$$\Psi\ f \cdot (\mathit{succ} \times id) = (\mathit{succ} \times f) \cdot \Psi\ f$$

Ψ preserves injectivity

First note that $[\underline{0} \triangledown id, succ \times f]$ is **injective** iff f is injective, by the following rule

---

$[R, S]$ **injective** *iff both* R, S *injective and* $R^\circ \cdot S \subseteq \bot$.

(Note that $\underline{0}^\circ \cdot succ \subseteq \bot$ since there is no $n \in \mathbb{N}_0$ such that $succ\ n = 0$.)

Therefore, to show that $\Psi\ f = (\![\underline{0} \triangledown id, succ \times f]\!)$ preserves **injectivity** it is enough to show that $(\!\_\!)$ does so:

$$f \text{ injective} \Rightarrow (\!f\!) \text{ injective} \tag{12}$$

(Proof in the annex.)

Moving to a truly quantum setting

Quoting (Mu et al., 2004):

> *The motivation to study languages for* **reversible** *programs traditionally comes from the* **thermodynamics** *view of computation.*

What about **quantum programming** (QP)?

In **QP** we actually rely on **quantum mechanics** to run our programs. How can this be?

Quantum mechanics (QM) is normally "explained" using **linear algebra**.

Relation algebra and **linear** algebra are tightly related. Moving from the former to the latter is quite smooth. 

"Matrices are arrows"

In the same way we extended **functional** declarations $f : A \to B$ to **relational** ones, $R : A \to B$, we do the same for **matrices**:

$M : A \to B$ declares a matrix with $\#A$-many **columns** *and $\#B$-many* **rows**. *Writing $M : A \to B$ or $M : B \leftarrow A$ is the same.*[5]

In **QM**, matrices are complex-number-valued, for instance that describing the so-called **T-gate**,

$$\mathbf{2} \xleftarrow{T} \mathbf{2} \;=\; \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{i\,\pi}{4}} \end{bmatrix}$$

where $e^{i\,x} = \cos\,x + i\,\sin\,x$ (Euler's formula).

---

[5] Assume A and B finite, for simplicity.

"Matrices are arrows"

(Constant) functions of type $1 \to A$ expand to **column-vectors** of complex numbers, for instance, $\mathbf{2} \xleftarrow{q} 1 = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$.

What about arrow **composition**, recall $f \cdot g$ and $R \cdot S$?

Easy: $M \cdot N$ is **matrix multiplication**:
$$\begin{tikzcd} B & A \arrow[l, "M"'] & C \arrow[l, "N"'] \arrow[ll, bend left, "M \cdot N"] \end{tikzcd}$$

$$b(M \cdot N)c = \langle \sum a :: (b\ M\ a) \times (a\ N\ c) \rangle$$

NB: we denote **matrix cells**, e.g. $b\ M\ a$, as we did for relations. Why a different notation?

Bijections $\rightarrow$ unitary transformation

Our original **relations** and **functions** are accepted, as $\{0,1\}$-valued matrices.

Functions, in particular, are the only $\{0,1\}$-matrices such that $! \cdot f = !$.

But they become "divisible". For instance, you can take *"the sqrt of negation"*, since

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

where

$$\sqrt{\neg} = \frac{1}{2} \begin{bmatrix} 1+i & 1-i \\ 1-i & 1+i \end{bmatrix}$$

Thus one moves into the wonderland of *actual* **quantum logic**, in which **classical logic** operations are no longer primitive.

Bijections $\rightarrow$ unitary transformation

What kind of matrix is $\sqrt{\neg}$?

It is **unitary** — a refined notion of **reversible**:

A matrix $A \xleftarrow{M} A$ *is* **unitary** *iff*

$$M^{\dagger} \cdot M = id = M \cdot M^{\dagger} \tag{13}$$

where $M^{\dagger} = \overline{M}^{\circ}$ *is the* **conjugate transpose** *of* M *and:*

$$\overline{x + y\ i} = x - y\ i$$

$$\overline{\begin{bmatrix} M & N \\ P & Q \end{bmatrix}} = \begin{bmatrix} \overline{M} & \overline{N} \\ \overline{P} & \overline{Q} \end{bmatrix}$$

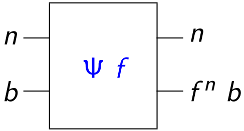
Quantum mechanical processes governed by **unitary** matrices are the building blocks of **QP**.

Reversible $\rightarrow$ Unitary

To what extent does what we did for reversibility apply to **QP**?

The nice story is that our investment in **pointfree** notation pays off now.

Recall, for example,



defined by

$$\Psi\ f \cdot \alpha = [\underline{0} \triangledown id, succ \times f] \cdot (id + \Psi\ f)$$

We just need to extend pairing $(\_ \triangledown \_)$ and junction $[\_, \_]$ to arbitrary matrices.

Linearity

Pairing gives rise to the Khatri-Rao product:

$$(x, y)\ (M \triangledown N)\ a = (x\ M\ a)\ (y\ N\ a)$$

What about $R \cup S$ and $R \cap S$? They become (cell-wise) addition and multiplication, respectively:

$$b\ (M + N)\ a = B\ M\ a + b\ N\ a$$
$$b\ (M \times N)\ a = (B\ M\ a)\ (b\ N\ a)$$

Note that, unlike $R \cup R = R$, $M + M = 2\ M$.

Linearity is the essence it all:

$$Q \cdot (M + N) = Q \cdot M + Q \cdot N$$
$$(M + N) \cdot Q = M \cdot Q + N \cdot Q$$

Tensor product and direct sum

The **Khatri-Rao** product leads the so-called **Kronecker** (or **tensor**) product

$$
\begin{array}{ccc}
A & B & A \times B \\
M \downarrow & N \downarrow & \downarrow M \otimes N \\
C & D & C \times D
\end{array}
$$

by

$$M \otimes N = (M \cdot fst) \triangledown (N \cdot snd)$$

— cf. relational product $R \times S$.

Finally, $[R, S]$ corresponds to $[M|N]$ which collates matrices horizontally, for instance:

$$[id|\neg] = [\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \mid \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}] = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Towards "quantamorphisms"

The following property of relations

$$[R, S] \cdot [P, Q]^{\circ} = R \cdot P^{\circ} \cup S \cdot Q^{\circ}$$

also holds for matrices:

$$[M|N] \cdot [P|Q]^{\circ} = M \cdot P^{\circ} + N \cdot Q^{\circ} \tag{14}$$

Then

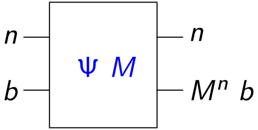
$$
\begin{array}{ll}
& \Psi\ M = [\underline{0} \triangledown id, (succ \otimes M) \cdot \Psi\ M] \cdot \alpha^{\circ} \\
\Leftrightarrow & \qquad \{\ \text{unfold } \alpha\ \} \\
& \Psi\ M = [\underline{0} \triangledown id, (succ \otimes M) \cdot \Psi\ M] \cdot [\underline{0} \triangledown id, succ \otimes id]^{\circ} \\
\Leftrightarrow & \qquad \{\ \ \} \\
& \Psi\ M = (\underline{0} \triangledown id) \cdot (\underline{0} \triangledown id)^{\circ} + (succ \otimes M) \cdot \Psi\ M \cdot (succ^{\circ} \otimes id)
\end{array}
$$

Towards "quantamorphisms"

Thus we obtain a recursive matrix definition whose least fixpoint is

$$\begin{array}{rcl} \Psi\ M & = & \mu X.(B + (succ \otimes M) \cdot X \cdot (succ^{\circ} \otimes id)) \\ & & \textbf{where } B = (\underline{0} \triangledown id) \cdot (\underline{0} \triangledown id)^{\circ} \end{array}$$

is the "quantamorphism"



implementing the quantum **for** gate which iterates M over the second input controlled by the first one (a naural number).

Quantamorphism Ψ M in Matlab / Octave

matlab — vi quanta.m — 54×30

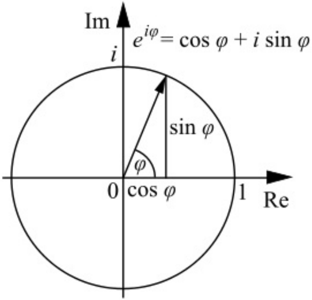
```
function R = quanta(n,M)

%   n * b <---- alpha ------ b + n * b
%     |                      |
%     |                      |
%     X                      id + X
%     |                      |
%     |                      |
%     v                      V
%   n * b <---- [ A B ]----- b + n * b

   [b,a] = size(M);
   if ~(b==a)
       error('M must be square');
   else
       R0=zeros(n*b,n*b); id=eye(b);
       A=kr(const(b,n,1),id);
       alpha=[A kron(succ(n),id)];
       B=kron(succ(n),M);
       C=[A B];
       R = fix(b,R0,C,alpha);
   end
end

function R = fix(b,X,C,alpha)
   id=eye(b);
   Y= C*(oplus(id,X))*alpha';
   if (Y==X) R = X; else R = fix(b,Y,C,alpha); end
end
```

Iterating a phase-shift gate

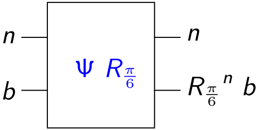
Consider the so-called **phase shift gate** defined by $R_\phi = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\,\phi} \end{bmatrix}$

Recalling $e^{i\,\phi} = \cos\phi + i\,\sin\cdot\phi$, we get, for instance,

$$R_{\frac{\pi}{6}} = \begin{bmatrix} 1 & 0 \\ 0 & 0.867 + 0.5\,i \end{bmatrix}$$



The finite approximation to



for $\#B = 2$ and control $n \leqslant 4$ 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 **complementation** $(fst \triangledown \_)$ shifting the corresponding iteration of gate $R_{\frac{\pi}{6}}$ along the diagonal.

Summary

Quantamorphisms have the advantage over other quantum strategies of dispensing with **measurements**. But the concept is still experimental.

Building upon previous work on **stochastic folds** in LAoP (Murta and Oliveira, 2015).

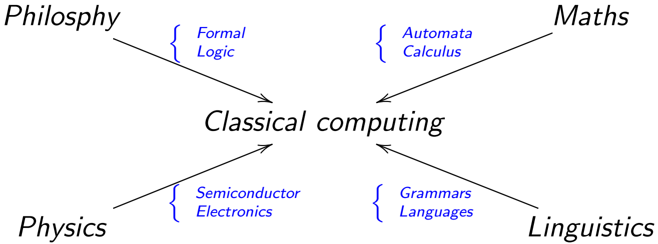
The (linear) algebra of (**unitary**) **quantamorphisms** is the topic of Ana Neri's MSc project (grantee INESC TEC).

Towards **correct by construction** quantum programs.

Categorial approach — investigate Hinze (2013) "Adjoint folds" in the context of monoidal closed categories.

Doomed to repeat history?

Classical computing — Happy blend of diverse bodies of knowledge:



Maths dreamed of it...

1936

Turing (1912-1954) develops in detail an **abstract** *notion of what we now call a* **programmable computer** — *known as the* **Turing machine**.

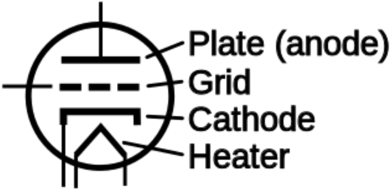
1936

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

A. Turing (1912-1954)

Church-Turing thesis: λ-computable $\Leftrightarrow$ Turing-computable.

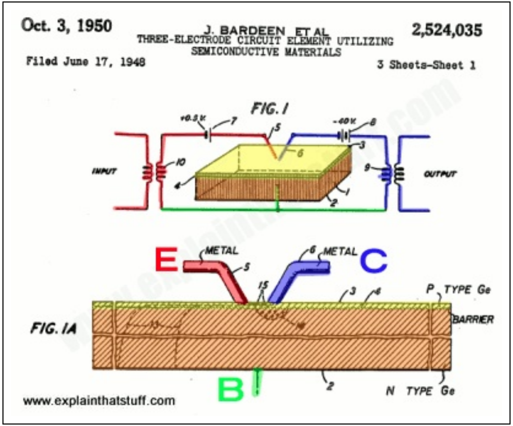
Physics made it happen...



Vacuum tubes, triodes (1912)

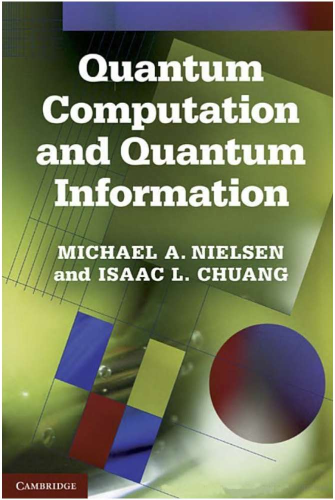
Credits: `https://en.wikipedia.org/wiki/Triode`

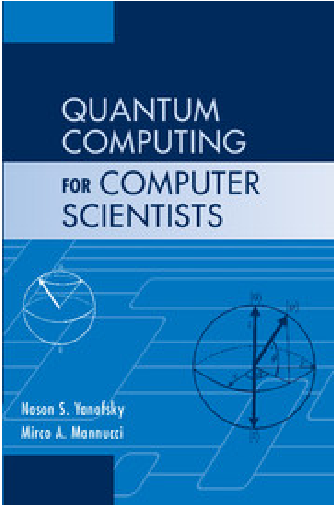
Physics made it happen...



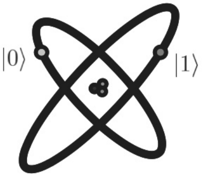
Transistors (1948)

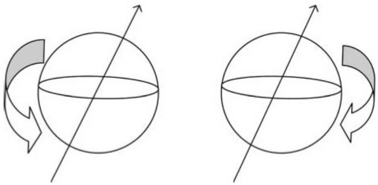
Quantum literature is vast (2000s)





Physics (again) will make it happen...





but this time it sounds far more challenging — particle **spins**, **ion traps**, ...

> *"(...) the implementation of quantum computing machines represents a formidable challenge to the communities of engineers and applied physicists." (Yanofsky and Mannucci, 2008)*

Intuition far less helpful... Thus the need for a **calculational** approach!

Annex — proof of (12)

Let $k = \Psi\ f$. By the UP (11), $k = f \cdot \underbrace{(id + k)}_{\mathbf{F}\ k} \cdot \alpha^\circ$. We calculate

$K = \ker\ k$ assuming $\ker\ f = id$:

$$
\begin{array}{cl}
 & K = k^\circ \cdot k \\
\Leftrightarrow & \quad \{\ \text{unfold } f \cdot \mathbf{F}\ k \cdot \alpha^\circ\ \} \\
 & K = \alpha \cdot \mathbf{F}\ k^\circ \cdot f^\circ \cdot f \cdot \mathbf{F}\ k \cdot \alpha^\circ \\
\Leftrightarrow & \quad \{\ \text{assumption: } f^\circ \cdot f = id\ \} \\
 & K = \alpha \cdot \mathbf{F}\ k^\circ \cdot \mathbf{F}\ k \cdot \alpha^\circ \\
\Leftrightarrow & \quad \{\ \mathbf{F}\ (R \cdot S) = (\mathbf{F}\ R) \cdot (\mathbf{F}\ S) \text{ and } \mathbf{F}\ R^\circ = (\mathbf{F}\ R)^\circ\ \} \\
 & K = \alpha \cdot \mathbf{F}\ k^\circ \cdot k \cdot \alpha^\circ \\
\Leftrightarrow & \quad \{\ K = k^\circ \cdot k;\ \text{UP (for relations)}\ \} \\
 & K = (\!|\alpha|\!) \\
\Leftrightarrow & \quad \{\ \text{Reflexion: } (\!|\alpha|\!) = id\ \} \\
 & K = id
\end{array}
$$

Annex — Free ⦇_⦈-theorem

In the **functional** case:

$$f \cdot (R + S) \ \subseteq \ S \cdot g \Rightarrow (\!| f |\!) \cdot (id \times R) \ \subseteq \ S \cdot (\!| g |\!) \tag{15}$$

recall

$$\begin{tikzcd}
\mathbb{N}_0 \times B \arrow[d, "(\!| f |\!)"'] & & B + \mathbb{N}_0 \times B \arrow[ll, "\alpha"'] \arrow[d, "id + (\!| f |\!)"] \\
C & & B + C \arrow[ll, "f"]
\end{tikzcd}$$

Corollaries (**fusion** laws):

$$(\!| f |\!) \cdot (id \times r) = (\!| f \cdot (r + id) |\!)$$

$$f \cdot (id + s) = s \cdot g \Rightarrow (\!| f |\!) = s \cdot (\!| g |\!)$$

To do: check these properties in the **linear algebra** case.

References

F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM TDS*, 6(4):557–575, December 1981.

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

J.N. Foster, M.B. Greenwald, J.T. Moore, B.C. Pierce, and A. Schmitt. Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. *ACM Trans. Program. Lang. Syst.*, 29(3):17, 2007. ISSN 0164-0925.

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

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

Hsiang-Shang Ko and Zhenjiang Hu. An axiomatic basis for bidirectional programming. *PACMPL*, 2(POPL):41:1–41:29, 2018. doi: 10.1145/3158129. URL http://doi.acm.org/10.1145/3158129.

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.

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

S-C. Mu, Z. Hu, and M. Takeichi. An injective language for reversible computation. In *MPC 2004*, pages 289–313, 2004. doi: 10.1007/978-3-540-27764-4_16.

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

J.N. Oliveira. A relation-algebraic approach to the “Hoare logic” of functional dependencies. *JLAP*, 83(2):249–262, 2014.

N.S. Yanofsky and M.A. Mannucci. *Quantum Computing for Computer Scientists*. Cambridge University Press, 2008. doi: 10.1017/CBO9780511813887.