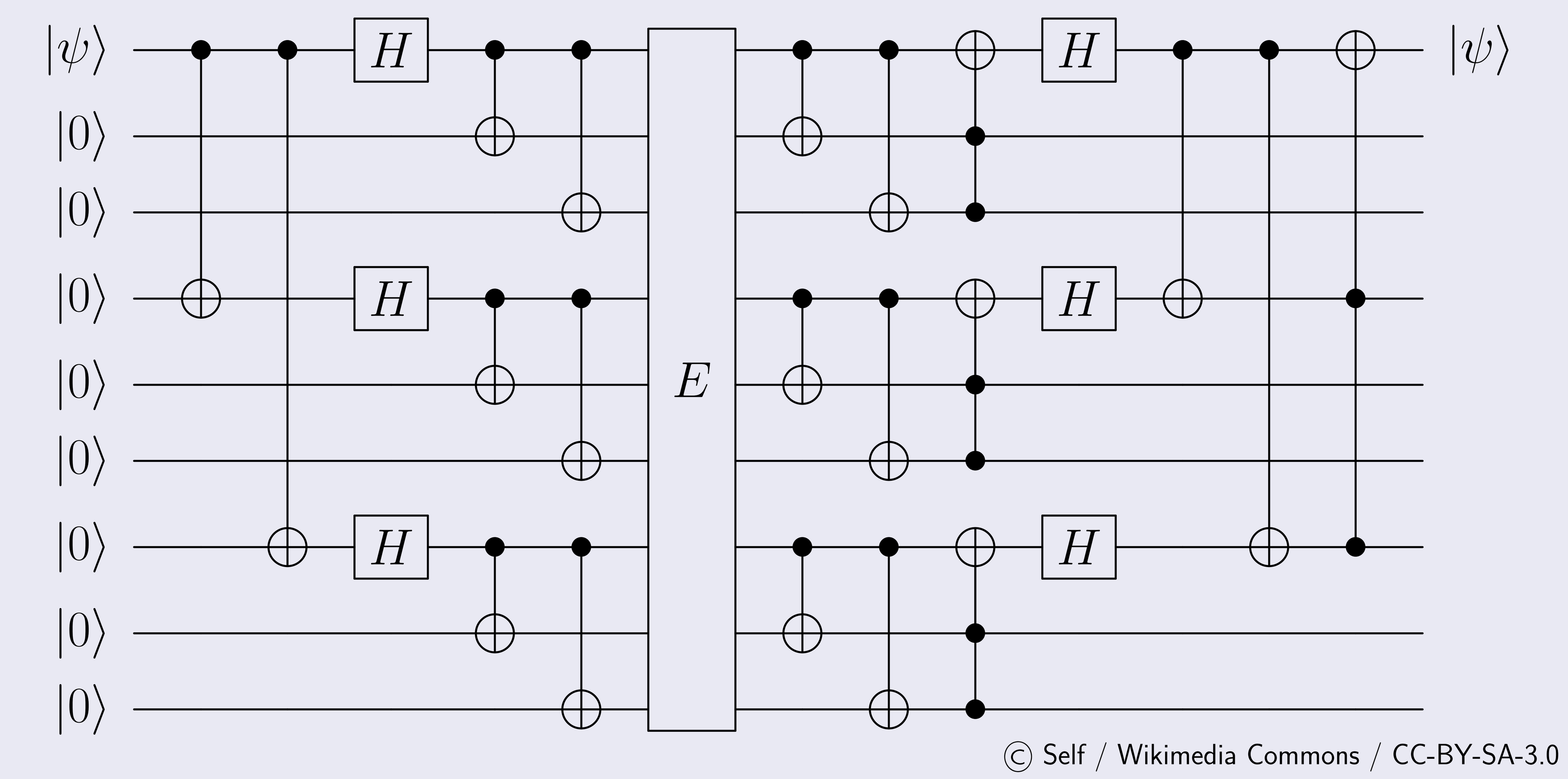
Formalizing quantum circuits with MathComp/Ssreflect

Takafumi Saikawa and Jacques Garrigue Nagoya University

Motivation

We want to formalize quantum circuits

Shor's 9-qubit code (correcting both flips)



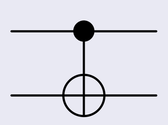
© Self / Wikimedia Commons / CC-BY-SA-3.0

Basic differences: bits and qubits

Classical	Quantum
bit $\in \{0,1\}$	qubit $\in \mathbb{C}^2$
functions in **Set**	unitary transformations in **FdHilb**
direct product:	tensor product
$\mathbf{Set}(X \times Y, Z) \cong \mathbf{Set}(X, Z^Y)$	$\mathbf{FdHilb}(X \otimes Y, Z) \cong \mathbf{FdHilb}(X, Z^Y)$

Problem

- Each gate (= unitary transformation) is fairly simple:



$$= \texttt{CNOT} = \begin{bmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&1\\ 0&0&1&0 \end{bmatrix}$$

- but when put in a circuit, it becomes a monster:

$$\begin{bmatrix} x & y \\ z & w \end{bmatrix} \otimes \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} xa & xb & ya & yb \\ xc & xd & yc & yd \\ za & zb & wa & wb \\ zc & zd & wc & wd \end{bmatrix}$$

State-as-function

Tensor power

Tensor power

- Tensor power $V^{\otimes n}$
  = iterated tensor product $V \otimes \cdots \otimes V$
- If $V = K^m$, $V^{\otimes n} \cong K^{m^n}$

Array of qubits

- Qubit $\in \mathbb{C}^2$
- Array of qubits $\in (\mathbb{C}^2)^{\otimes n}$

Functional view

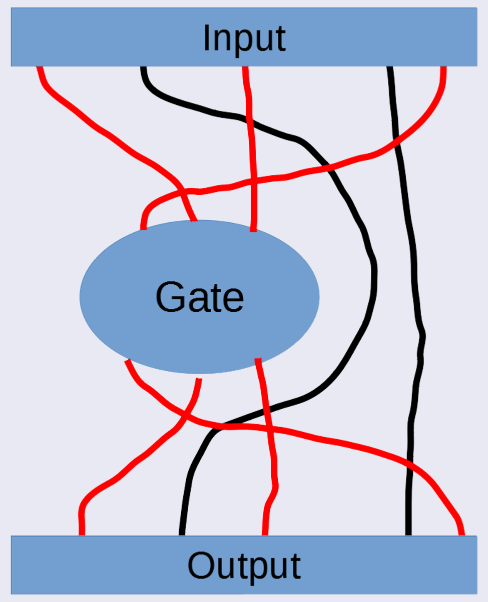
The state of n qubits can be seen as a function from a classical state to $\mathbb{C}$:

$$\texttt{qustate}_n = (\mathbb{C}^2)^{\otimes n} \cong \mathbb{C}^{2^n} \cong (\{0,1\}^n \to \mathbb{C})$$

Gates are unitary transformations on these state functions.

Lens

Lens, curry-uncurry, focus



- Lens = injection between finite ordinals, indicating the choice of wires (red wires in the picture)
- Curry / Uncurry = currying along a given lens which quotients away the unused (black) wires
- Focusing = composing curry, gate and uncurry to build the diagram

`lens`

$$\texttt{lens n m} = (\{1,\ldots,m\} \rightarrowtail \{1,\ldots,n\})$$

`curry` and `uncurry`

For T a vector space and ℓ : `lens n m` ,

$$(T^2)^{\bigotimes n} \cong T^{2^n} \xrightarrow{\texttt{curry}_\ell} \left(T^{2^{n-m}}\right)^{2^m} \cong \left(\left(T^{2^{n-m}}\right)^2\right)^{\bigotimes m}$$

$$\texttt{uncurry}_\ell = \texttt{curry}_\ell^{-1}$$

`focus`

And for G unitary,

$$\texttt{focus}_\ell\ G = \texttt{uncurry}_\ell \circ G \circ \texttt{curry}_\ell$$

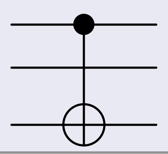
Polymorphic operator

For `focus` to typecheck, the unitary operator G must actually be polymorphic:

$$G : \forall T : \text{vector sp.}, (T^2)^{\bigotimes n} \xrightarrow{\text{unitary}} (T^2)^{\bigotimes n} \quad (= \texttt{endo}\ n)$$

$$\texttt{focus}_\ell\ G = \lambda T.(\texttt{uncurry}_\ell \circ G_{T^{2^{n-m}}} \circ \texttt{curry}_\ell)$$

Example:



$$= \texttt{focus}_{\{1\mapsto 1, 2\mapsto 3\}_3}\ \texttt{CNOT}$$

Parametricity and naturality

Polymorphism is not enough

- We know from its type that G is polymorphically linear / unitary
- But it could be unitary / linear differently for each T
- I.e., the matrix representing the linearity might differ between different `T` s
- And `focus` does change `T`

Parametricity

We want G to be represented by a single matrix:

$$\exists M : \text{matrix}, \ \forall T : \text{vector sp.}, \ \forall v : (T^2)^{\bigotimes n}, \ G_T(v) = Mv.$$

Naturality

We can rephrase parametricity without reference to a matrix, using naturality:

$$\begin{tikzcd} T \arrow[d, "\forall\varphi"'] \\ T' \end{tikzcd} \qquad \begin{tikzcd} T^{\otimes I^k} \arrow[r, "f_T"] \arrow[d, "\varphi^{\otimes I^k}"'] & T^{\otimes I^k} \arrow[d, "\varphi^{\otimes I^k}"] \\ T'^{\otimes I^k} \arrow[r, "f_{T'}"'] & T'^{\otimes I^k} \end{tikzcd}$$

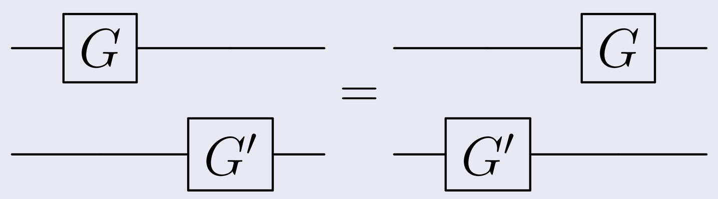
Proving properties

Proof strategy

- Prove basic properties of lenses.
  Some of them are hard.
- Use them to prove properties of tensor powers and endomorphisms.
- Need also lemmas to connect with matrix representations.

Proofs of properties

- Equivalence of parametricity and naturality. (We assume them below.)
- If G is unitary, so is $\texttt{focus}_\ell\ G$.
- For ℓ : `lens n m` and ℓ' : `lens m p`, $\texttt{focus}_{\ell\circ\ell'} = \texttt{focus}_\ell \circ \texttt{focus}_{\ell'}$
- For ℓ : `lens n m` and ℓ' : `lens n p` disjoint,
  $\texttt{focus}_{\ell'}\ G' \circ \texttt{focus}_\ell\ G = \texttt{focus}_\ell\ G \circ \texttt{focus}_{\ell'}\ G'$



- and many more!

Applications

Shor's code

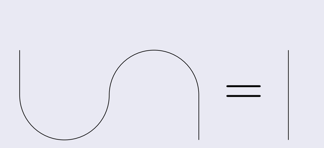
```
Definition bit_flip_enc : endo 3 :=
  tsapp [lens 0; 2] cnot \v  tsapp [lens 0; 1] cnot.
Definition bit_flip_dec : endo 3 :=
  tsapp [lens 1; 2; 0] toffoli \v bit_flip_enc.

Definition sign_flip_dec := bit_flip_dec \v hadamard3.
Definition sign_flip_enc := hadamard3 \v bit_flip_enc.

Definition shor_enc : endo 9 :=
  focus [lens 0; 1; 2] bit_flip_enc \v
  focus [lens 3; 4; 5] bit_flip_enc \v
  focus [lens 6; 7; 8] bit_flip_enc \v
  focus [lens 0; 3; 6] sign_flip_enc.
Definition shor_dec : endo 9 :=
  focus [lens 0; 3; 6] sign_flip_dec \v
  focus [lens 0; 1; 2] bit_flip_dec \v
  focus [lens 3; 4; 5] bit_flip_dec \v
  focus [lens 6; 7; 8] bit_flip_dec.

Definition shor_code (chan : endo 9) :=
  shor_dec \v chan \v shor_enc.
```

Kindergarten Quantum Mechanics (wip)

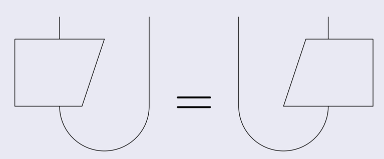
[Coecke & Kissinger 2017] Picturing Quantum Processes.

```
Definition cap : mor n (n-2) := ...
Definition cup : mor (n-2) n := ...
```

`straighten` :



`transpose_cup` :



```
Lemma straighten : cap [lens 1; 2] \v cup [lens 0; 1] =e idmor 1.
Lemma transpose_cup (M : tsquare 1) :
  focus [lens 0] (tsmor M) \v cup (n:=2) [lens 0; 1] =e
  focus [lens 1] (tsmor (transpose_tsquare M)) \v cup [lens 0; 1].
```

Problem: proving properties of focusing on non-endomorphic transformations is much harder.

The code is available at: `https://github.com/t6s/qecc/`