A Type-Theoretic Account of Quantum Computation

Jacques Garrigue Takafumi Saikawa

September 4, 2023

Table of contents

Background: Unitary semantics

Unitary semantics of pure quantum computation

- An isolated qubit is a vector of norm 1 in $\mathbb{C}^2$, with basis $|0\rangle = (1,0)$ and $|1\rangle = (0,1)$
- States composed of n qubits are vectors of norm 1 in the Hilbert space of the n-iterated tensor product

$$(C^2)^{\otimes n} = \mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^n}$$

  with computational basis states $|i_1\rangle \otimes \cdots \otimes |i_n\rangle$
- Other states are linear combinations, in particular those that cannot be expressed as a n-ary tensor are entangled
- Pure operations are unitary transformations (linear and norm preserving) on that space

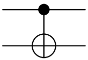
Quantum gates

- Basic operations are unitary transformations called gates
- They can be described by their matrix representation

Hadamard gate

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

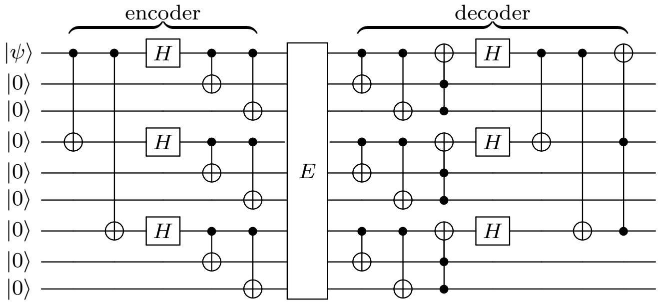
CNOT gate



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

Example of quantum circuit

Shor's 9-qubit code



- A quantum circuit applies unitary transformations to an input state to obtain an output state
- Here $\boxed{E}$ denotes a possibly noisy quantum channel

Semantics of composition

- For pure computations, the whole circuit can also be described by a matrix
- Application of a gate to a large state uses padding, i.e. taking a tensor product with an identity matrix and reordering dimensions.
  For instance the first CNOT gate becomes:

$$U_{2\otimes 9}((42)) \begin{bmatrix} I_{128} & 0 & 0 & 0 \\ 0 & I_{128} & 0 & 0 \\ 0 & 0 & 0 & I_{128} \\ 0 & 0 & I_{128} & 0 \end{bmatrix} U_{2\otimes 9}((24))$$

  where $U_{2\otimes 9}((24))$ is the tensor permutation matrix exchanging 2nd and 4th component of the tensor product

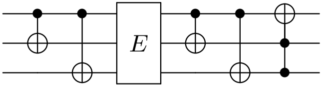
Problems with this semantics

1. The size of matrices becomes huge (here 512×512)
2. The reorderings are particularly cumbersome
3. While these problems can be fixed to some extent by using a symbolic representation of Kronecker products, and/or by adopting the so-called labelled Dirac notation, this comes at a cost in terms of compositionality

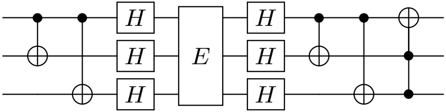
Compositionality

Shor's code is actually based on the following two simpler codes, which are able to fix respectively bit-flips and sign-flips.

Bit-flip code



Sign-flip code



We would like to be able to handle such subcircuits just like gates, but we do not want to be bothered by the permutations.

Direct power vector space and naturality

Quantum states as functions

An alternative view of quantum states

- uses the isomorphism $\mathbb{C}^2 \cong \{0,1\} \to \mathbb{C}$
- The basis states $|0\rangle = (1,0)$ and $|1\rangle = (0,1)$ become

$$\begin{array}{rcl} |0\rangle & = & \lambda x : 2.\mathbf{if}\ x = 0\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \\ |1\rangle & = & \lambda x : 2.\mathbf{if}\ x = 1\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \end{array}$$

  where $2 = \{0,1\}$
- This extends to states composed of n qubits:

$$\begin{array}{rcl} \mathsf{qustate}_n & = & \{0,1\}^n \to \mathbb{C} \\ |i_1, \ldots, i_n\rangle & = & \lambda x : 2^n. \\ & & \quad \mathbf{if}\ x = (i_1, \ldots, i_n)\ \mathbf{then}\ 1\ \mathbf{else}\ 0 \end{array}$$

This looks like probabilistic programming

Generalization : Direct Power

For any vector space T, we define the direct power vector space of functions from the nth power of a finite type (e.g. 2^n) to T.

We use mathematical definitions from MathComp.

direct power vector space

$\texttt{Variables (I : } \text{finite type}\texttt{) (dI : I) (K : } \text{field}\texttt{).}$

$\texttt{Definition dpower n T := I}^{\texttt{n}} \xrightarrow{\text{fin}} \texttt{T.}$

$\texttt{Notation T}^{\widehat{\texttt{n}}} \texttt{ := (dpower n T).}$

$\texttt{Definition dpbasis m (vi : I}^{\texttt{n}}\texttt{) : }(\texttt{K}^{1})^{\widehat{\texttt{m}}} \texttt{ :=}$

$\quad (\texttt{vj} : \texttt{I}^{\texttt{n}}) \overset{\text{fin}}{\longmapsto} \texttt{if vi == vj then 1 else 0.}$

$\texttt{Definition morlin m n := } \forall\, \texttt{T} : \texttt{Vect}_{\texttt{K}}, \texttt{T}^{\widehat{\texttt{m}}} \xrightarrow{\text{lin}} \texttt{T}^{\widehat{\texttt{n}}}\texttt{.}$

This allows to nest quantum states without tensor product.

Generalization : Direct Power

For any vector space T, we define the direct power vector space of functions from the nth power of a finite type (e.g. 2^n) to T.

We use mathematical definitions from MathComp.

direct power vector space

$\texttt{Variables (I : } \text{finite type}\texttt{) (dI : I) (K : } \text{field}\texttt{).}$

$\texttt{Definition dpower n T := } \mathtt{I}^{\mathtt{n}} \xrightarrow{\text{fin}} \mathtt{T}\texttt{.}$

$\texttt{Notation } \mathtt{T}^{\widehat{\mathtt{n}}} \texttt{ := (dpower n T).}$

$\texttt{Definition dpbasis m (vi : } \mathtt{I}^{\mathtt{n}}\texttt{) : } (\mathtt{K}^{1})^{\widehat{\mathtt{m}}} \texttt{ :=}$

$\quad (\mathtt{vj} : \mathtt{I}^{\mathtt{n}}) \xmapsto{\text{fin}} \texttt{if vi == vj then 1 else 0.}$

$\texttt{Definition morlin m n := } \forall\, \mathtt{T} : \mathtt{Vect}_{\mathtt{K}}, \mathtt{T}^{\widehat{\mathtt{m}}} \xrightarrow{\text{lin}} \mathtt{T}^{\widehat{\mathtt{n}}}\texttt{.}$

This allows to nest quantum states without tensor product.

Problem: how do we ensure that functions in `morlin m n` have a unique matrix representation, that does not depend on T ?

Naturality

The solution is to additionally require naturality.

naturality

$$
\begin{tikzcd}
T_1 \arrow[d, "\varphi"'] & T_1^{I^m} \arrow[r, "G_{T_1}"] \arrow[d, "\varphi^{I^m}"'] & T_1^{I^m} \arrow[d, "\varphi^{I^m}"] \\
T_2 & T_2^{I^m} \arrow[r, "G_{T_2}"'] & T_2^{I^m}
\end{tikzcd}
$$

$\mathtt{Definition}\ \mathtt{dpmap}\ \mathtt{m}\ \mathtt{T_1}\ \mathtt{T_2}\ (\varphi : \mathtt{T_1} \to \mathtt{T_2})\ (\mathtt{st} : \mathtt{T_1^{\widehat{m}}}) : \mathtt{T_2^{\widehat{m}}}\ :=$

$\quad (\mathtt{v} : \mathtt{I^m}) \xmapsto{\text{fin}} \varphi\ (\mathtt{st}(\mathtt{v})).$ ((dpmap φ) is denoted φ^{I^m} above)

$\mathtt{Definition}\ \mathtt{naturality}\ \mathtt{m}\ \mathtt{n}\ (\mathtt{G} : \mathtt{morlin}\ \mathtt{m}\ \mathtt{n})\ :=$

$\quad \forall (\mathtt{T_1}\ \mathtt{T_2} : \mathtt{Vect_K}),\ \forall (\varphi : \mathtt{T_1} \xrightarrow{\text{lin}} \mathtt{T_2}),$

$\quad\quad (\mathtt{dpmap}\ \varphi) \circ (\mathtt{G}\ \mathtt{T_1}) = (\mathtt{G}\ \mathtt{T_2}) \circ (\mathtt{dpmap}\ \varphi).$

Linearity, naturality, unitarity

(* *natural morphisms* *)
Record mor m n := $\{\varphi : \texttt{morlin m n} \mid \texttt{naturality } \varphi\}$.
Notation endo n := (mor n n).

(* *from matrix* *)
Definition tsmor n m : $(\mathrm{K}^{\widehat{m}})^{\widehat{n}} \to$ mor m n.

(* *vertical composition* *)

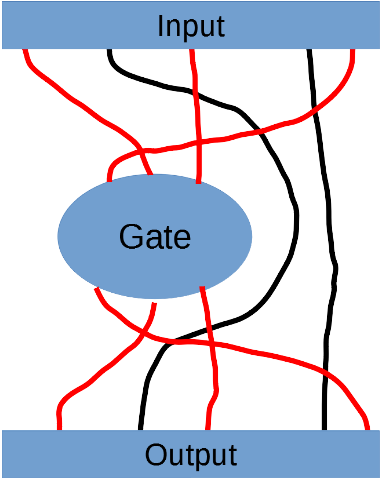
```
Definition comp_mor : mor m p -> mor n m -> mor n p.
Notation "F \v G" := (comp_mor F G).
```

(* *unitarity* *)
Definition unitary_endo m n (F : mor m n) :=
  $\forall$ s t, tinner (F K^1 s) (F K^1 t) = tinner s t.

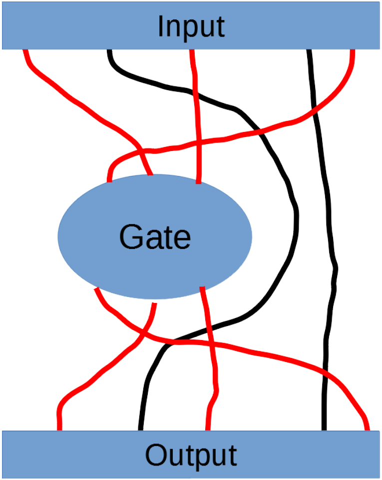
(K^1 is the field K seen as a vector space)

Lens, curry-uncurry, focus

Lens, curry-uncurry, focus

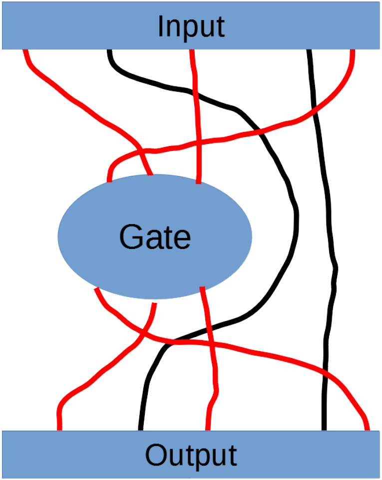


Lens, curry-uncurry, focus



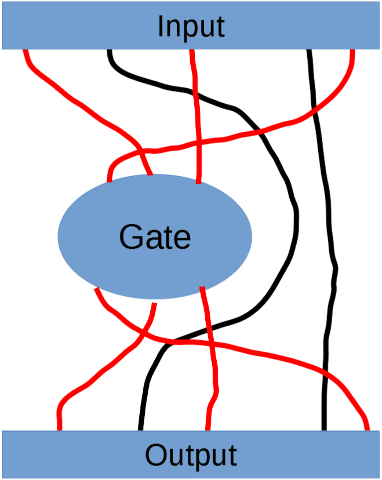
- Lens = choice of wires to be connected to gates; basic combinatorial data

Lens, curry-uncurry, focus



- Lens = choice of wires to be connected to gates; basic combinatorial data
- Currying = quotienting unused wires away

Lens, curry-uncurry, focus



- Lens = choice of wires to be connected to gates; basic combinatorial data
- Currying = quotienting unused wires away
- Focusing = composing curry, gate and uncurry to build the diagram

Lens

lens

$$\texttt{lens n m} : \{1, \ldots, m\} \xrightarrow{\text{injective}} \{1, \ldots, n\}$$

- Provides both inclusion and permutation
- Basic operations:

```
Variables (n m p : nat) (I : Type).
Definition extract : lens n m → I^n → I^m.
Definition merge : lens n m → I^m → I^(n−m) → I^n.
Definition lensC : lens n m → lens n (n - m).
Definition lens_comp :
           lens n m -> lens m p -> lens n p.
```

Currying

curry and uncurry

$$\texttt{curry} : T^{I^n} \overset{\cong}{\longleftrightarrow} \left(T^{I^{n-m}}\right)^{I^m} : \texttt{uncurry}$$

$$\left(T^{I^n} = \mathcal{S}\mathrm{et}(I^n, T) \cong \mathcal{S}\mathrm{et}(I^m, \mathcal{S}\mathrm{et}(I^{n-m}, T))\right)$$

$$\texttt{Variables (n m : nat) } (\ell \texttt{ : lens n m) (T : Vect}_K\texttt{).}$$

$$\texttt{Definition curry (st} : \texttt{T}^{\widehat{\texttt{n}}}) : \left(\texttt{T}^{\widehat{\texttt{n}-\texttt{m}}}\right)^{\widehat{\texttt{m}}} \texttt{ :=}$$

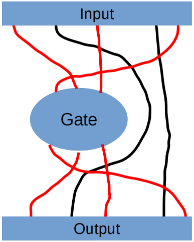
$$(\texttt{v} : \texttt{I}^{\texttt{m}}) \overset{\text{fin}}{\longmapsto} \left((\texttt{w} : \texttt{I}^{\texttt{n}-\texttt{m}}) \overset{\text{fin}}{\longmapsto} \texttt{st (merge } \ell \texttt{ v w)}\right).$$

$$\texttt{Definition uncurry (st} : \left(\texttt{T}^{\widehat{\texttt{n}-\texttt{m}}}\right)^{\widehat{\texttt{m}}}) : \texttt{T}^{\widehat{\texttt{n}}} \texttt{ :=}$$

$$(\texttt{v} : \texttt{I}^{\texttt{n}}) \overset{\text{fin}}{\longmapsto} \texttt{st (extract } \ell \texttt{ v) (extract (lensC } \ell \texttt{) v).}$$

focus

focus




```
Variables (n m : nat) (ℓ : lens n m).
Definition focuslin (G : endo m) : morlin n n :=
  λT. uncurry_{ℓ,T} ∘ G_{\widehat{T^{n-m}}} ∘ curry_{ℓ,T}.
Definition focus (G : endo m) : endo n.
```

Properties

```
(* Distributivity wrt vertical composition *)
Lemma focus_comp n m (f g : endo m) (l : lens n m) :
  focus l (f \v g) = focus l g \v focus l g.

(* Composition of lenses *)
Lemma focusM n m p
  (l : lens n m) (l' : lens m p) (f : endo p) :
  focus (lens_comp l l') f = focus l (focus l' f).

(* Composition of disjoint lenses commutes *)
Lemma focusC n m p (l : lens n m) (l' : lens n p)
  (f : endo m) (g : endo n) : [disjoint l & l'] ->
  focus l f \v focus l' g = focus l' g \v focus l f.

(* Unitarity *)
Lemma focusU n m (l : lens n m) (f : endo m) :
  unitary_endo f -> unitary_endo (focus l f).
```

Proving circuits correct

Representing Shor's code

```
Notation tsapp l M := (focus l (tsmor M)).

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 hadamard3 : endo 3 :=
  tsapp [lens 2] hadamard \v tsapp [lens 1] hadamard
  \v tsapp [lens 0] hadamard.
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 := ...
```

Proving Shor's code

We have only proved the correctness for error-free channels.

```
Definition flip (i : 'I_2) := rev_ord i. (* exchanges 0 and 1 *)
Lemma tsmor_cnot0 i : tsmor cnot Co ¦0, i⟩ = ¦0, i⟩.
Lemma tsmor_cnot1 i : tsmor cnot Co ¦1, i⟩ = ¦1, flip i⟩.
Lemma tsmor_toffoli00 i : tsmor toffoli Co ¦0,0,i⟩ = ¦0,0,i⟩.
Lemma hadamardK T : involutive (tsmor hadamard T).

Lemma bit_flip_enc0 j k : bit_flip_enc Co ¦0,j,k⟩ = ¦0,j,k⟩.
Lemma bit_flip_enc1 j k :
  bit_flip_enc Co ¦1,j,k⟩ = ¦1, flip j, flip k⟩.

Lemma bit_flip_toffoli :
  (bit_flip_dec \v bit_flip_enc) = tsapp [lens 1; 2; 0] toffoli.
Lemma sign_flip_toffoli :
  (sign_flip_dec \v sign_flip_enc) = tsapp [lens 1; 2; 0] toffoli.

Theorem shor_code_id i :
 (shor_dec \v shor_enc) Co ¦i,0,0,0,0,0,0,0,0⟩ = ¦i,0,0,0,0,0,0,0,0⟩.
```

The above lemmas require about 80 lines of proof in total.

Focusing in and out

We provide a number of functions and lemmas that allow to change the view of the current state.

```
Variables (n m : nat) (l : lens n m).
Definition dpmerge : (K^1)^n̂ → (K^1)^m̂ -lin-> (K^1)^n̂.

Lemma focus_dpbasis (f : endo n) (vi : I^n) :
  focus l f _ (dpbasis vi) =
  dpmerge vi (f _ (dpbasis (extract l vi))).

Lemma dpmerge_dpbasis (vi : I^n) (vj : I^m) :
  dpmerge vi (dpbasis vj) =
  dpbasis (merge l vj (extract (lensC l) vi)).

Lemma decompose_scaler k (st : (K^1)^k̂) :
  st = ∑_{t:I^k} st t *: dpbasis t.
```

Proof of `bit_flip_enc1` (first half)

```
   bit_flip_enc Co ¦1,j,k⟩
rewrite /=.
= tsapp [lens 0; 2] cnot Co (tsapp [lens 0; 1] cnot Co ¦ 1, j, k ⟩)
rewrite focus_dpbasis.
= tsapp [lens 0; 2] cnot Co
    (dpmerge [lens 0; 1] [tuple 1; j; k]
       (tsmor cnot Co
          (dpbasis (extract [lens 0; 1] [tuple 1; j; k]))))
simpl_extract.
= tsapp [lens 0; 2] cnot Co
    (dpmerge C [lens 0; 1] [tuple 1; j; k] (tsmor cnot Co ¦ 1, j ⟩))
rewrite tsmor_cnot1.
= tsapp [lens 0; 2] cnot Co
       (dpmerge C [lens 0; 1] [tuple 1; j; k] ¦ 1, flip j ⟩)
rewrite dpmerge_dpbasis.
= tsapp [lens 0; 2] cnot Co
    (dpbasis (merge [lens 0; 1] [tuple 1; flip j]
                    (extract (lensC [lens 0; 1]) [tuple 1; j; k])))
rewrite (_ : merge _ _ _ = [tuple 1; flip j; k]); last by eq_lens.
= tsapp [lens 0; 2] cnot Co ¦ 1, flip j, k ⟩
```

Conclusion

Conclusion

We have provided an alternative account of pure quantum computation, based on

- quantum state seen as function
- currying of this function for focusing
- parametric polymorphic definition of transformations
- characterizing parametricity by naturality

This approach allowed us to prove a number of pure circuits

- Shor's 9-qubit code (on error-free channel)
- GHZ state preparation
- reverse circuit

The ability to manipulate state through currying really seems to simplify proofs!