Quantum Software Lab

THE UNIVERSITY of EDINBURGH

Quantum Circuits are Just a Phase

Alex Rice, University of Edinburgh

IQPS Lecture - 12th March 2026

Overview

- Quantum circuit model of quantum computation
- A quantum “if let”.
- The quantum phase language.

Overview

- Quantum circuit model of quantum computation
- A quantum “if let”.
- The quantum phase language.

```
if let |1⟩ = a {
  if let |−⟩ = b {
    Ph(π)
  }
}
```

Qubits and Unitaries

Classical Bits	Quantum Qubits
$\{\mathrm{false}, \mathrm{true}\}$	unit vectors in $\mathbb{C}^2$
false	$\lvert 0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
true	$\lvert 1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
N/A	$\lvert +\rangle = \frac{1}{\sqrt{2}}(\lvert 0\rangle + \lvert 1\rangle)$
	$\lvert -\rangle = \frac{1}{\sqrt{2}}(\lvert 0\rangle - \lvert 1\rangle)$

Qubits and Unitaries

Classical Bits	Quantum Qubits
$\{\mathrm{false}, \mathrm{true}\}$	unit vectors in $\mathbb{C}^2$
false	$\lvert 0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
true	$\lvert 1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$
N/A	$\lvert +\rangle = \frac{1}{\sqrt{2}}(\lvert 0\rangle + \lvert 1\rangle)$
	$\lvert -\rangle = \frac{1}{\sqrt{2}}(\lvert 0\rangle - \lvert 1\rangle)$

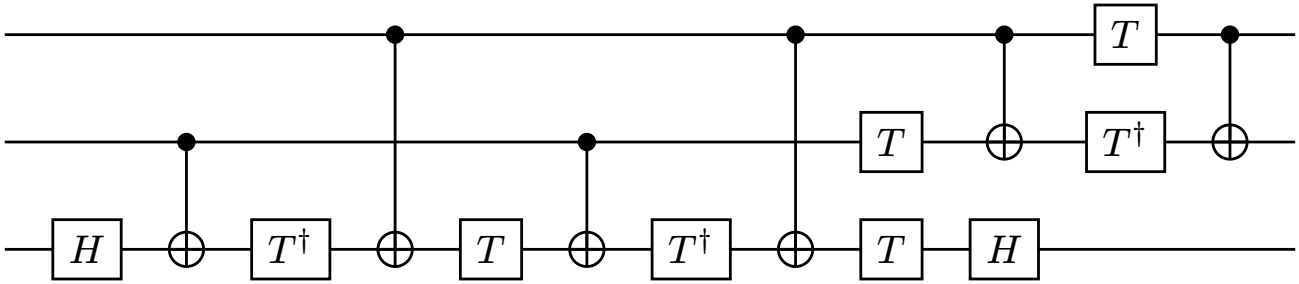
(Measurement-free) quantum computations correspond to *unitary maps*.

$$U \text{ is unitary} \Leftrightarrow UU^\dagger = U^\dagger U = I$$

What programming constructions can we use for unitary maps?

Quantum circuits

Quantum computations are often graphically represented as circuits.



- Wires represent qubits
- Each symbol is a primitive *gate*
- Gates are composed in sequence and parallel

Quantum gates

Each gate is a “black boxed” unitary.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$$

$$\mathrm{CX} = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Quantum gates

Each gate is a “black boxed” unitary.

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix} \quad T = \begin{pmatrix} 1 & 0 \\ 0 & e^{\frac{i\pi}{4}} \end{pmatrix}$$

$$\mathrm{CX} = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Claim: Quantum gates sit at an awkward level of abstraction

Just a phase

Perhaps the simplest unitary map takes the following form:

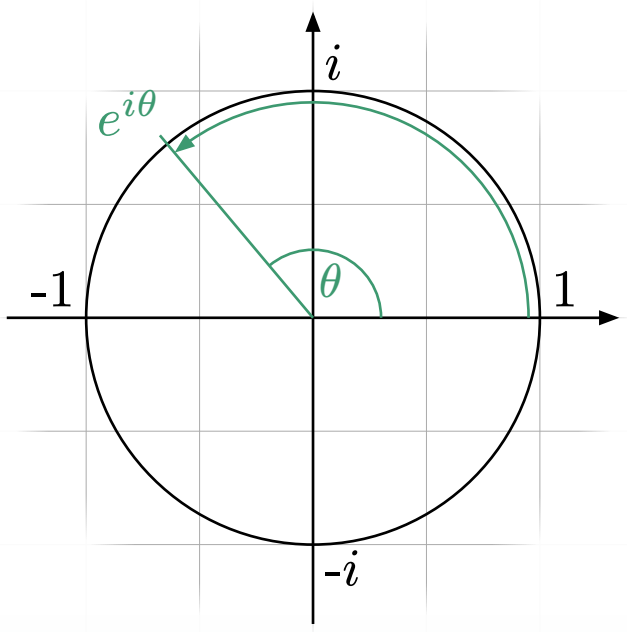
$$v \mapsto e^{i\theta} v$$

We call such a map a *phase* and represent it by the term:

```
Ph(θ)
```

Our language consists of just:

- this phase operator
- a quantum pattern matching construction



Pattern matching for unitary maps.

Maps from classical data can be specified by pattern matching:

```
match x {
  false => { /* Do something */ }
  true => { /* Do something else */ }
}
```

Pattern matching for unitary maps.

Maps from classical data can be specified by pattern matching:

```
match x {
  false => { /* Do something */ }
  true => { /* Do something else */ }
}
```

Linear maps only need to be specified on a basis.

```
match x {
  |0⟩ => { /* Do something */ }
  |1⟩ => { /* Do something else */ }
}
```

Z gate

We can already define the Z gate. Recall:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad Z|0\rangle = |0\rangle \quad Z|1\rangle = -|1\rangle$$

Z gate

We can already define the Z gate. Recall:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad Z|0\rangle = |0\rangle \quad Z|1\rangle = -|1\rangle$$

```
match q {
  |0⟩ => { /* Do something */ }
  |1⟩ => { /* Do something else */ }
}
```

Z gate

We can already define the Z gate. Recall:

$$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \quad Z|0\rangle = |0\rangle \quad Z|1\rangle = -|1\rangle$$

```
match q {
  |0⟩ => { }
  |1⟩ => { Ph(π) }
}
```

“if let” construction

To simplify the syntax, we borrow Rust’s “if let” expression.

```
match q {
  |0⟩ => { }
  |1⟩ => { /* Do something */ }
}
```

```
if let |1⟩ = q {
  /* Do something */
}
```

“if let” construction

To simplify the syntax, we borrow Rust’s “if let” expression.

```
match q {
  |0⟩ => { }
  |1⟩ => { /* Do something */ }
}
```

```
if let |1⟩ = q {
  /* Do something */
}
```

```
Z(q) = if let |1⟩ = q { Ph(π) }
S(q) = if let |1⟩ = q { Ph(0.5π) }
T(q) = if let |1⟩ = q { Ph(0.25π) }
```

X as an “if let”

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle$$

X as an “if let”

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle$$

$$X|+\rangle = X\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{\sqrt{2}}(X|0\rangle + X|1\rangle) = \frac{1}{\sqrt{2}}(|1\rangle + |0\rangle) = |+\rangle$$

$$X|-\rangle = X\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = \frac{1}{\sqrt{2}}(X|0\rangle - X|1\rangle) = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|-\rangle$$

X as an “if let”

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad X|0\rangle = |1\rangle \quad X|1\rangle = |0\rangle$$

$$X|+\rangle = X\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{\sqrt{2}}(X|0\rangle + X|1\rangle) = \frac{1}{\sqrt{2}}(|1\rangle + |0\rangle) = |+\rangle$$

$$X|-\rangle = X\left(\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\right) = \frac{1}{\sqrt{2}}(X|0\rangle - X|1\rangle) = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|-\rangle$$

```
X(q) = if let |−⟩ = q { Ph(π) }
```

Applying a unitary to a pattern

Consider $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$.

Applying a unitary to a pattern

Consider $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$. Letting $|L\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ and $|R\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$:

$$Y|L\rangle = |L\rangle \qquad Y|R\rangle = -|R\rangle$$

We could add $|L\rangle$ and $|R\rangle$ as patterns.

Applying a unitary to a pattern

Consider $Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$. Letting $|L\rangle = \frac{1}{\sqrt{2}}(|0\rangle + i|1\rangle)$ and $|R\rangle = \frac{1}{\sqrt{2}}(|0\rangle - i|1\rangle)$:

$$Y|L\rangle = |L\rangle \qquad Y|R\rangle = -|R\rangle$$

We could add $|L\rangle$ and $|R\rangle$ as patterns. But:

$$|L\rangle = S|+\rangle \qquad |R\rangle = S|-\rangle$$

So we instead add the pattern `s . P` for unitary `s` and pattern `P`.

```
Y(q) = if let S . |−⟩ = q { Ph(π) }
```

Quantum phase language

These two constructions form our universal quantum phase language.

Terms represent unitary maps:

```
s, t ::= Ph(θ) | s ; t | s ⊗ t | if let P = q { s }
```

Quantum phase language

These two constructions form our universal quantum phase language.

Terms represent unitary maps:

```
s, t ::= Ph(θ) | s ; t | s ⊗ t | if let P = q { s }
```

Patterns represent isometries, allowing subspace selection:

```
P, Q ::= |0⟩ | |1⟩ | |+⟩ | |−⟩ | s . P | P ⊗ Q | q
```

Quantum phase language

These two constructions form our universal quantum phase language.

Terms represent unitary maps:

```
s, t ::= Ph(θ) | s ; t | s ⊗ t | if let P = q { s }
```

Patterns represent isometries, allowing subspace selection:

```
P, Q ::= |0⟩ | |1⟩ | |+⟩ | |−⟩ | s . P | P ⊗ Q | q
```

From just these we can derive a universal gate set.

```
if let |1⟩ = a {
  if let |−⟩ = b {
    Ph(π)
  }
}
```

```
if let |1⟩ = a {
  if let |−⟩ = b {
    Ph(π)
  }
}
```

$$\mathrm{CX} = \quad = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Metaoperations

Our language admits two interesting metaoperations:

Inverses

Can be defined inductively:

- $(s \ ; \ t)^\dagger = t^\dagger \ ; \ s^\dagger$
- $(s \otimes t)^\dagger = s^\dagger \otimes t^\dagger$
- $\mathtt{Ph}(\theta)^\dagger = \mathtt{Ph}(-\theta)$
- $(\mathtt{if\ let\ P = q\ \{\ e\ \}})^\dagger = \mathtt{if\ let\ P = q\ \{\ } e^\dagger \mathtt{\ \}}$

Exponentiation

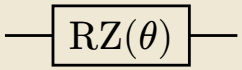
“Composition-free” terms can be exponentiated:

- $(s \otimes t)^x = s^x \otimes t^x$
- $\mathtt{Ph}(\theta)^x = \mathtt{Ph}(x*\theta)$
- $(\mathtt{if\ let\ P = q\ \{\ e\ \}})^x = \mathtt{if\ let\ P = q\ \{\ } e^x \mathtt{\ \}}$

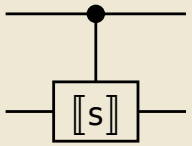
Circuit compilation

A term s can be reduced down to a circuit ⟦s⟧.

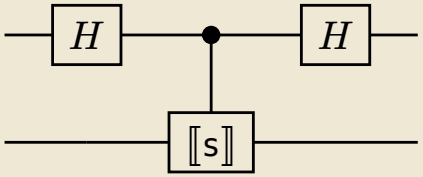
⟦if let |1⟩ = q { Ph(θ) }⟧



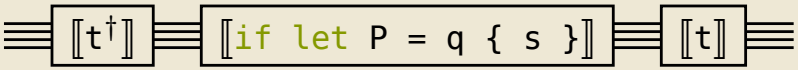
⟦if let |1⟩ = q { s }⟧



⟦if let |−⟩ = q { s }⟧



⟦if let t . P = q { s }⟧



Compiling the Y gate

$$
\begin{aligned}
&[\![\texttt{Y(q)}]\!] \\
=\ &[\![\texttt{if let S . |−⟩ = q \{ Ph(π) \}}]\!] \\
=\ &\text{—}\boxed{[\![\texttt{S}^\dagger]\!]}\text{—}\boxed{[\![\texttt{if let |−⟩ = q \{ Ph(π) \}}]\!]}\text{—}\boxed{[\![\texttt{S}]\!]}\text{—} \\
=\ &\text{—}\boxed{[\![\texttt{S}^\dagger]\!]}\text{—}\boxed{H}\text{—}\boxed{[\![\texttt{if let |1⟩ = q \{ Ph(π) \}}]\!]}\text{—}\boxed{H}\text{—}\boxed{[\![\texttt{S}]\!]}\text{—} \\
=\ &\text{—}\boxed{S^\dagger}\text{—}\boxed{H}\text{—}\boxed{Z}\text{—}\boxed{H}\text{—}\boxed{S}\text{—}
\end{aligned}
$$

Compiling the Y gate

$$\llbracket \texttt{Y(q)} \rrbracket$$

$$= \llbracket \texttt{if let S . |−⟩ = q \{ Ph(π) \}} \rrbracket$$

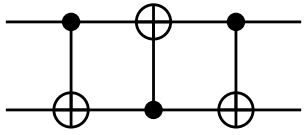
$$= \quad — \boxed{\llbracket \texttt{S}^\dagger \rrbracket} — \boxed{\llbracket \texttt{if let |−⟩ = q \{ Ph(π) \}} \rrbracket} — \boxed{\llbracket \texttt{S} \rrbracket} —$$

$$= \quad — \boxed{\llbracket \texttt{S}^\dagger \rrbracket} — \boxed{H} — \boxed{\llbracket \texttt{if let |1⟩ = q \{ Ph(π) \}} \rrbracket} — \boxed{H} — \boxed{\llbracket \texttt{S} \rrbracket} —$$

$$= \quad — \boxed{S^\dagger} — \boxed{H} — \boxed{Z} — \boxed{H} — \boxed{S} —$$

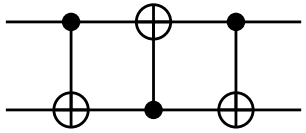
$$\approx \quad — \boxed{S^\dagger} — \boxed{X} — \boxed{S} —$$

The swap gate

$\mathrm{Swap} =$ 

Can we decompile this to a term?

The swap gate

$\mathrm{Swap} =$ 

Can we decompile this to a term?

```
  Swap(q1, q2)
= CX(q1 ⊗ q2) ; CX(q2 ⊗ q1) ; CX(q1 ⊗ q2)
= CX(q1 ⊗ q2) ; if let |1⟩ ⊗ |−⟩ = q2 ⊗ q1 { Ph(π) } ; CX(q1 ⊗ q2)
= CX(q1 ⊗ q2) ; if let |−⟩ ⊗ |1⟩ = q1 ⊗ q2 { Ph(π) } ; CX(q1 ⊗ q2)
= if let CX . (|−⟩ ⊗ |1⟩) = q1 ⊗ q2 { Ph(π) }
= if let "|01⟩ - |10⟩" = q1 ⊗ q2 { Ph(π) }
```

Summary

In the paper

- Defined a type system.
- Gave a categorical semantics.
- Defined an evaluation to circuits.
- Proved semantic equalities on terms.
- Defined well-known algorithms.

In the future

- Measurement.
- Equational theory.
- Operational semantics.
- Optimisations.

Implementation (https://github.com/alexarice/phase-rs)

- Parsing
- Typechecking
- Compiling to circuits
- Evaluating to unitary matrix