Quantum Information Processing
Quantum Algorithms (II)

Richard Cleve
Institute for Quantum Computing & Cheriton School of Computer Science
University of Waterloo

October 9, 2021

Abstract

The goal of these notes is to explain the basics of quantum information processing, with intuition and technical definitions, in a manner that is accessible to anyone with a solid understanding of linear algebra and probability theory.

These are lecture notes for the second part of a course entitled "Quantum Information Processing" (with numberings QIC 710, CS 768, PHYS 767, CO 681, AM 871, PM 871 at the University of Waterloo). The other parts of the course are: a primer for beginners, quantum information theory, and quantum cryptography. The course web site http://cleve.iqc.uwaterloo.ca/qic710 contains other course materials, including video lectures.

I welcome feedback about errors or any other comments. This can be sent to cleve@uwaterloo.ca (with "Lecture notes" in subject heading, if at all possible).

Contents

1 Definition of the discrete log problem

The quantum "algorithms" that we've seen up until now are not algorithms in usual sense. They are in the black-box model, where one is given an unknown function and the goal is to extract information about the function with as few queries to the function as possible. Often, the unknown function is promised to have a special kind of structure (such as the function arising in Simon's problem).

In the next section I will describe a remarkable algorithm for solving the *discrete log problem (DLP)*. This is not a black box problem! It is a conventional computational problem where the input is given as a binary string and the output is also a binary string. No classical polynomial algorithm is known for this problem. In fact, the presumed hardness of this problem has been the basis of cryptosystems (such as the Diffie-Hellman key exchange protocol). In 1994, Peter Shor discovered a polynomial quantum algorithm for this problem, and it's based on the underlying methodologies used in Simon's algorithm—which may sound surprising, since Simon's problem is a black-box problem.

In this section, I define the discrete log problem. In the next section I will describe Shor's polynomial quantum algorithm for this problem. In order to define the discrete log problem, we first need to be familiar with two sets, called $\mathbb{Z}_m$ and $\mathbb{Z}_m^*$.

1.1 Definitions of $\mathbb{Z}_m$ and $\mathbb{Z}_m^*$

Definition 1.1 (of the ring $\mathbb{Z}_m$)**.** *For any positive integer $m \geq 2$, define $\mathbb{Z}_m$ as the set $\{0, 1, \ldots, m-1\}$, equipped with addition and multiplication modulo m.*

A more familiar example of a ring is the set of all integers $\mathbb{Z}$ equipped with "ordinary" addition and multiplication. Note that there are only two elements of $\mathbb{Z}$ that have multiplicative inverses (namely $+1$ and -1). What are the elements of $\mathbb{Z}_m$ that have multiplicative inverses? It depends on m. Let's look at the case where $m = 9$. The elements of $\mathbb{Z}_9$ that have multiplicative inverses are 1, 2, 4, 5, 7, and 8 (0, 3, and 6 do not have multiplicative inverses).

Definition 1.2 (of the group $\mathbb{Z}_m^*$)**.** *For any positive integer $m \geq 2$, the set $\mathbb{Z}_m^*$ consists of all elements of $\mathbb{Z}_m$ that have multiplicative inverses. $\mathbb{Z}_m^*$ is a group.*

For the purposes of DLP, we need only consider $\mathbb{Z}_p^*$ for prime p. In that case, it's known that every non-zero element of $\mathbb{Z}_p^*$ has a multiplicative inverse. Therefore we can use this simple definition of the group $\mathbb{Z}_p^*$ for case where p is prime.

Definition 1.3 (of the group $\mathbb{Z}_p^*$)**.** *For any prime p, define $\mathbb{Z}_p^*$ as the group with elements $\{1, \ldots, p-1\}$, where the operation is multiplication modulo p.*

1.2 Generators of $\mathbb{Z}_p^*$ and the exponential/log functions

An element g of the group $\mathbb{Z}_p^*$ is called a *generator* of the group if the set of all powers g is the group. Whenever a group has such an element, it is called *cyclic*.

Let's consider the case where $p = 7$ as an example. $\mathbb{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$. Obviously 1 is not a generator, since all powers of 1 are just 1. What about 2? 2 is not a generator either, because of we list the powers of 2, which are 2^0, 2^1, 2^2, and so on, we get the sequence $1, 2, 4, 1, 2, 4, \ldots$ so we only get the set $\{1, 2, 4\}$, which is a proper subgroup of $\mathbb{Z}_7^*$. What about 3? 3 *is* a generator, since the powers of 3 are $1, 3, 2, 6, 4, 5, 1, 3, 2, \ldots$, which is the entire set $\mathbb{Z}_7^*$. It turns out that $\mathbb{Z}_p^*$ is cyclic for any prime p.

Theorem 1.1. *For any prime p, there exists $g \in \mathbb{Z}_p^*$ such that*

$$\mathbb{Z}_p^* = \left\{ g^k : k \in \{0, 1, \ldots, p-2\} \right\}. \tag{1}$$

Notice that the exponents run from 0 to $p-2$. This makes sense because the size of $\mathbb{Z}_p^*$ is $p-1$ (it's not p because $0 \notin \mathbb{Z}_p^*$). So the set of exponents of a generator is $\mathbb{Z}_{p-1}$ (it's not $\mathbb{Z}_p$). And, if the elements of $\mathbb{Z}_p^*$ are expressed as powers of a generator g, then $g^x g^y = g^{x+y \bmod p-1}$ holds for any $x, y \in \mathbb{Z}_{p-1}$. In other words, multiplication in $\mathbb{Z}_p^*$ corresponds to addition mod $p-1$ in the exponents of g. Formally, there is an isomorphism between the additive group $\mathbb{Z}_{p-1}$ and the multiplicative group $\mathbb{Z}_p^*$.

Definition 1.4 (discrete exp function)**.** *Relative to a prime modulus p and a generator g of $\mathbb{Z}_p^*$, let's first define the* discrete exponential function $\exp_g : \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ *as, for all $r \in \mathbb{Z}_{p-1}$,*

$$\exp_g(r) = g^r. \tag{2}$$

Definition 1.5 (discrete log function)**.** *Relative to a prime modulus p and a generator g of $\mathbb{Z}_p^*$, define the* discrete log function $\log_g : \mathbb{Z}_p^* \to \mathbb{Z}_{p-1}$ *as the inverse of the discrete exponential function* $\exp_g$*. The input to* $\log_g$ *is some $s \in \mathbb{Z}_p^*$, and the output is the value of $r \in \mathbb{Z}_{p-1}$ such that $g^r = s$.*

1.3 Discrete exponential problem

For the *discrete exp problem*, the input is (p, g, r), where

- p is an n-bit prime.
- g is a generator of $\mathbb{Z}_p^*$.
- $r \in \mathbb{Z}_{p-1}$.

And the output is: $\exp_g(r)$, which is g^r.

How hard is it to compute this function? Of course, g^r is equal to g multiplied by itself r times. So $\exp_g(r)$ can obviously be computed by r multiplications (the precise number of multiplications is actually $r-1$). But r is an n-bit number, so r can be roughly as large as 2^n. That's an exponentially large number of multiplication operations! Using this approach, the circuit-size would be exponential in n. But there's a simple trick for doing this more efficiently, called the *repeated squaring trick*.

1.3.1 Repeated squaring trick

The idea is the following. You multiply g by itself to get g^2. Then you multiply g^2 by itself to get g^4. And then you multiply g^4 by itself get g^8, and so on. This way, you can compute g^{2^n} at the cost of only n multiplications. That's how the repeated squaring trick works when the exponent r is a power of 2.

What if r is not a power of 2? The above idea can be adjusted to compute *any* n-bit exponent r with fewer than $2n$ multiplications. The idea is based on the fact that g^r can be written as

$$g^{r_n \cdots r_3 r_2 r_1} = ((\cdots (g^{r_n})^2 \cdots g^{r_3})^2 g^{r_2})^2 g^{r_1}, \tag{3}$$

where $r_n ... r_3 r_2 r_1$ is the binary representation of an n-bit exponent r.

Note that $O(n)$ multiplications, at cost $O(n \log(n))$ gates each, leads to a classical gate cost of order $O(n^2 \log(n))$ for computing the discrete exponential function.

1.4 Discrete log problem

For the *discrete log problem (DLP)*, the input is (p, g, s), where

- p is an n-bit prime.
- g is a generator of $\mathbb{Z}_p^*$.

- $s \in \mathbb{Z}_p^*$.

And the output is: $\log_g(s)$, which is the $r \in \mathbb{Z}_{p-1}$ for which $g^r = s$.

2 Shor's algorithm for the discrete log problem

The overall idea behind Shor's algorithm is to convert the discrete log problem (DLP) into a generalization of Simon's problem, and then to solve that generalization of Simon's problem. Since DLP is not a black-box problem, how can such a conversion work? It works by creating a function with a property similar to Simon's *that can be efficiently implemented* so as to simulate black-box queries to the function. This is Shor's function.

2.1 Shor's function with a property similar to Simon's

Recall that an instance of the discrete log problem consists of three n-bit numbers: p (an n-bit prime), g (a generator of $\mathbb{Z}_p^*$), and s (an element of $\mathbb{Z}_p^*$).

Shor's function $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ is defined as

$$f(a_1, a_2) = g^{a_1} s^{a_2}, \tag{4}$$

for all $a_1, a_2 \in \mathbb{Z}_{p-1}$. What's interesting about this function is where the collisions are. When is $f(a_1, a_2) = f(b_1, b_2)$?

Theorem 2.1. *For an instance (p, g, s) of DLP, the function $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$ defined by Eq. (4) has the property that*

$$f(a_1, a_2) = f(b_1, b_2) \quad \textit{if and only if } (a_1, a_2) - (b_1, b_2) \textit{ is a multiple of } (r, 1), \tag{5}$$

where $r = \log_g(s)$.

The significance of this is that it resembles the Simon property. It's the analogue of the Simon property when we switch from mod 2 arithmetic to mod $p-1$ arithmetic. To see this, recall that the Simon property can be stated as: $f(a) = f(b)$ if and only if $a \oplus b$ is either a string of n zeroes or the string r. Notice that: $a \oplus b$ is the same as $a - b$ in mod 2 arithmetic. So we can write the Simon property as:

> **Simon property for** $f : (\mathbb{Z}_2)^n \to (\mathbb{Z}_2)^n$
> $f(a_1, \ldots, a_n) = f(b_1, \ldots, b_n)$ if and only if $(a_1, \ldots, a_n) - (b_1, \ldots, b_n)$ is a multiple of $(r_1, \ldots, r_n)$ in mod 2 arithmetic.

And here's again is the property that Shor's function has:

> **Property of Shor's function for** $f : \mathbb{Z}_{p-1} \times \mathbb{Z}_{p-1} \to \mathbb{Z}_p^*$
> $f(a_1, a_2) = f(b_1, b_2)$ if and only if $(a_1, a_2) - (b_1, b_2)$ is a multiple of $(r, 1)$ in mod $p - 1$ arithmetic.

Proof of Theorem 2.1. The proof is elementary, but we'll go through it carefully. Although we do not know what r is, we do know that an r exists such that $s = g^r$. This means that $s^{a_2} = g^{ra_2}$. Therefore, $f(a_1, a_2) = g^{a_1} g^{-ra_2} = g^{a_1 - ra_2}$. It's nice that to write f this way, because then

$$f(a_1, a_2) = f(b_1, b_2) \tag{6}$$

$$\text{if and only if} \quad a_1 - ra_2 = b_1 - rb_2 \tag{7}$$

$$\text{if and only if} \quad (a_1, a_2) \cdot (1, -r) = (b_1, b_2) \cdot (1, -r) \tag{8}$$

$$\text{if and only if} \quad \big((a_1, a_2) - (b_1, b_2)\big) \cdot (1, -r) = 0. \tag{9}$$

In equation (9), the dot-product being zero is like an orthogonality relation between the vector $(a_1, a_2) - (b_1, b_2)$ and the vector $(1, -r)$. Here's a schematic sketch of the vector $(1, -r)$.

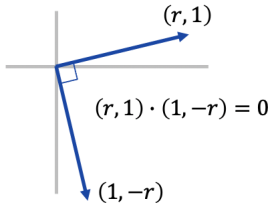

Figure 1: Schematic illustration of $(r, 1)$ orthonormal to $(1, -r)$.

Notice that the vector $(r, 1)$ is orthogonal to the vector $(1, -r)$ in that their dot-product is zero. Now, in a two-dimensional space, the vectors that are orthogonal to a particular vector are all multiples of *one* of the orthogonal vectors. So, we might expect $(a_1, a_2) - (b_1, b_2)$ to be a multiple of $(r, 1)$. This intuition (valid for vector spaces with inner products) is confirmed in our context by a simple calculation:

$$(v_1, v_2) \cdot (1, -r) = 0 \tag{10}$$

$$\text{if and only if} \quad v_1 = rv_2 \tag{11}$$

$$\text{if and only if} \quad v_2 = k \ \text{ and } \ v_1 = rk, \ \text{ for some } k \in \mathbb{Z}_{p-1} \tag{12}$$

$$\text{if and only if} \quad (v_1, v_2) = k(r, 1), \ \text{ for some } k \in \mathbb{Z}_{p-1}. \tag{13}$$

Therefore, $f(a_1, a_2) = f(b_1, b_2)$ if and only if $(a_1, a_2) - (b_1, b_2)$ is a multiple of $(r, 1)$. □

2.2 The Simon mod m problem

We've established that Shor's function satisfies a variant of Simon's property where the domain of the function is changed from $\{0,1\}^n = (\mathbb{Z}_2)^n$ to $(\mathbb{Z}_{p-1})^2$.

Now, what we're going to do is digress from DLP to investigate the generalization of Simon's problem, where the modulus is changed from 2 to m. We'll first find an efficient quantum black-box algorithm for the Simon mod m problem, where we are given a function

$$f : (\mathbb{Z}_m)^d \to T, \tag{14}$$

where T can be any set—but for convenience we'll assume that $T = \{0,1\}^k$ for some k (any T can be enlarged so that it's size is a power of 2).

Definition 2.1 (of an m-to-1 function)**.** *A function is* m-to-1 *if, for every value attained by the function, there are exactly* m *preimages. In other words, all colliding sets are of size* m*.*

Recall that, in the Simon's original problem, we had colliding pairs. Now, here is a mod m analogue of the original Simon property.

Definition 2.2 (Simon mod m property)**.** *An* m*-to-1 function* $f : (\mathbb{Z}_m)^d \to T$ *satisfies the* Simon mod m property*, if there exists a non-zero* $r \in (\mathbb{Z}_m)^d$ *such that: for all* $a, b \in (\mathbb{Z}_m)^d$*, it holds that* $f(a) = f(b)$ *if and only if* $a - b$ *is a multiple of* r*.*

Notice that the Simon mod m property is equivalent to the property that every colliding set of f is of the form $\{a, a+r, a+2r, \ldots, a+(m-1)r\} = \{a+kr : k \in \mathbb{Z}_m\}$, for some $a \in (\mathbb{Z}_m)^d$. Figure 2 is a schematic diagram of the two-dimensional case.

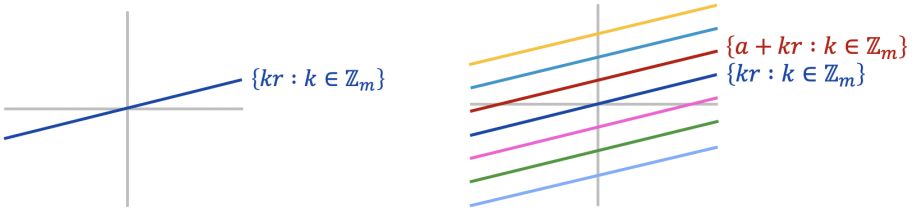

Figure 2: Each colliding set is one of the colored lines (an offset of the line $\{kr : k \in \mathbb{Z}_m\}$).

Think of $\{kr : k \in \mathbb{Z}_m\}$ as a line in $(\mathbb{Z}_m)^d$ passing through the origin. And think of $\{a+kr : k \in \mathbb{Z}_m\}$ as an *offset* of the line by $a \in (\mathbb{Z}_m)^d$. Each colliding set is an offset.

Recall that, for the original Simon property, the colliding pairs were of the form $\{a, a \oplus r\}$, which are offsets of $\{0, r\}$. In that case, the colored lines in figure 2 correspond to the colliding pairs.

Definition 2.3. *For a function $f : (\mathbb{Z}_m)^d \to T$, define an f-query as unitary operation U_f such that, for every $(a_1, a_2, \ldots, a_n) \in (\mathbb{Z}_m)^d$ and $b \in T$,*

$$U_f\big(|a_1\rangle\,|a_2\rangle \ldots |a_n\rangle\,|b\rangle\big) = |a_1\rangle\,|a_2\rangle \ldots |a_n\rangle\,|b \oplus f(a_1, a_2, \ldots, a_n)\rangle\,, \tag{15}$$

where the $\oplus$ operation denotes bit-wise XOR (recall our assumption that $T = \{0,1\}^k$).

We use the following notation for f-queries.

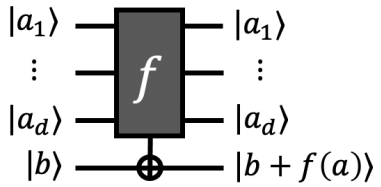

Figure 3: Notation for a quantum f-query gate for $f : (\mathbb{Z}_m)^d \to T$.

Notice that the lines are drawn thicker than those in our previous quantum circuits where the lines carried qubits. This is to help convey the fact that the data flowing through these lines is generally not qubits, but states of dimension higher than 2. The first d lines are carrying an m-dimensional quantum states, and the last line is carrying a $|T|$-dimensional state.

Definition 2.4 (Simon mod m problem)**.** *For Simon's problem mod m, you're given a black-box that computes an f-query for an unknown function f that is promised to have the Simon mod m property, and your goal is to determine the parameter r, based on queries to f.*

How do we solve the Simon mod m problem? Let's start by recalling the algorithm for the original Simon's problem. It was based on repeated runs of this circuit that makes a single f-query.

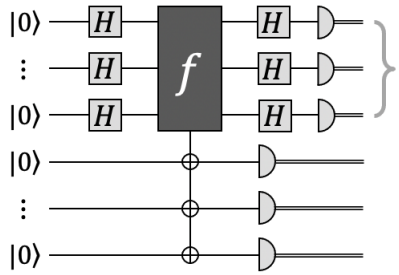

Figure 4: Circuit used for Simon's problem.

Each run of the circuit to produces a uniformly random element of of the orthogonal complement of r. After a few runs, we obtain enough such b's to be able to deduce r by solving a system of linear equations in mod 2 arithmetic.

Notice all the one-qubit Hadamard gates in the above circuit. Come to think of it, the Hadamard gate has been a very prominent part of every quantum algorithm and communication protocol that we've seen so far. For Simon mod m, the registers are not binary. A unitary operation on an m-dimensional register is an $m \times m$ matrix. It turns out that the Hadamard transform has a natural m-dimensional generalization, the *Fourier transform* F_m, which will be explained in the next section.

The algorithm for the Simon mod m problem will be based on a quantum circuit similar to the one in figure 4, but with Fourier transforms in place of Hadamard transforms.

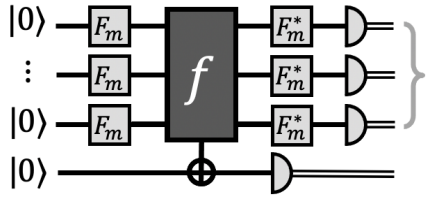

Figure 5: Proposed circuit for the Simon mod m problem.

(In the $m = 2$ case, since $H^* = H$, the distinction between F and F^* didn't matter.) To understand this circuit, let's first acquaint ourselves with the Fourier transform.

2.3 The quantum Fourier transform

A primitive m^{th} root of unity is a complex number of the form $e^{2\pi i/m}$. Call this number ω. Clearly, $\omega^m = 1$. Here's where ω, and all its powers, lie in the complex plane $\mathbb{C}$.

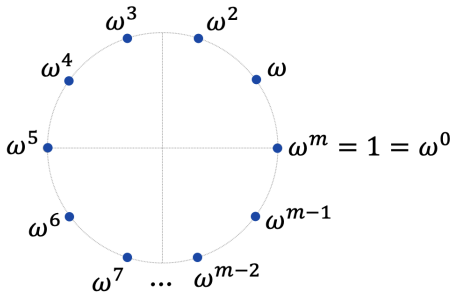

Figure 6: The m^{th} roots of unity are m equally spaces points on the unit circle in $\mathbb{C}$.

If we sum all these powers of ω, we get zero: $1 + \omega + \omega^2 + \cdots + \omega^{m-1} = 0$. Can you see why? Moreover, if we sum all the powers of ω^k (assuming $1 \leq k \leq m-1$) we also get zero.

Exercise 2.1. *Prove that, for all $k \in \{1, 2, \ldots, m-1\}$, it holds that*

$$1 + \omega^k + \omega^{2k} + \cdots + \omega^{(m-1)k} = \sum_{j=0}^{m-1} \omega^{kj} = 0. \tag{16}$$

What about the powers of ω^m? Since $\omega^m = 1$, it's obvious that $\sum_{j=0}^{m-1} \omega^{mj} = m$.

Definition 2.5 (Quantum Fourier transform)**.** *The Fourier transform mod m is the $m \times m$ matrix*

$$F_m = \frac{1}{\sqrt{m}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{m-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{2(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{m-1} & \omega^{2(m-1)} & \cdots & \omega^{(m-1)^2} \end{bmatrix}. \tag{17}$$

Note that (after the normalization factor $\frac{1}{\sqrt{m}}$) the first column is 1s, the second column is the powers of ω, the third column is powers of ω^2 and so on.

Exercise 2.2. *Prove that F_m is unitary.*

Exercise 2.3. *Prove that the inverse Fourier transform F_m^* is*

$$F_m^* = \frac{1}{\sqrt{m}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega^{-1} & \omega^{-2} & \cdots & \omega^{-(m-1)} \\ 1 & \omega^{-2} & \omega^{-4} & \cdots & \omega^{-2(m-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{-(m-1)} & \omega^{-2(m-1)} & \cdots & \omega^{-(m-1)^2} \end{bmatrix}. \tag{18}$$

For all $a \in \mathbb{Z}_m$, applying the Fourier transform F_m to the computational basis state $|a\rangle$ results the state

$$F|a\rangle = \tfrac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \omega^{ja} |j\rangle , \tag{19}$$

which corresponds to column a of F_m. We call this a *Fourier basis state*. Similarly, applying the inverse Fourier transform F_m^* to $|a\rangle$ results in the state

$$F^*|a\rangle = \tfrac{1}{\sqrt{m}} \sum_{j=0}^{m-1} \omega^{-ja} |j\rangle . \tag{20}$$

If there are n registers that are each m-dimensional, and you apply F_m to each register (which is $F_m \otimes F_m \otimes \cdots \otimes F_m = F_m^{\otimes n}$ on the n-register system) then, for all $(a_1, a_2, \ldots, a_n) \in \mathbb{Z}_m^n$,

$$(F_m)^{\otimes n} |a_1, a_2, \ldots, a_n\rangle = \frac{1}{\sqrt{m^n}} \sum_{b \in \mathbb{Z}_m^n} \omega^{a \cdot b} |b_1, b_2, \ldots, b_n\rangle \tag{21}$$

$$(F_m^*)^{\otimes n} |a_1, a_2, \ldots, a_n\rangle = \frac{1}{\sqrt{m^n}} \sum_{b \in \mathbb{Z}_m^n} \omega^{-a \cdot b} |b_1, b_2, \ldots, b_n\rangle , \tag{22}$$

where $a \cdot b = (a_1, a_2, \ldots, a_n) \cdot (b_1, b_2, \ldots, b_n) = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$ (in modulo m arithmetic).

2.4 Query algorithm for Simon mod m

The algorithm is based on this circuit.

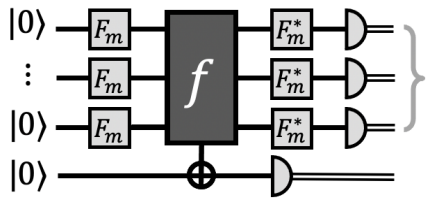

Figure 7: Circuit for Simon mod m problem.

What's the output of this circuit? We'll analyze this for the case where $d = 2$, which the following circuit.

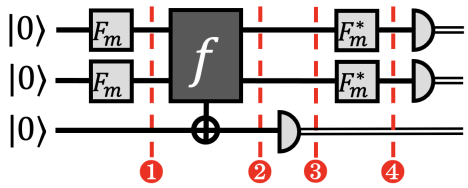

Figure 8: Circuit for Simon mod m problem in the case where $d = 2$.

Let's trace through the evolution of the state of the registers.

State at stage ❶

Since $F_m |0\rangle = \frac{1}{\sqrt{m}}(|0\rangle + |1\rangle + \cdots + |m-1\rangle)$, this state is

$$\frac{1}{\sqrt{m^2}} \sum_{a \in \mathbb{Z}_m \times \mathbb{Z}_m} |a_1, a_2\rangle |0\rangle . \tag{23}$$

State at stage ❷

Since the query maps each basis state $|a_1, a_2\rangle |0\rangle$ to $|a_1, a_2\rangle |f(a_1, a_2)\rangle$, when the input is superposition, the output of the query is

$$\frac{1}{\sqrt{m^2}} \sum_{a \in \mathbb{Z}_m \times \mathbb{Z}_m} |a_1, a_2\rangle |f(a_1, a_2)\rangle . \tag{24}$$

State at the first two registers at stage ❸

Measuring the state of the third register causes its state to collapse to some value of f and the state of the first two registers to collapse to a uniform superposition of all the pre-images of that value of f. This preimage is one of the collapsing sets of f (one of the colored lines in figure 2). Therefore, the state after this measurement is

$$\frac{1}{\sqrt{m}} \sum_{k \in \mathbb{Z}_m} |(a_1, a_2) + k(r_1, r_2)\rangle , \tag{25}$$

for some $(a_1, a_2) \in \mathbb{Z}_m \times \mathbb{Z}_m$.

This state is identical to the state that is the outcome of the following process:

1. Randomly choose one of the colliding sets (the colored lines in figure 2).
2. Take the uniform superposition of the elements of that colliding set.

State at the first two registers at stage ❹

$$F_m^* \otimes F_m^* \left(\tfrac{1}{\sqrt{m}} \sum_{k \in \mathbb{Z}_m} |(a_1, a_2) + k(r_1, r_2)\rangle \right) \tag{26}$$

$$= \tfrac{1}{\sqrt{m}} \sum_{k \in \mathbb{Z}_m} \left(F_m^* \otimes F_m^* \, |(a_1, a_2) + k(r_1, r_2)\rangle \right) \tag{27}$$

$$= \tfrac{1}{\sqrt{m}} \sum_{k \in \mathbb{Z}_m} \left(\tfrac{1}{\sqrt{m^2}} \sum_{b \in \mathbb{Z}_m \times \mathbb{Z}_m} \omega^{-(a+kr)\cdot b} \, |b_1, b_2\rangle \right) \tag{28}$$

$$= \tfrac{1}{\sqrt{m}} \sum_{k \in \mathbb{Z}_m} \left(\tfrac{1}{m} \sum_{b \in \mathbb{Z}_m \times \mathbb{Z}_m} \omega^{-a\cdot b} \omega^{-k(r\cdot b)} \, |b_1, b_2\rangle \right) \tag{29}$$

$$= \tfrac{1}{\sqrt{m}} \sum_{b \in \mathbb{Z}_m \times \mathbb{Z}_m} \omega^{-a\cdot b} \left(\tfrac{1}{m} \sum_{k \in \mathbb{Z}_m} \omega^{-k(r\cdot b)} \right) |b_1, b_2\rangle \, . \tag{30}$$

Notice that Eq. (30) contains an expression for the amplitude of $|b_1, b_2\rangle$ for any $(b_1, b_2) \in \mathbb{Z}_m \times \mathbb{Z}_m$. Since $|\omega^{-a\cdot b}| = 1$, what's important in Eq. (30) is the expression in parentheses, which is

$$\tfrac{1}{m} \sum_{k \in \mathbb{Z}_m} \omega^{-k(r\cdot b)} = \begin{cases} 1 & \text{if } (r_1, r_2) \cdot (b_1, b_2) = 0 \\ 0 & \text{if } (r_1, r_2) \cdot (b_1, b_2) \neq 0. \end{cases} \tag{31}$$

This implies that, for any $(b_1, b_2) \in \mathbb{Z}_m \times \mathbb{Z}_m$,

$$\Pr[(b_1, b_2)] = \begin{cases} \frac{1}{m} & \text{if } (r_1, r_2) \cdot (b_1, b_2) = 0 \\ 0 & \text{if } (r_1, r_2) \cdot (b_1, b_2) \neq 0. \end{cases} \tag{32}$$

Therefore the output of the circuit in figure 8 is a uniformly random sample from the set $\{(b_1, b_2) \in \mathbb{Z}_m \times \mathbb{Z}_m : b_1 r_1 + b_2 r_2 = 0\}$, which we loosely call the *orthogonal complement* of (r_1, r_2).

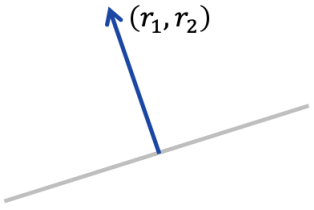

Figure 9: Schematic illustration of the orthogonal complement of (r_1, r_2).

All of the preceeding analysis generalizes in a straightforward way from two dimensions to d dimensions.

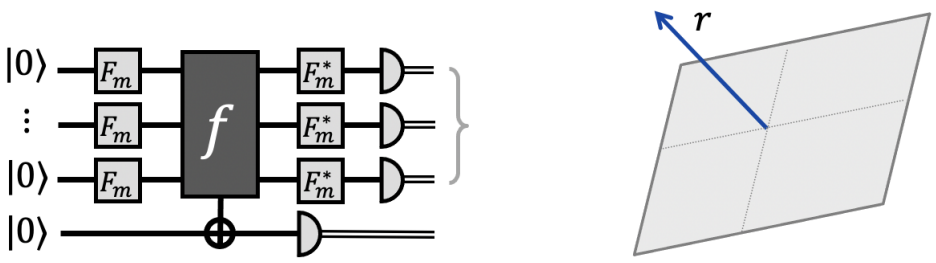

Figure 10: Simon mod m circuit produces a random element of the orthogonal complement of r.

In that case, output from the first d registers is a uniformly random element of the set $\{b \in \mathbb{Z}_m^d : b \cdot r = 0\}$ (the orthogonal complement of r).

As with Simon's algorithm, after repeated runs of the circuit in figure 10, we obtain several b's, from which we can hope to deduce r by solving a system of linear equations in mod m arithmetic (we'll return to this matter in section xxx).

2.5 Returning to the discrete log problem

Now that we've analyzed the Simon mod m problem, let's get back to the original problem that we're concerned with, which is the discrete log problem. The input is: p, an n-bit prime number; g, a generator of $\mathbb{Z}_p^*$; and s, an element of $\mathbb{Z}_p^*$ (all three inputs are binary strings of length n). And the goal to produce $\log_g(s)$, which is the unique $r \in \mathbb{Z}_{p-1}$ for which $s = g^r$.

The reason why we turned our attention to the Simon mod m problem is that there is a special Shor function that satisfies the Simon mod m property (with m set to $p-1$), and a solution to that instance of Simon mod $p-1$ yields $r = \log_g(s)$.

How can we use our solution to Simon mod m to solve an instance of DLP, which is not in the query framework? The idea is to efficiently *implement* the query gate of the Shor function, as well as the other parts of the query circuit in terms of qubits and elementary gates (1- and 2-qubit gates). The elements of $\mathbb{Z}_{p-1}$ and $\mathbb{Z}_p^*$ can be represented by their n-bit binary representations as n-bit strings. Imagine a $3n$-qubit quantum circuit of the form of figure 11, where each grey box represents a quantum circuit consisting of elementary gates acting on qubits.

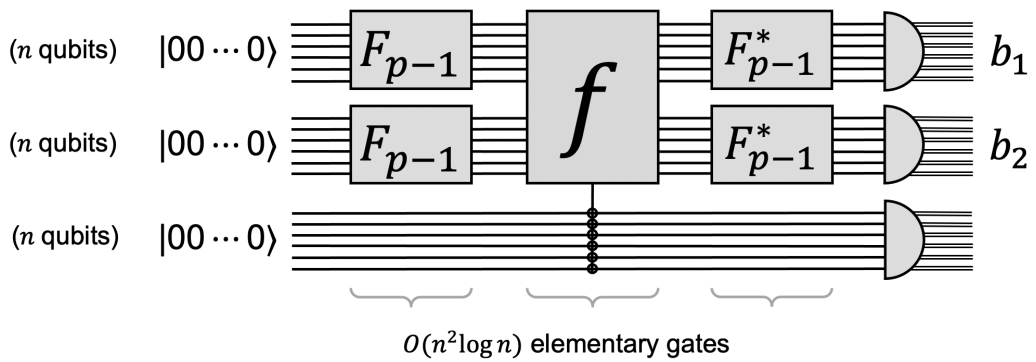

Figure 11: Implementation of the query algorithm in figure 12.

This circuit is an implementation of the circuit in figure 12.

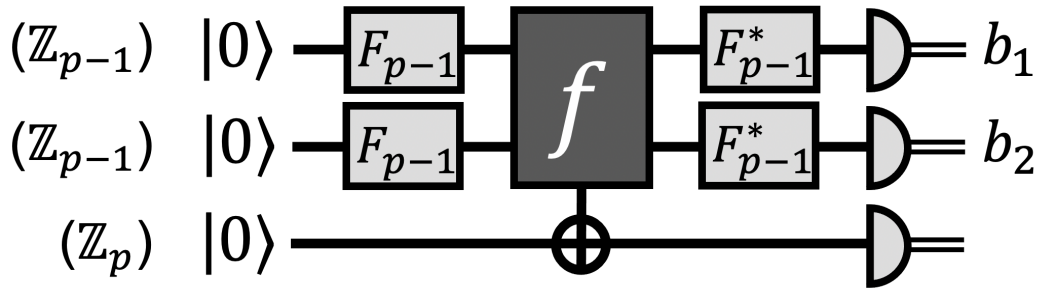

Figure 12: This is what is simulated by the circuit in figure 11.

A property of the Shor function f (defined is Eq. (4)) that's crucial to make this work is that $f(a_1, a_2) = g^{a_1} s^{a_2}$ can be computed *efficiently* by a classical circuit. The exponentiations can be computed efficiently using the repeated squaring trick (explained in section 1.3.1).

That's the overall idea behind Shor's algorithm for the discrete log problem. However, there are a few loose ends that have not been explained.

2.6 Loose ends

One loose end is that concerns technicalities in extracting r from repeated runs of the circuit in figure 8. This is discussed in section 2.6.1.

Another loose end is an important issue that arises in implementing the f-query, which is discussed in sections 2.6.2 and 2.6.3.

Finally, there's the matter of efficiently computing F_m, which is discussed in section 2.6.4.

2.6.1 How to extract r

As was done for the original Simon problem, we can set up a system of linear equations in mod m arithmetic. However, a complication arises when $\mathbb{Z}_m$ is not a field—and in fact, it's *not* a field in our case of interest, where $m = p - 1$, for a prime p.

Recall that, for DLP, the quantum circuit in figure 8 produces a uniformly random (b_1, b_2) such that

$$(b_1, b_2) \cdot (r, 1) \equiv 0 \pmod{p-1}. \tag{33}$$

How do we calculate r from this? From Eq. (33), we can solve for r as

$$r = -b_2/b_1 \bmod p - 1. \tag{34}$$

But, for this to work, b_1 must have an inverse modulo $p - 1$. It might not.

It can be proven that the fraction of elements of $\mathbb{Z}_{p-1}$ that have inverses is not too small. I won't get into further details here about how this is quantified. But, as a result of this, we can simply repeat the process of running the circuit in figure 8 a few times until we obtain a (b_1, b_2) where b_1 has an inverse in $\mathbb{Z}_{p-1}$.

2.6.2 How *not to* compute an f-query

Suppose that there is an efficient classical circuit that computes a function f. How do we simulate an f-query (as in Definition 2.3) in terms of elementary operations on qubits?

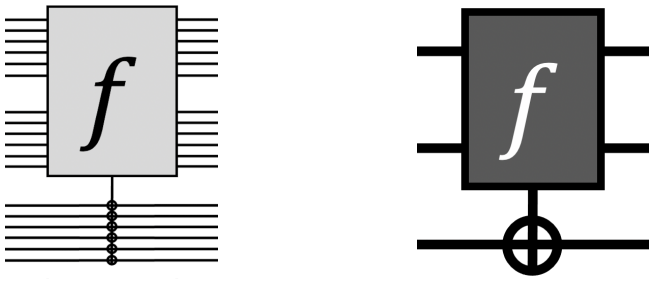

Figure 13: A circuit on qubits (left) so as to simulate an f-query (right).

We know from the lectures notes in Part I notes that any classical circuit can be simulated by a quantum circuit with the same efficiency (up to a constant factor). However, that simulation used ancilla qubits and resulted in a quantum quantum circuit that maps each computational basis state of the form $|a\rangle |0^m\rangle |b\rangle$ to

$|a\rangle\,|g(a)\rangle\,|f(a)\oplus b\rangle$, where $|0^m\rangle$ is a string of several $|0\rangle$ qubits that are used as ancillas, and $g(a)$ consists of some intermediate results of the computation.

To help clarify the point, here again in the quantum circuit from Part I, for computing the majority of three bits.

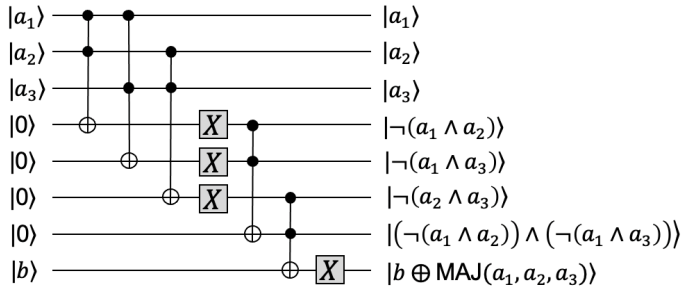

Figure 14: Quantum circuit that simulates classical circuit for the majority function.

The first three qubits contain the input to the function. The last qubit is where the output is XORed. And there are four ancilla qubits, whose states at the end of the computation are shown. We can refer to this as the "garbage", because we might think of disposing of those four qubits. Or ignoring them.

Is this OK for simulating an f-query? No, this leads to a problem if the f-query is applied at a state that's a superposition of computational basis states, such as

$$\sum_{a,b} \alpha_{a,b}\,|a\rangle\,|b\rangle\,|0^m\rangle\,. \tag{35}$$

The above purported implementation of an f-query maps this state to

$$\sum_{a,b} \alpha_{a,b}\,|a\rangle\,|b\oplus f(a)\rangle\,|g(a)\rangle \neq \Big(\sum_{a,b} \alpha_{a,b}\,|a\rangle\,|b\oplus f(a)\rangle\Big)\,|g(a)\rangle\,. \tag{36}$$

The state of the first two registers need not be

$$\sum_{a,b} \alpha_{a,b}\,|a\rangle\,|b\oplus f(a)\rangle\,. \tag{37}$$

(Note that, in Eqns. (35)(36)(37), I've moved the garbage register to the end so that the statement of inequality is simpler to write.)

But this problem has a simple remedy.

2.6.3 How *to* compute an f-query

The first step is to compute f with the ancilla registers. After that, the computation is reversed, so as to restore the ancilla registers back to state $|0\rangle$. But just before the reversal, the answer is XORed to another register, using CNOT gates.

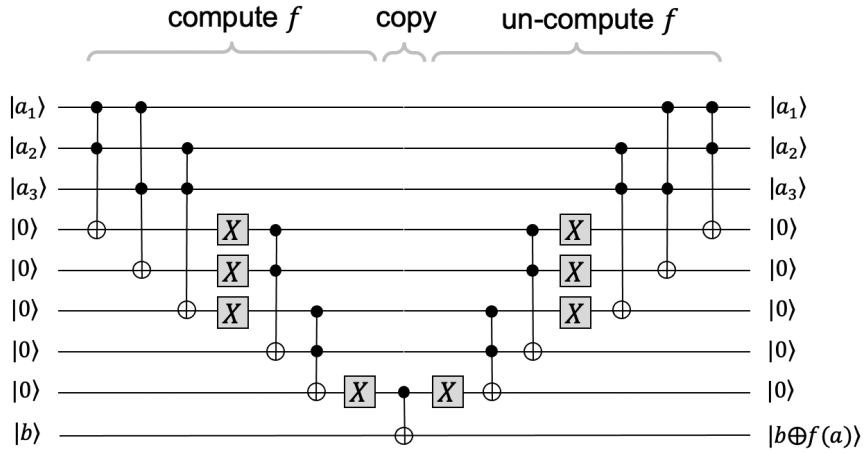

Figure 15: Clean computation of the majority function (the ancilla qubits are reset to $|0\rangle$).

The resulting circuit computes the f-query and restores the ancilla qubits back to their original states. Therefore, this works even if the f-query is applied to a superposition of computational basis states as

$$\sum_{a,b} \alpha_{a,b} |a\rangle |b \oplus f(a)\rangle |0^m\rangle = \Big(\sum_{a,b} \alpha_{a,b} |a\rangle |b \oplus f(a)\rangle\Big) |0^m\rangle . \tag{38}$$

The garbage register ends end up in a product state with the other registers.

Computing the function this way roughly doubles the number of gates needed.

2.6.4 How to compute the Fourier transform F_{p-1}

Another issue, is how to compute the Fourier transform mod $p-1$. It's an exponentially large matrix, and we need to compute it with a polynomial number of elementary gates acting on n qubits.

In fact, for modulus $p-1$, computing F_{p-1} is tricky. Shor's algorithm doesn't actually compute use this. Rather, it uses a Fourier transform with modulus a power of 2, which is easy to compute efficiently. The power of 2 is set so as to be close to $p-1$ (within a factor of 2 of $p-1$).

So Shor's algorithm uses the "wrong" Fourier transform. But it's not too wrong. Shor uses careful error analysis to show that, if the modulus is only off by a factor of two then the resulting wrong output state of the circuit is not too wrong. The result of the measurement still succeeds with some constant probability. I won't go into the details of this error analysis here. If you would like to see these details, you can find them in section 6 of Shor's paper[1] (https://arxiv.org/abs/quant-ph/9508027).

In the next section, I'll show you a simple efficient way of computing the Fourier transform when the modulus is a power of 2. Such Fourier transforms also have other applications, beyond the discrete log problem.

3 Computing the quantum Fourier transform

The Fourier transform F_m is defined in section 2.3. We're interested in computing F_m efficiently by a quantum circuit consisting of elementary operations acting on qubits. There's an elegant way to do this in the case where $m = 2^n$. Note that F_{2^n} is a unitary operation acting on n qubits. I will show you a fairly simple quantum circuit consisting of $O(n^2)$ gates that computes F_{2^n}.

It's useful to visualize how the powers of a (primitive) 2^n-th root of unity ω are aligned in the complex plane. They are 2^n equally spaced points on the unit circle, and here's what they look like when $n = 3$.

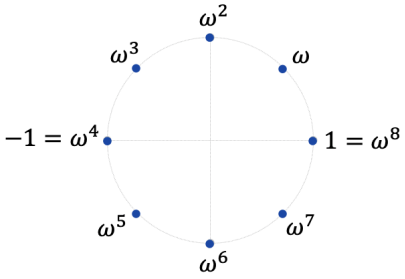

Figure 16: Powers of ω, a primitive 8-th root of unity in $\mathbb{C}$.

For a 2^n-th root of unity ω, a couple of simple but valuable observations are:

1. $\omega^{2^{n-1}} = -1$. (For example, in figure 16, ω is an 8-th root of unity and $\omega^4 = -1$.)

2. ω^2 is a 2^{n-1}-th root of unity. (In figure 16, ω^2 is an 4-th root of unity.)

[1] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer", *SIAM Journal on Computing*, Vol. 26, No. 5, pp. 1484–1509, October 1997.

3.1 Expressing F_{2^n} in terms of $F_{2^{n-1}}$

To get a feeling for how this works, let's first consider the case where $n = 3$. For each $a \in \{0,1\}^3 \equiv \{0,1,2,3,4,5,6,7\}$, the corresponding Fourier basis state $F_8 |a\rangle$ is

$$|000\rangle + \omega^a|001\rangle + \omega^{2a}|010\rangle + \omega^{3a}|011\rangle + \omega^{4a}|100\rangle + \omega^{5a}|101\rangle + \omega^{6a}|110\rangle + \omega^{7a}|111\rangle\,.$$

(where the normalization factor $\frac{1}{\sqrt{8}}$ is omitted). Question: Are these three qubits entangled or in a product state? In fact, this state can be written as

$$\begin{aligned} &|0\rangle \otimes \big(|00\rangle + \omega^a|01\rangle + \omega^{2a}|10\rangle + \omega^{3a}|11\rangle\big) \\ &\qquad + \omega^{4a}|1\rangle \otimes \big(|00\rangle + \omega^a|01\rangle + \omega^{2a}|10\rangle + \omega^{3a}|11\rangle\big) \qquad (39) \\ &= \big(|0\rangle + \omega^{4a}|1\rangle\big) \otimes \big(|00\rangle + \omega^a|01\rangle + \omega^{2a}|10\rangle + \omega^{3a}|11\rangle\big) \qquad (40) \\ &= \big(|0\rangle + \omega^{4a}|1\rangle\big) \otimes \big(|0\rangle + \omega^{2a}|1\rangle\big) \otimes \big(|0\rangle + \omega^a|1\rangle\big). \qquad (41) \end{aligned}$$

So $F_8 |a\rangle$ is a product of three 1-qubit states.

This generalizes to $F_{2^n} |a\rangle$, for all $n \geq 1$, as follows.

Lemma 3.1. *For all $n \geq 1$ and $a \in \{0,1\}^n$,*

$$F_{2^n}|a\rangle = \big(|0\rangle + \omega^{2^{n-1}a}|1\rangle\big) \otimes \cdots \otimes \big(|0\rangle + \omega^{4a}|1\rangle\big) \otimes \big(|0\rangle + \omega^{2a}|1\rangle\big) \otimes \big(|0\rangle + \omega^a|1\rangle\big), \quad (42)$$

where there is a normalization factor of $\frac{1}{2^{n/2}}$.

Our quantum circuit for computing the Fourier transform will make use of this structure. Remember that, if we compute a linear operator that matches the Fourier transform on all computational basis states then it must be the Fourier transform.

Let's return our attention to the $n = 3$ case, where the Fourier basis states are

$$F_8 |a\rangle = \big(|0\rangle + \omega^{4a}|1\rangle\big) \otimes \big(|0\rangle + \omega^{2a}|1\rangle\big) \otimes \big(|0\rangle + \omega^a|1\rangle\big), \quad (43)$$

for all $a \in \{0,1\}^3$. Each $a = a_2 a_1 a_0$ denotes an element of $\{0,1,2,3,4,5,6,7\}$ in the usual way (a_0 is the low-order bit and a_2 is the high-order bit).

Consider the state of the first qubit in Eq. (43). Since $\omega^4 = -1$, the state of the first qubit simplifies to $|0\rangle + (-1)^a |1\rangle$. Note that this is $|+\rangle$ when a is even and $|-\rangle$ when a is odd. The parity of a is determined by its low-order bit a_0. Therefore the first qubit of $F_8 |a\rangle$ is simply $H |a_0\rangle$.

What about the remaining qubits? Let $|\psi\rangle$ denote the second and third qubit in Eq. (43). That is,

$$|\psi\rangle = \big(|0\rangle + \omega^{2a}|1\rangle\big) \otimes \big(|0\rangle + \omega^a|1\rangle\big). \quad (44)$$

Now, please look at the $n = 2$ case of the expression in Eq. (42) in Lemma 3.1. Does $|\psi\rangle$ look like $F_4 |a\rangle$?

There is certainly a superficial resemblance; however, $|\psi\rangle$ and $F_4 |a\rangle$ are not *exactly* the same. The state $F_4 |a\rangle$ is with respect to a 4-th root of unity—not an 8-th root of unity. Another difference is that F_4 acts on 2 qubits; whereas the parameter a in Eq. (44) is a 3-bit integer.

It will be fruitful to explore in more detail the difference between $|\psi\rangle$ and $F_4 |a\rangle$. Let's see what these states look like in terms of the digits $a_2 a_1 a_0$ of the binary representation of a. We'll use the Greek letter ϖ to denote the 4-th root of unity, while reserving ω for the 8-th root of unity.

If we apply F_4 to $|a_2 a_1\rangle$ (the two higher order digits of a), the result is

$$F_4 |a_2 a_1\rangle = \big(|0\rangle + \varpi^{2[a_2 a_1]} |1\rangle\big) \otimes \big(|0\rangle + \varpi^{[a_2 a_1]} |1\rangle\big) \tag{45}$$

$$= \big(|0\rangle + (\omega^2)^{2[a_2 a_1]} |1\rangle\big) \otimes \big(|0\rangle + (\omega^2)^{[a_2 a_1]} |1\rangle\big) \tag{46}$$

$$= \big(|0\rangle + \omega^{2[a_2 a_1 0]} |1\rangle\big) \otimes \big(|0\rangle + \omega^{[a_2 a_1 0]} |1\rangle\big). \tag{47}$$

In the exponent, I've surrounded the binary representations by square brackets so that they can be clearly read; the two-digit number in the square brackets is either 0, 1, 2, or 3. Eq. (45) is due to Lemma 3.1. Eq. (46) is due to $\varpi = \omega^2$. And Eq. (47) is due[2] to $2[a_2 a_1] = [a_2 a_1 0]$.

Now let's express $|\psi\rangle$ in terms of $[a_2 a_1 a_0]$. This is

$$|\psi\rangle = \big(|0\rangle + \omega^{2[a_2 a_1 a_0]} |1\rangle\big) \otimes \big(|0\rangle + \omega^{[a_2 a_1 a_0]} |1\rangle\big) \tag{48}$$

$$= \big(|0\rangle + \omega^{2[a_2 a_1 0]} \omega^{2a_0} |1\rangle\big) \otimes \big(|0\rangle + \omega^{[a_2 a_1 0]} \omega^{a_0} |1\rangle\big). \tag{49}$$

Comparing Eq. (47) with Eq. (49), we can see precisely where they differ: the factors ω^{2a_0} and ω^{a_0} (highlighted in red). We'll refer to these as *phase corrections*.

Based on the above observations, let's try to compute $F_8 |a_2 a_1 a_0\rangle$ in terms of $F_4 |a_2 a_1\rangle$ and $H |a_0\rangle$, combined with additional gates that perform phase corrections. To perform the phase corrections, we introduce the following new 2-qubit gate.

Definition 3.1 (controlled-phase gate)**.** *For any $r \in \mathbb{Z}$, the 2-qubit* controlled-phase gate (with phase $e^{2\pi i/r}$) *is defined as the unitary operation that, for all $a, b \in \{0, 1\}$,*

[2] This is a simple maneuver. It's the binary equivalent of what we do in base ten when we multiply an integer by ten: we add a zero digit and shift all the other digits left. 10 times 23 is 230.

maps $|a\rangle\,|b\rangle$ *to* $(e^{2\pi i/r})^{ab}\,|a\rangle\,|b\rangle$*. The unitary matrix for the gate is*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & e^{2\pi i/r} \end{bmatrix} \tag{50}$$

and our circuit notation for this gate is shown in figure 17.

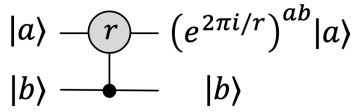

Figure 17: Our notation for the controlled-phase gate, with phase $e^{2\pi i/r}$.

Now we'll analyze the following circuit and show that it computes F_8.

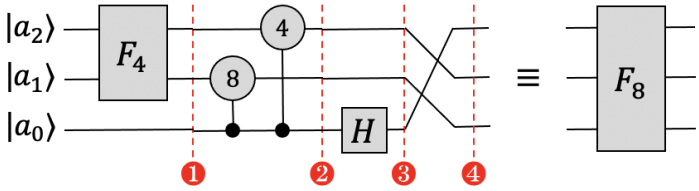

Figure 18: Caption.

We will trace through the stages of this circuit.

State at stage ❶

From Eq. (47), this is $\left(|0\rangle + \omega^{2[a_2a_10]}\,|1\rangle\right) \otimes \left(|0\rangle + \omega^{[a_2a_10]}\,|1\rangle\right) \otimes |a_0\rangle$.

State at stage ❷

The two controlled-phase gates change the state to

$$\left(|0\rangle + \omega^{2[a_2a_10]}\omega^{2a_0}\,|1\rangle\right) \otimes \left(|0\rangle + \omega^{[a_2a_10]}\omega^{a_0}\,|1\rangle\right) \otimes |a_0\rangle \tag{51}$$

$$= \left(|0\rangle + \omega^{2[a_2a_1a_0]}\,|1\rangle\right) \otimes \left(|0\rangle + \omega^{[a_2a_1a_0]}\,|1\rangle\right) \otimes |a_0\rangle \tag{52}$$

$$= \left(|0\rangle + \omega^{2a}\,|1\rangle\right) \otimes \left(|0\rangle + \omega^{a}\,|1\rangle\right) \otimes |a_0\rangle\,. \tag{53}$$

State at stage ❸

Applying a Hadamard gate to the third qubit changes the state to

$$\left(|0\rangle + \omega^{2a}|1\rangle\right) \otimes \left(|0\rangle + \omega^{a}|1\rangle\right) \otimes \left(|0\rangle + (-1)^{a_0}|1\rangle\right) \tag{54}$$

$$= \left(|0\rangle + \omega^{2a}|1\rangle\right) \otimes \left(|0\rangle + \omega^{a}|1\rangle\right) \otimes \left(|0\rangle + \omega^{4a}|1\rangle\right). \tag{55}$$

Notice that this is the state in Eq. (43), except the qubits are in the wrong order.

State at stage ❹

Moving the third qubit to the left and shifting the other two qubits right yields

$$\left(|0\rangle + \omega^{4a}|1\rangle\right) \otimes \left(|0\rangle + \omega^{2a}|1\rangle\right) \otimes \left(|0\rangle + \omega^{a}|1\rangle\right) = F_8|a\rangle\,. \tag{56}$$

What we have shown is a special case of the following more general recurrence for F_{2^n}.

Theorem 3.1. *For all $n \geq 1$, the Fourier transform F_{2^n} can be expressed as*

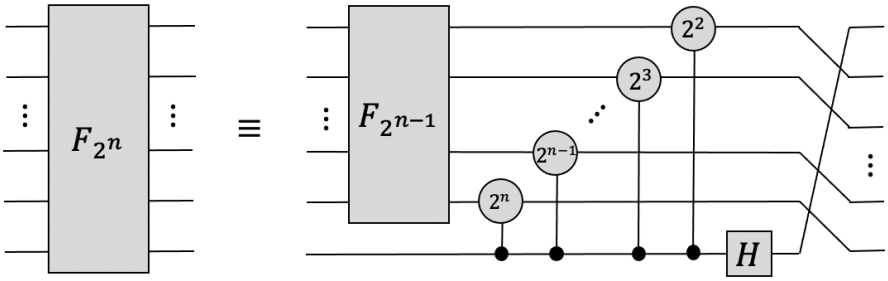

Figure 19: Quantum circuit for F_{2^n} recursively constructed in terms of $F_{2^{n-1}}$.

This is straightforward to verify, along the lines of the analysis for the $n = 3$ case.

3.2 Unravelling the recurrence

By repeatedly applying Theorem 3.1, we can construct a quantum circuit for the Fourier transform for any $n \geq 1$. The recurrence bottoms out at $n = 1$, where $F_2 = H$. Here is the circuit for the $n = 4$ case.

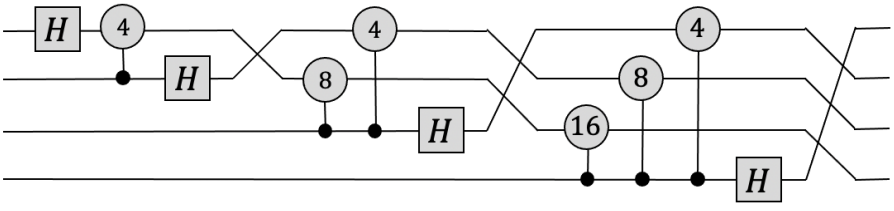

Figure 20: Quantum circuit for F_{2^4}.

Notice that there are places where the qubits are rearranged. Instead of doing these rearrangements, we can move the gates around. Then there's just one net rearrangement at the very end, which turns out to be a reversal of the order of the qubits.

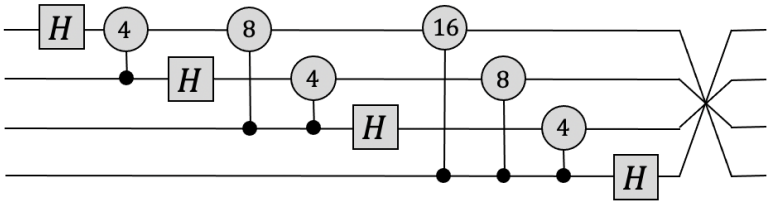

Figure 21: Quantum circuit for F_{2^4}, with rearrangements deferred until the end.

How do we perform this reversal of the order of the qubits? We can define a 2-qubit SWAP gate.

Definition 3.2 (SWAP gate)**.** *The 2-qubit* SWAP gate *is defined as the unitary operation that, for all $a, b \in \{0, 1\}$, maps $|a\rangle\,|b\rangle$ to $|b\rangle\,|a\rangle$. The unitary matrix for the gate is*

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{57}$$

and circuit notation for this gate is shown in figure 22.

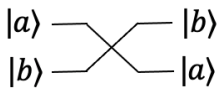

Figure 22: Notation for SWAP gate.

The reversal consists of around $n/2$ SWAP gates.

Intuitively, the notation for SWAP gate suggests a physically movement of the qubits. In principle, one could do that, but it would be nice not to be adding a brand new elementary operation into our repertoire of 2-qubit gates (if we can avoid it). It turns out this this SWAP gate can be computed by three CNOT gates.

Exercise 3.1. *Show that the 2-qubit SWAP gate can be implemented by a sequence of three CNOT gates (appropriately oriented).*

Finally, let's count the number of gates in our circuit construction (which is the natural generalization of the construction in figure 21 to F_{2^n}). There are n Hadamard gates, one for each qubit. The number of controlled-phase gates is

$$1 + 2 + 3 + \cdots + n - 1 = \frac{n(n-1)}{2} = O(n^2). \tag{58}$$

And there are most $\frac{n}{2}$ SWAP gates—which translates into $\frac{3n}{2}$ CNOT gates. That's a total of $O(n^2)$ gates for computing the Fourier transform F_{2^n}.

Exercise 3.2 (straightforward)**.** *Give a quantum circuit that computes the inverse Fourier transform $F_{2^n}^*$ with $O(n^2)$ gates. (There are two different approaches that lead to slightly different circuits.)*

4 Phase estimation algorithm

In this section, we are going to investigate the *phase estimation problem*, which involves finding an eigenvalue of an unknown unitary operation, given as a black-box. This is an abstract problem that turns out to be a powerful algorithmic primitive.

4.1 A simple introductory example

Let U be an n-qubit unitary and $|\psi\rangle$ be an eigenvector of U with eigenvalue either $+1$ or -1. Suppose that we're given a controlled-U gate as a black-box

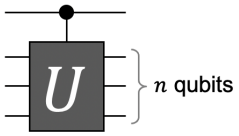

Figure 23: A controlled-U gate as a black box.

and we are also given n qubits that are in state $|\psi\rangle$. We're given no other information about U. Our controlled-U gate is essentially a black-box. Also, we're given no information about what state $|\psi\rangle$ is. All we know is that $|\psi\rangle$ is an eigenvector of U with eigenvalue $\lambda \in \{+1, -1\}$. Our goal is to determine whether λ is $+1$ or -1.

It turns out that we can solve this problem with just one single query to the controlled-U gate. I'd like to you to think about how this can be done. What state should you set the control qubit to? What state should you set the n target qubits to? Now is a good time to stop reading and think about this ...

So how did it go? Did you come up with a quantum circuit for this? If you did not then I'd like to urge you to try again. The solution is a pretty simple circuit. And here is a hint: it's similar to the very first quantum algorithm that we saw, for Deutsch's problem. So please give it another shot ...

(A solution is on the next page.)

Here's a quantum circuit that works.

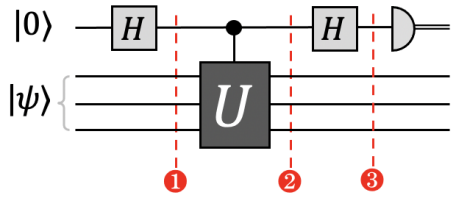

Figure 24: Quantum circuit determining the eigenvalue of U in the special case.

Let's trace through this to see how it works.

State at stage ❶

Applying H to the control qubit results in the state

$$\left(\tfrac{1}{\sqrt{2}}\left|0\right\rangle + \tfrac{1}{\sqrt{2}}\left|1\right\rangle\right)\left|\psi\right\rangle = \tfrac{1}{\sqrt{2}}\left|0\right\rangle\left|\psi\right\rangle + \tfrac{1}{\sqrt{2}}\left|1\right\rangle\left|\psi\right\rangle. \tag{59}$$

State at stage ❷

After the controlled-U gate, the state is

$$\tfrac{1}{\sqrt{2}}\left|0\right\rangle\left|\psi\right\rangle + \tfrac{1}{\sqrt{2}}\left|1\right\rangle U\left|\psi\right\rangle = \tfrac{1}{\sqrt{2}}\left|0\right\rangle\left|\psi\right\rangle + \tfrac{1}{\sqrt{2}}\left|1\right\rangle\lambda\left|\psi\right\rangle \tag{60}$$

$$= \left(\tfrac{1}{\sqrt{2}}\left|0\right\rangle + \tfrac{1}{\sqrt{2}}\lambda\left|1\right\rangle\right)\left|\psi\right\rangle. \tag{61}$$

Notice that, since $\lambda \in \{+1, -1\}$, the control qubit is in either the state $\left|+\right\rangle$ or state $\left|-\right\rangle$ (depending in λ).

State at stage ❸

It follows that, when H is applied again to the control qubit becomes

$$\begin{cases} \left|0\right\rangle & \text{if } \lambda = +1 \\ \left|1\right\rangle & \text{if } \lambda = -1. \end{cases} \tag{62}$$

We have just solved a special case of the *phase estimation problem*.

I will show you how the general case is defined—and then we'll see that it turns out not to be not that much harder than this case to solve. In order to define the phase estimation problem in full generality, we first need to extend our notion of a controlled-U gate.

4.2 Multiplicity controlled-U gates

Let's begin with a review of what a standard controlled-U gate is.

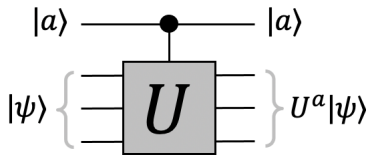

Figure 25: Controlled-U gate.

The unitary matrix is the block matrix

$$\begin{bmatrix} I & 0 \\ 0 & U \end{bmatrix}. \tag{63}$$

When the control qubit is in computational basis state $|a\rangle$ and the target qubit is in state $|\psi\rangle$, we can write the output state of the target qubit succinctly as $U^a |\psi\rangle$ (where $U^0 = I$ means "do nothing", and $U^1 = U$ means "apply U"). So, in the computational basis, the control qubit is a number which indicates how many times U should be applied to the target: 0 times or 1 time.

We can define a more general type of controlled-U gate where there are ℓ control qubits, and the number of times that U is applied is an ℓ-bit integer.

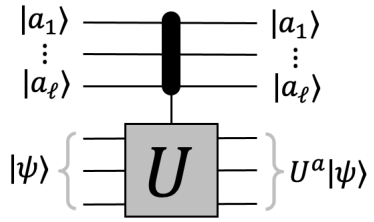

Figure 26: Multiplicity-controlled-U gate.

In this case, U can be applied zero times, U can be applied once, twice, three times, all the way up to $2^\ell - 1$ times. For example, if $\ell = 5$ and the control qubits are in state $|11010\rangle$ then U gets applied 26 times. Why 26? because 11010 is the number 26 in binary.

The unitary matrix of this kind of controlled-U gate is the block matrix

$$\begin{bmatrix} I & 0 & 0 & \cdots & 0 \\ 0 & U & 0 & \cdots & 0 \\ 0 & 0 & U^2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & U^{2^\ell - 1} \end{bmatrix}. \tag{64}$$

Let's call this a *multiplicity-controlled-U gate*. In the computational basis, the state of the control qubits specifies how many times U should be applied.

As an aside, I want to distinguish this kind of gate from a different generalization of a controlled-U that we saw previously in the Toffoli gate.

4.2.1 Aside: multiplicity-control gates vs. AND-control gates

For the Toffoli gate, there are two control qubits and the unitary U is the Pauli X gate (a.k.a. NOT gate). For the Toffoli gate, the NOT is applied to the target qubit if both control qubits are $|1\rangle$, and nothing happens for the other computational basis states $|00\rangle$, $|01\rangle$, $|10\rangle$.

Our notation distinguishes between these controlled gates and our multiplicity-controlled gate.

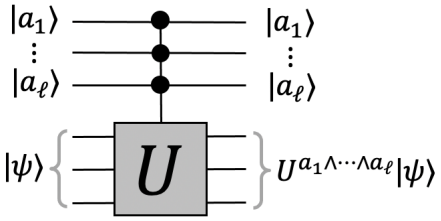

Figure 27: AND-controlled-U gate.

When we have a separate *dot* at each control qubit, that means that the U gets applied once if *all* control qubits are in state $|1\rangle$ and, for other computational basis states, nothing happens. So the unitary matrix is the block matrix

$$\begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ 0 & I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 \\ 0 & 0 & \cdots & 0 & U \end{bmatrix}. \tag{65}$$

Our multiplicity-controlled-U gates are denoted differently, with *one single dot*, that's stretched out so as to cover all of the control qubits (figure 26).

4.3 Definition of the phase estimation problem

Now, we can define the phase estimation problem in generality.

Let U be an arbitrary unknown unitary operation acting on n qubits. Let $|\psi\rangle$ be an eigenvector of U. But now the eigenvalue is not restricted to being $+1$ or -1. The eigenvalue can be any complex number on the unit circle. So the eigenvalue is of the form $e^{2\pi i\varphi}$, where φ can be any element of the interval $[0, 1]$.

Definition 4.1 (Phase estimation problem)**.** *In the* phase estimation problem *we are given: a black-box for a multiplicity-controlled-U gate with ℓ control qubits and*

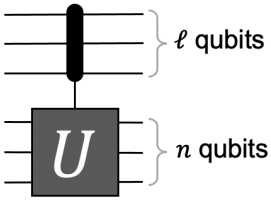

Figure 28: Caption.

one copy of state $|\psi\rangle$, which is an eigenvector of U with eigenvalue $e^{2\pi i\varphi}$ for some $\varphi \in [0, 1]$. The goal is to determine an ℓ-bit approximation of the value of φ. (Since φ can take on a continuum of values, determining it exactly *is unreasonable.)*

There's an *exact case* of this where φ is an ℓ-bit binary fraction. This case is simpler to work with than the general case. An ℓ-bit binary fraction is a rational number with 2^ℓ in the denominator and an integer $a \in \{0, 1, \ldots, 2^\ell - 1\}$ in the numerator. In other words, the binary representation of φ can be written exactly with only ℓ bits occurring after the radix point. For example, for $\ell = 6$,

$$\frac{19}{2^6} = 0.010011. \tag{66}$$

The *general case* is where φ is arbitrary. For example, if $\varphi = \frac{1}{3}$ then

$$\phi = 0.\overline{01} = 0.010101010101\ldots \tag{67}$$

(where the pattern repeats forever). That's not a binary fraction for any ℓ. The 8-bit approximation of this number is 0.01010101 (it's $\frac{1}{3}$ rounded down to the closest

8-bit binary fraction). The 7-bit approximation of this number is 0.0101011 (note that last bit is 1, not 0, because we round *up* to the closest 7-bit binary fraction. We always round φ towards the ℓ-bit binary fraction that's closest. Sometimes that means rounding down and sometimes that means rounding up.

For our applications, we will want to solve the general case of phase estimation, but it's conceptually useful to to first solve this problem in the exact case.

4.4 Solving the exact case

Here we will solve the phase estimation problem in the exact case—which turns out to be quite easy. In the exact case,

$$\varphi = \frac{a}{2^\ell} = 0.a_1 a_2 \dots a_\ell \,, \tag{68}$$

where $a \in \{0, 1, \dots, 2^\ell - 1\}$. Note that the eigenvalue can be written as

$$e^{2\pi i \varphi} = e^{2\pi i a/2^\ell} = \left(e^{2\pi i/2^\ell}\right)^a = \omega^a, \tag{69}$$

where ω is a primitive 2^ℓ-th root of unity. It clarifies things to think of the eigenvalue as ω^a in this manner. Note that the goal of determining the eigenvalue parameter φ is now equivalent to determining the value of the number a.

We'll start by putting the control qubits into a uniform superposition of all computational basis sates on ℓ qubits, which is accomplished by ℓ Hadamard transforms. And then we'll perform the multiplicity-controlled-U gate.

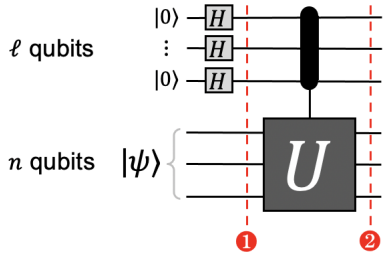

Figure 29: Caption.

Let's determine the output state of this quantum circuit.

State at stage ❶

The $H^{\otimes \ell}$ operation on $|0\rangle^{\otimes \ell}$ produces that state

$$\Big(|0...00\rangle + |0...01\rangle + |0...10\rangle + |0...11\rangle + \cdots + |1...11\rangle\Big)\,|\psi\rangle$$
$$= \Big(|0\rangle + |1\rangle + |2\rangle + |3\rangle + \cdots + \big|2^\ell - 1\big\rangle\Big)\,|\psi\rangle\,. \tag{70}$$

State at stage ❷

The multiplicity-controlled-U gate changes the state to

$$|0\rangle\,|\psi\rangle + |1\rangle\,U\,|\psi\rangle + |2\rangle\,U^2\,|\psi\rangle + |3\rangle\,U^3\,|\psi\rangle + \cdots + \big|2^\ell - 1\big\rangle\,U^{2^\ell - 1}\,|\psi\rangle$$
$$= |0\rangle\,|\psi\rangle + |1\rangle\,\omega^a\,|\psi\rangle + |2\rangle\,\omega^{2a}\,|\psi\rangle + |3\rangle\,\omega^{3a}\,|\psi\rangle + \cdots + \big|2^\ell - 1\big\rangle\,\omega^{(2^\ell - 1)a}\,|\psi\rangle \tag{71}$$
$$= \Big(|0\rangle + \omega^a\,|1\rangle + \omega^{2a}\,|2\rangle + \omega^{3a}\,|3\rangle + \cdots + \omega^{(2^\ell - 1)a}\,\big|2^\ell - 1\big\rangle\Big)\,|\psi\rangle\,. \tag{72}$$

Now, do you recognize this state of the first ℓ qubits? Haven't you seen the state $|0\rangle + \omega^a\,|1\rangle + \omega^{2a}\,|2\rangle + \cdots + \omega^{(2^\ell - 1)a}\,\big|2^\ell - 1\big\rangle$ before?

It's the Fourier basis state $F_{2^\ell}\,|a\rangle$. Remember that our goal is to determine a. The fact that the Fourier basis states for all the potential values of parameter a are orthogonal is a good sign. It means that they are distinguishable in principle. But we can also explicitly compute the value of a using a polynomial number of gates. Can you see how?

We can compute a by applying the the *inverse Fourier transform*. If we apply $F^*_{2^\ell}$ to $F_{2^\ell}\,|a\rangle$, the result is $|a\rangle$. So our phase estimation algorithm for the exact case is the quantum circuit in figure 30.

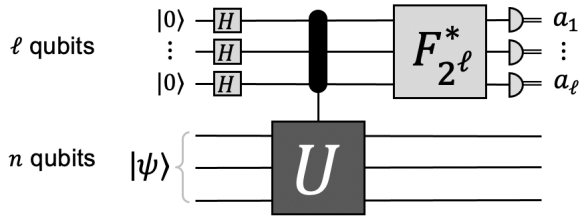

Figure 30: Quantum algorithm for phase estimation in the exact case.

The output of the ℓ measured qubits is a, the numerator of the binary fraction for φ, or, equivalently, the ℓ bits of the binary representation of φ.

Note that the algorithm makes only one query to the multiplicity-controlled-U gate, and all the other operations in the algorithm (the grey boxes) can be implemented with a polynomial number of elementary gates (with respect to ℓ, the number of control qubits). The dominant cost is that of computing $F^*_{2^\ell}$, which costs $O(\ell^2)$ elementary gates (section 3).

By the way, the $\ell = 1$ version of the exact case is that simple example that we solved in section 4.1 (and in that case $\omega = -1$).

4.5 Solving the general case

Now, what happens if we use this same algorithm for the general case, where φ can be *any* real number between 0 and 1? Recall that, in that case, we need to determine the ℓ-bit binary number that best approximates the true value of φ.

We can write φ as an ℓ-bit binary fraction plus a quantity δ that represents the remaining bits that get rounded up or down. More precisely,

$$\varphi = \frac{a}{2^\ell} + \delta, \tag{73}$$

where $a \in \{0, 1, \ldots, 2^\ell - 1\}$ and

$$|\delta| \leq \frac{1}{2^{\ell+1}}. \tag{74}$$

If φ gets rounded down then δ is positive; if φ gets rounded up then δ is negative. If $\delta = 0$, we are in the exact case. Let's analyze what the circuit that we used for the exact case does in the case where $\delta \neq 0$.

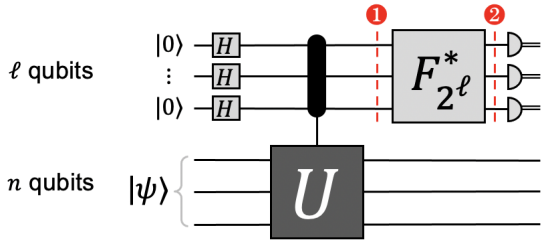

Figure 31: Quantum algorithm for phase estimation applied in the general case.

Since $|\psi\rangle$ is an eigenvector of U, the state of the second register (the last n qubits) does not change. All the action is with the first register (the first ℓ qubits). We trace through the evolution of the first ℓ qubits.

State at stage ❶

After the multiplicity-controlled-U gate, the state of the first ℓ qubits is

$$\frac{1}{\sqrt{2^\ell}} \sum_{b=0}^{2^\ell-1} \left(e^{2\pi i\varphi}\right)^b |b\rangle = \frac{1}{\sqrt{2^\ell}} \sum_{b=0}^{2^\ell-1} e^{2\pi i\left(\frac{a}{2^\ell}+\delta\right)b} |b\rangle \tag{75}$$

$$= \frac{1}{\sqrt{2^\ell}} \sum_{b=0}^{2^\ell-1} \omega^{ab} e^{2\pi i\delta b} |b\rangle\,, \tag{76}$$

where $\omega = e^{2\pi i/2^\ell}$. The effect of δ is highlighted in red. If $\delta = 0$ then $e^{2\pi i\delta b} = 1$ and the expression is $F_{2^\ell} |a\rangle$, which is consistent with what we obtained for the exact case.

State at stage ❷

After applying the inverse Fourier transform, the state becomes

$$F_{2^\ell}^* \left(\frac{1}{\sqrt{2^\ell}} \sum_{b=0}^{2^\ell-1} \omega^{ab} e^{2\pi i\delta b} |b\rangle \right) = \frac{1}{\sqrt{2^\ell}} \sum_{b=0}^{2^\ell-1} \omega^{ab} e^{2\pi i\delta b} \left(\frac{1}{\sqrt{2^\ell}} \sum_{c=0}^{2^\ell-1} \omega^{-bc} |c\rangle \right) \tag{77}$$

$$= \frac{1}{2^\ell} \sum_{c=0}^{2^\ell-1} \left(\sum_{b=0}^{2^\ell-1} \omega^{b(a-c)} e^{2\pi i\delta b} \right) |c\rangle\,. \tag{78}$$

Now, recall that the correct outcome for the phase estimation problem is a (the bits of an ℓ-bit approximation of φ). What's the probability that measuring the state in Eq. (78) results in outcome a? To figure this out, we look at the amplitude of the $|a\rangle$ term in this expression. Notice that if we substitute a for c then the factor $\omega^{b(a-c)}$ simplifies to $\omega^0 = 1$. So the amplitude of the $|a\rangle$ term in Eq. (78) is the sum

$$\frac{1}{2^\ell} \sum_{b=0}^{2^\ell-1} e^{2\pi i\delta b}. \tag{79}$$

As a reality check, what is this sum in the exact case, where $\delta = 0$? Can you see why it's 1? This makes sense, because, for the exact case, we've already seen that the measurement outcome is a with probability 1.

Returning to our case of interest, where $0 \neq |\delta| \leq 1/2^{\ell+1}$, we're interested in the absolute value squared of the amplitude in Eq. (79). At first glance, the sum may look a little daunting, but there's a closed-form expression for it. Can you see what it is?

The sum in Eq. (79) is a geometric series.[3] Therefore, we can use the formula for the sum of a geometric series to obtain

$$\frac{1}{2^\ell}\sum_{b=0}^{2^\ell-1} e^{2\pi i\delta b} = \frac{1}{2^\ell}\frac{1-(e^{2\pi i\delta})^{2^\ell}}{1-e^{2\pi i\delta}}. \tag{80}$$

What we're going to do next is use some simple geometric reasoning to show that the absolute value of this expression is at least $\frac{2}{\pi}$.

Lemma 4.1. *If δ is such that $0 \neq |\delta| \leq 1/2^{\ell+1}$ then*

$$\frac{1}{2^\ell}\left|\frac{1-(e^{2\pi i\delta})^{2^\ell}}{1-e^{2\pi i\delta}}\right| \geq \frac{2}{\pi}. \tag{81}$$

Proof. To lower-bound the expression, we'll show that the numerator is not too small and the denominator is not too large. We give the proof for the case where $\delta > 0$ (the case where $\delta < 0$ is similar, with upside-down versions of figures 32 and 33).

First, let's show that the denominator is not too large. Here are 1 and $e^{2\pi i\delta}$ as points in the complex plane.

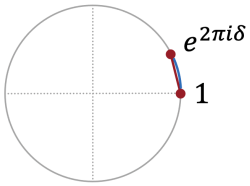

Figure 32: Geometric view of $|1-e^{2\pi i\delta}|$ as the distance between two points in $\mathbb{C}$.

The length of the red line segment is $|1-e^{2\pi i\delta}|$. The length of the red line is upper bounded by the arc-length between the two points, which is $2\pi\delta$. Therefore, we can upper bound of the denominator as

$$|1-e^{2\pi i\delta}| \leq 2\pi\delta. \tag{82}$$

Next, we'll lower bound the numerator. In this case, $|1-e^{2\pi i\delta 2^\ell}|$ is the length of the red line.

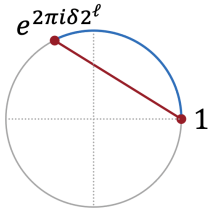

Figure 33: Geometric view of $|1-e^{2\pi i\delta 2^\ell}|$ as the distance between two points in $\mathbb{C}$.

[3]For $s \neq 1$, the formula for the sum of the geometric series $1+s+s^2+\cdots+s^{m-1} = \frac{1-s^m}{1-s}$.

Let's also consider the arc length between the two points, which is $2\pi\delta 2^\ell$. Note that the arc-length is *not* a lower bound of the distance—it's longer than the line segment. Think of the arc-length as the length of the line times some stretch-factor. How big is the stretch-factor? If the line segment is short then stretch-factor is just slightly larger than 1 (in that case, the line and arc have almost the same lengths).

But the line and arc need not be short. The extremal case is where δ is as large as possible: $\delta = 1/2^{\ell+1}$. In that case, $(e^{2\pi i\delta})^{2^\ell} = e^{\pi i} = -1$, so the length of the line is 2 and the length of the arc is π. Thus, the maximum possible stretch-factor is $\pi/2$.

Therefore, we can lower bound the length of the line by the arc-length times $2/\pi$. Multiplying the arc-length by the reciprocal of the maximum stretch-factor compensates for how much longer the arc-length can be than the line segment. Therefore, the numerator lower bounded as

$$|1 - e^{2\pi i\delta 2^\ell}| \geq 2\pi\delta 2^\ell\Big(\frac{2}{\pi}\Big) = 4\delta 2^\ell. \tag{83}$$

Now, we can substitute in our bounds in Eq. (82) and Eq. (83) for the numerator and the denominator to obtain

$$\frac{1}{2^\ell}\left|\frac{1-(e^{2\pi i\delta})^{2^\ell}}{1-e^{2\pi i\delta}}\right| \geq \frac{1}{2^\ell}\frac{4\delta 2^\ell}{2\pi\delta} = \frac{2}{\pi}. \tag{84}$$

□

This means that, if we measure (in the computational basis), we get the correct answer a with probability at least $4/\pi^2 = 0.405...$ (squaring the amplitude). That's slightly more than 40%. This might not seem like a very high success probability. But, for our purposes, what's important is that it's a *constant* (independent of ℓ and n). In the contexts where we will use phase estimation to achieve something else (such as to factor a number), we will be able to amplify the success probability by repeating the entire process a few times.

In summary, for the phase estimation problem, you are given an multiplicity-controlled-U gate with ℓ control-qubits and a state $|\psi\rangle$ that's an eigenvector of U with eigenvalue $e^{2\pi i\varphi}$. Your goal is to find an ℓ-bit approximation of φ. The following quantum circuit solves this problem with success probability at least $\frac{4}{\pi^2} = 0.405\ldots$.

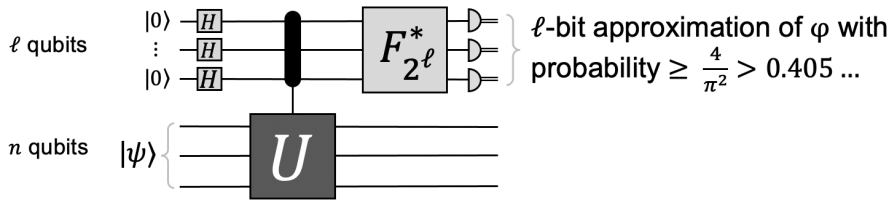

Figure 34: Summary of phase-estimation algorithm for approximating the eigenvalue $e^{2\pi i\varphi}$ of $|\psi\rangle$.

The algorithm makes one query to the multiplicity-controlled-U gate and has $O(\ell^2)$ elementary gates (the dominant cost being the implementation of $F^*_{2^\ell}$).

4.6 The case of superpositions of eigenvectors

Before ending this section, I'd like to bring your attention to a slightly different scenario. Suppose that, instead of being given an eigenvector of U, we're given a superposition of two eigenvectors with different eigenvalues.

Suppose that we're provided with a state of the form $\alpha_1 |\psi_1\rangle + \alpha_2 |\psi_2\rangle$, where:

- $|\psi_1\rangle$ is an eigenvector of U with eigenvalue $e^{2\pi i\varphi_1}$,
- $|\psi_2\rangle$ is an eigenvector of U with eigenvalue $e^{2\pi i\varphi_2}$,
- $|\alpha_1|^2 + |\alpha_2|^2 = 1$, and $\varphi_1 \neq \varphi_2$.

What happens if we run our phase estimation algorithm with this state in the target register?

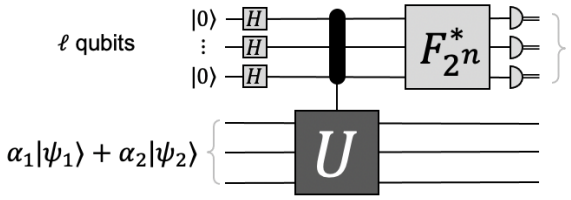

Figure 35: Scenario where input to phase algorithm is a superposition of two eigenvectors.

Exercise 4.1. *For the exact case, where φ_1 and φ_2 are both ℓ-bit binary integers, show that the output of the circuit in figure 35 is*

$$\begin{cases} \varphi_1 & \text{with probability } |\alpha_1|^2 \\ \varphi_2 & \text{with probability } |\alpha_2|^2. \end{cases} \tag{85}$$

So the net result is exactly the same as what occurs if, instead of a superposition of eigenvalues, we had just randomly selected one of the eigenvectors as

$$\begin{cases} |\psi_1\rangle & \text{with probability } |\alpha_1|^2 \\ |\psi_2\rangle & \text{with probability } |\alpha_2|^2 \end{cases} \tag{86}$$

and then input the selected eigenvector to the target register.

That's for the exact case. For the general case, it's similar, with the statement of the result qualified as "with success probability at least $4/\pi^2$". Also, although I've described the case of a superposition of *two* eigenvectors, there's nothing so special about two. There are similar results for the case of superpositions of more than two eigenvectors.